MacApp™ 2.0b5 Source Listings

The following page is a letter from Apple's Software Licensing department which discusses distribution of MacApp. Please note that the source code in these listings is provided for your own use—this package does not give you any rights to distribute them in any form.

The MacApp 2.0b5 Source Listings consists of 482 pages of cross-referenced source listings which were created using PasRef, an MPW tool included with the MPW Pascal product. The cross-reference includes the following source files of MacApp and MacApp building blocks (which can be found on the disks contained in the MacApp 2.0b5 product):

Each line in the source files is preceded by the number of the file it came from (using the above numbers) and the line number within that file.

The cross-reference contains an alphabetical list of all identifiers used in the source files. Following each identifier is a list of the line number with file numbers in parentheses. An asterisk (*) following a line number indicates a definition of the variable or routine. An equal sign (=) indicates an assignment. A line number with nothing following it means a reference to an identifier.

Note that the following assembly language and MacApp debugger source files were not included in the listing (interested users can list them from the files on the MacApp 2.0b5 disks):

CommonObjLib.a
Macros.a
UBusyCursor.a
UFailure.a
UInspector.p
UInspector.inc1.p
UMAUtil.a
UObject.a
UTrace.a
UTrace.p
UTrace.inc1.p
UViewCoords.a
UWriteInWindow.p
UWriteInWindow.inc1.p

Also not included in this listing are the MacApp resource files, make files, and sample programs.

Dear MacApp User:

MacApp™ is a copyrighted program owned by Apple Computer, Inc. and is designed for combination with Macintosh™ programs written by you. However, this copy of the MacApp Source Listings does not give you any rights to redistribute MacApp in any form.

If you wish to distribute programs which include object code from the MacApp library or building blocks you will need to complete a MacApp Object Code Distribution Agreement and return it with the appropriate annual licensing fee to Apple's Software Licensing Department. A copy of that agreement is included in the MacApp package; it is also available by writing to:

Software Licensing Department
Mailstop 28B
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014

The annual fee is $100.00 if any of your applications are distributed commercially (i.e. in an effort to earn a profit) or $10.00 if all of your applications are distributed non-commercially. The Software Licensing Department does provide commercial licensees with upgrades to new versions of software, but no updates will be provided to holders of the non-commercial distribution license.

Once you have returned the Agreement and fee, you may distribute any number of different applications for one year, subject to the terms and conditions of the Agreement. The only requirement is that you send a list of any additional applications to Apple's Software Licensing Department within 90 days of when you first ship the applications. This may be sent via mail or to SW.LICENSE on AppleLink™.

Apple will not support or answer questions from end-users of your applications, so you should provide user support for applications you build using MacApp. Problems with the MacApp software should be reported to Apple through a designated contact at your company or institution.

If you have any questions, please contact us at (408) 973-4667.

Sincerely,

Software Licensing

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
408 996-1010
TLX 171-576

```
1     1 --    {$P}
1     2 --    {UAssociation.p}
1     3 --    {Copyright © 1988 by Apple Computer Inc.  All rights reserved.}
1     4 --
1     5 --    UNIT UAssociation;
1     6 --
1     7 --    INTERFACE
1     8 --
1     9 --    {-----------------------------------+
1    10 --    |   Uses                            |
1    11 --    +-----------------------------------}
1    12 --    USES
1    13 --        {$LOAD MacIntf.LOAD}
1    14 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
1    15 --        {$LOAD}
1    16 --            UMAUtil,
1    17 --            UFailure,
1    18 --            UObject,
1    19 --            UList;
1    20 --
1    21 --    {-----------------------------------+
1    22 --    |   Types                           |
1    23 --    +-----------------------------------}
1    24 --    TYPE
1    25 --        {-----------------------------------+
1    26 --        |   TEntry                          |
1    27 --        +-----------------------------------}
1    28 --        TEntry = OBJECT(TObject)
1    29 --
1    30 --            fKey:           StringHandle;
1    31 --            fValue:         StringHandle;
1    32 --
1    33 --            PROCEDURE TEntry.IEntry (itsKey, itsValue: Str255);
1    34 --
1    35 --            PROCEDURE TEntry.Free; OVERRIDE;
1    36 --
1    37 --    {$IFC qDebug}
1    38 --            PROCEDURE TEntry.Fields (PROCEDURE DoToField (fieldName: Str255;
1    39 --                                                          fieldAddr: Ptr;
1    40 --                                                          fieldType: INTEGER)); OVERRIDE;
1    41 --    {$ENDC}
1    42 --
1    43 --            END {TEntry};
1    44 --
1    45 --
1    46 --        {-----------------------------------+
1    47 --        |   TEntriesList                    |
1    48 --        +-----------------------------------}
1    49 --        TEntriesList = OBJECT(TSortedList)
1    50 --
1    51 --            FUNCTION  TEntriesList.Compare (item1, item2: TObject): INTEGER; OVERRIDE;
1    52 --
1    53 --            END {TEntriesList};
1    54 --
1    55 --
1    56 --        {-----------------------------------+
1    57 --        |   TAssociation                    |
1    58 --        +-----------------------------------}
1    59 --        TAssociation = OBJECT(TObject)
1    60 --
1    61 --            fEntries:       TEntriesList;
1    62 --
1    63 --            PROCEDURE TAssociation.IAssociation;
1    64 --
1    65 --            PROCEDURE TAssociation.Free; OVERRIDE;
1    66 --
1    67 --            FUNCTION  TAssociation.ValueAt (keyStr: Str255; VAR valueStr: Str255): BOOLEAN;
1    68 --
1    69 --            FUNCTION  TAssociation.KeyAt (valueStr: Str255; VAR keyStr: Str255): BOOLEAN;
1    70 --
1    71 --            PROCEDURE TAssociation.EachEntryDo (PROCEDURE DoToEntry (theEntry: TEntry));
1    72 --
1    73 --            FUNCTION  TAssociation.EntryWithKey (keyStr: Str255): TEntry;
1    74 --
1    75 --            FUNCTION  TAssociation.EntryWithValue (valueStr: Str255): TEntry;
1    76 --
1    77 --            FUNCTION  TAssociation.FirstEntryThat (FUNCTION TestEntry (theEntry: TEntry): BOOLEAN): TEntry;
1    78 --
1    79 --            PROCEDURE TAssociation.InsertEntry (keyStr, valueStr: Str255);
1    80 --
1    81 --            PROCEDURE TAssociation.RemoveValueAt (keyStr: Str255);
1    82 --
1    83 --            PROCEDURE TAssociation.RemoveKeyAt (valueStr: Str255);
1    84 --
1    85 --    {$IFC qDebug}
1    86 --            PROCEDURE TAssociation.Fields (PROCEDURE DoToField (fieldName: Str255;
1    87 --                                                                fieldAddr: Ptr;
1    88 --                                                                fieldType: INTEGER)); OVERRIDE;
1    89 --    {$ENDC}
1    90 --
1    91 --            END {TAssocation};
1    92 --
1    93 --
1    94 --    IMPLEMENTATION
1    95 --
1    96 --    {$I UAssociation.incl.p}
```

```
2    1 --     {$P}
2    2 --     {UAssociation.incl.p}
2    3 --     {Copyright © 1988 by Apple Computer Inc.  All rights reserved.}
2    4 --
2    5 --     {$S MAMain}          { Everything goes in here}
2    6 --
2    7 --     (*****************************************************************************************)
2    8 --     (*      T E n t r y                                                                      *)
2    9 --     (*****************************************************************************************)
2   10 --     {-----------------------------------+
2   11 --     |   IEntry                          |
2   12 --     +-----------------------------------}
2   13 -- A   PROCEDURE TEntry.IEntry (itsKey, itsValue: Str255);
2   14 --
2   15 --     VAR
2   16 --         fi:         FailInfo;
2   17 --
2   18 --         {-----------------------------------+
2   19 --         |   HdlIEntry                       |
2   20 --         +-----------------------------------}
2   21 -- B       PROCEDURE HdlIEntry (error: OSErr; message: LONGINT);
2   22 --
2   23 0- B       BEGIN
2   24 --             DisposIfHandle(fKey);
2   25 --             Free;
2   26 -0 B       END;
2   27 --
2   28 0- A   BEGIN
2   29 --         CatchFailures(fi, HdlIEntry);
2   30 --
2   31 --         fKey := NewString(itsKey);
2   32 --         FailNIL(fKey);
2   33 --
2   34 --         fValue := NewString(itsValue);
2   35 --         FailNIL(fValue);
2   36 --
2   37 --         Success(fi);
2   38 -0 A   END {TEntry.IEntry};
2   39 --
2   40 --     {-----------------------------------+
2   41 --     |   Free                            |
2   42 --     +-----------------------------------}
2   43 -- A   PROCEDURE TEntry.Free; OVERRIDE;
2   44 --
2   45 0- A   BEGIN
2   46 --         DisposIfHandle(fKey);
2   47 --         DisposIfhandle(fValue);
2   48 --         INHERITED Free;
2   49 -0 A   END {TEntry.Free};
2   50 --
2   51 --     {$IFC qDebug}
2   52 --         {-----------------------------------+
2   53 --         |   Fields                          |
2   54 --         +-----------------------------------}
2   55 -- A       PROCEDURE TEntry.Fields (PROCEDURE DoToField (fieldName: Str255;
2   56 --                                                       fieldAddr: Ptr;
2   57 --                                                       fieldType: INTEGER)); OVERRIDE;
2   58 0- A       BEGIN
2   59 --             DoToField('TEntry', NIL, bClass);
2   60 --             DoToField('fKey', @fKey, bStringHandle);
2   61 --             DoToField('fValue', @fValue, bStringHandle);
2   62 --             INHERITED Fields(DoToField);
2   63 -0 A       END {TEntry.Fields};
2   64 --     {$ENDC}
2   65 --
2   66 --
2   67 --     (*****************************************************************************************)
2   68 --     (*      T E n t r y                                                                      *)
2   69 --     (*****************************************************************************************)
2   70 --     {-----------------------------------+
2   71 --     |   Compare                         |
2   72 --     +-----------------------------------}
2   73 -- A   FUNCTION  TEntriesList.Compare (item1, item2: TObject): INTEGER;
2   74 --
2   75 0- A   BEGIN
2   76 --         IF TEntry(item1).fKey^^ < TEntry(item2).fKey^^ THEN
2   77 --             Compare := kALessThanB
2   78 --         ELSE IF TEntry(item1).fKey^^ > TEntry(item2).fKey^^ THEN
2   79 --             Compare := kAGreaterThanB
2   80 --         ELSE
2   81 --             Compare := 0;
2   82 -0 A   END {TEntriesList.Compare};
2   83 --
2   84 --
2   85 --     (*****************************************************************************************)
2   86 --     (*      T A s s o c i a t i o n                                                          *)
2   87 --     (*****************************************************************************************)
2   88 --     {-----------------------------------+
2   89 --     |   IAssociation                    |
2   90 --     +-----------------------------------}
2   91 -- A   PROCEDURE TAssociation.IAssociation;
2   92 --
2   93 --     VAR
2   94 --         anEntriesList:  TEntriesList;
2   95 --
2   96 0- A   BEGIN
2   97 --         New(anEntriesList);
2   98 --         FailNIL(anEntriesList);
```

```
2   99 --          anEntriesList.IList;
2  100 --          fEntries := anEntriesList;
2  101 --      {$IFC qDebug}
2  102 --          fEntries.SetEltType('TEntry');
2  103 --      {$ENDC}
2  104 -0 A    END {TAssociation.IAssociation};
2  105 --
2  106 --      {-----------------------------------+
2  107 --      |   Free                            |
2  108 --      +-----------------------------------}
2  109 -- A    PROCEDURE TAssociation.Free; OVERRIDE;
2  110 --
2  111 --          {-----------------------------------+
2  112 --          |   FreeEntry                       |
2  113 --          +-----------------------------------}
2  114 -- B        PROCEDURE FreeEntry (theEntry: TEntry);
2  115 --
2  116 0- B        BEGIN
2  117 --              theEntry.Free;
2  118 -0 B        END;
2  119 --
2  120 0- A    BEGIN
2  121 --          EachEntryDo(FreeEntry);
2  122 --          fEntries.Free;
2  123 --          INHERITED Free;
2  124 -0 A    END {TAssociation.Free};
2  125 --
2  126 --      {-----------------------------------+
2  127 --      |   EachEntryDo                     |
2  128 --      +-----------------------------------}
2  129 -- A    PROCEDURE TAssociation.EachEntryDo (PROCEDURE DoToEntry (theEntry: TEntry));
2  130 --
2  131 0- A    BEGIN
2  132 --          fEntries.Each(DoToEntry);
2  133 -0 A    END {TAssociation.EachEntryDo};
2  134 --
2  135 --
2  136 --      {-----------------------------------+
2  137 --      |   EntryWithKey                    |
2  138 --      +-----------------------------------}
2  139 -- A    FUNCTION  TAssociation.EntryWithKey (keyStr: Str255): TEntry;
2  140 --
2  141 --          {-----------------------------------+
2  142 --          |   TestKey                         |
2  143 --          +-----------------------------------}
2  144 -- B        FUNCTION  TestKey (theEntry: TObject): INTEGER;
2  145 --
2  146 0- B        BEGIN
2  147 --              IF keyStr < TEntry(theEntry).fKey^^ THEN
2  148 --                  TestKey := kALessThanB
2  149 --              ELSE IF keyStr > TEntry(theEntry).fKey^^ THEN
2  150 --                  TestKey := kAGreaterThanB
2  151 --              ELSE
2  152 --                  TestKey := kAEqualB;
2  153 -0 B        END;
2  154 --
2  155 0- A    BEGIN
2  156 --          EntryWithKey := TEntry(fEntries.Search(TestKey));
2  157 -0 A    END;
2  158 --
2  159 --      {-----------------------------------+
2  160 --      |   EntryWithValue                  |
2  161 --      +-----------------------------------}
2  162 -- A    FUNCTION  TAssociation.EntryWithValue (valueStr: Str255): TEntry;
2  163 --
2  164 --          {-----------------------------------+
2  165 --          |   MatchesValue                    |
2  166 --          +-----------------------------------}
2  167 -- B        FUNCTION  MatchesValue (theEntry: TEntry): BOOLEAN;
2  168 --
2  169 0- B        BEGIN
2  170 --              MatchesValue := valueStr = theEntry.fValue^^;
2  171 -0 B        END;
2  172 --
2  173 0- A    BEGIN
2  174 --          EntryWithValue := FirstEntryThat(MatchesValue);
2  175 -0 A    END;
2  176 --
2  177 --
2  178 --      {-----------------------------------+
2  179 --      |   FirstEntryThat                  |
2  180 --      +-----------------------------------}
2  181 -- A    FUNCTION  TAssociation.FirstEntryThat (FUNCTION TestEntry (theEntry: TEntry): BOOLEAN): TEntry;
2  182 --
2  183 0- A    BEGIN
2  184 --          FirstEntryThat := TEntry(fEntries.FirstThat(TestEntry));
2  185 -0 A    END {TAssociation.FirstEntryThat};
2  186 --
2  187 --      {-----------------------------------+
2  188 --      |   InsertEntry                     |
2  189 --      +-----------------------------------}
2  190 -- A    PROCEDURE TAssociation.InsertEntry (keyStr, valueStr: Str255);
2  191 --
2  192 --      VAR
2  193 --          anEntry:        TEntry;
2  194 --          fi:             FailInfo;
2  195 --
2  196 --          {-------------------------------+
2  197 --          |   HdlSetString                |
```

```
2  198 --          +-------------------------------}
2  199 -- B       PROCEDURE HdlSetString (error: OSErr; message: LONGINT);
2  200 --
2  201 0- B       BEGIN
2  202 --              {If we couldn't increase the handle size, set it to something reasonable}
2  203 --              anEntry.fValue^^ := '';
2  204 -0 B       END;
2  205 --
2  206 --          {-------------------------------+
2  207 --          |   TestKey                     |
2  208 --          +-------------------------------}
2  209 -- B       FUNCTION  TestKey (theEntry: TEntry): BOOLEAN;
2  210 --
2  211 0- B       BEGIN
2  212 --              TestKey := theEntry.fKey^^ = keyStr;
2  213 -0 B       END;
2  214 --
2  215 0- A   BEGIN
2  216 --          anEntry := EntryWithKey(keyStr);
2  217 --          IF anEntry <> NIL THEN
2  218 1-              BEGIN
2  219 --              CatchFailures(fi, HdlSetString);
2  220 --              SetString(anEntry.fValue, valueStr);
2  221 --              FailMemError;                           { SetString may increase the handle size    }
2  222 --              Success(fi);
2  223 -1              END
2  224 --          ELSE
2  225 1-              BEGIN
2  226 --              New(anEntry);
2  227 --              FailNIL(anEntry);
2  228 --              anEntry.IEntry(keyStr, valueStr);
2  229 --              fEntries.Insert(anEntry);
2  230 -1              END;
2  231 -0 A   END {TAssociation.InsertEntry};
2  232 --
2  233 --      {-----------------------------------+
2  234 --      |   KeyAt                           |
2  235 --      +-----------------------------------}
2  236 -- A   FUNCTION  TAssociation.KeyAt (valueStr: Str255; VAR keyStr: Str255): BOOLEAN;
2  237 --
2  238 --      VAR
2  239 --          theEntry:       TEntry;
2  240 --
2  241 --          {-------------------------------+
2  242 --          |   MatchesValue                |
2  243 --          +-------------------------------}
2  244 -- B       FUNCTION  MatchesValue (theEntry: TEntry): BOOLEAN;
2  245 --
2  246 0- B       BEGIN
2  247 --              MatchesValue := valueStr = theEntry.fValue^^;
2  248 -0 B       END;
2  249 --
2  250 0- A   BEGIN
2  251 --          theEntry := FirstEntryThat(MatchesValue);
2  252 --          IF theEntry = NIL THEN
2  253 1-              BEGIN
2  254 --              keyStr := '';
2  255 --              KeyAt := FALSE;
2  256 -1              END
2  257 --          ELSE
2  258 1-              BEGIN
2  259 --              keyStr := theEntry.fKey^^;
2  260 --              KeyAt := TRUE;
2  261 -1              END;
2  262 -0 A   END {TAssociation.KeyAt};
2  263 --
2  264 --      {-----------------------------------+
2  265 --      |   RemoveValueAt                   |
2  266 --      +-----------------------------------}
2  267 -- A   PROCEDURE TAssociation.RemoveValueAt (keyStr: Str255);
2  268 --
2  269 --      VAR
2  270 --          theEntry:       TEntry;
2  271 --
2  272 --          {-------------------------------+
2  273 --          |   TestKey                     |
2  274 --          +-------------------------------}
2  275 -- B       FUNCTION  TestKey (anItem: TObject): INTEGER;
2  276 --
2  277 0- B       BEGIN
2  278 --              IF keyStr < TEntry(anItem).fKey^^ THEN
2  279 --                  TestKey := -1
2  280 --              ELSE IF keyStr > TEntry(anItem).fKey^^ THEN
2  281 --                  Testkey := 1
2  282 --              ELSE
2  283 --                  TestKey := 0;
2  284 -0 B       END;
2  285 --
2  286 0- A   BEGIN
2  287 --          theEntry := TEntry(fEntries.Search(TestKey));
2  288 --          IF theEntry <> NIL THEN
2  289 --              fEntries.Delete(theEntry);
2  290 -0 A   END {TAssociation.RemoveValueAt};
2  291 --
2  292 --      {-----------------------------------+
2  293 --      |   RemoveKeyAt                     |
2  294 --      +-----------------------------------}
2  295 -- A   PROCEDURE TAssociation.RemoveKeyAt (valueStr: Str255);
2  296 --
```

```
2  297 --      VAR
2  298 --          theEntry:       TEntry;
2  299 --
2  300 --          {-------------------------------+
2  301 --          |   MatchesValue                |
2  302 --          +-------------------------------}
2  303 -- B       FUNCTION  MatchesValue (theEntry: TEntry): BOOLEAN;
2  304 --
2  305 0- B       BEGIN
2  306 --              MatchesValue := valueStr = theEntry.fValue^^;
2  307 -0 B       END;
2  308 --
2  309 0- A   BEGIN
2  310 --          theEntry := FirstEntryThat(MatchesValue);
2  311 --          IF theEntry <> NIL THEN
2  312 --              fEntries.Delete(theEntry);
2  313 -0 A   END {TAssociation.RemoveKeyAt};
2  314 --
2  315 --      {---------------------------------+
2  316 --      |   ValueAt                       |
2  317 --      +---------------------------------}
2  318 -- A   FUNCTION  TAssociation.ValueAt (keyStr: Str255; VAR valueStr: Str255): BOOLEAN;
2  319 --
2  320 --      VAR
2  321 --          theEntry:       TEntry;
2  322 --
2  323 --          {-------------------------------+
2  324 --          |   CompareKey                  |
2  325 --          +-------------------------------}
2  326 -- B       FUNCTION  CompareKey (theEntry: TObject): INTEGER;
2  327 --
2  328 0- B       BEGIN
2  329 --              IF keyStr < TEntry(theEntry).fKey^^ THEN
2  330 --                  CompareKey := -1
2  331 --              ELSE IF keyStr > TEntry(theEntry).fKey^^ THEN
2  332 --                  CompareKey := 1
2  333 --              ELSE
2  334 --                  CompareKey := 0;
2  335 -0 B       END;
2  336 --
2  337 0- A   BEGIN
2  338 --          theEntry := TEntry(fEntries.Search(CompareKey));
2  339 --          IF theEntry = NIL THEN
2  340 1-              BEGIN
2  341 --              valueStr := '';
2  342 --              ValueAt := FALSE;
2  343 -1              END
2  344 --          ELSE
2  345 1-              BEGIN
2  346 --              valueStr := theEntry.fValue^^;
2  347 --              ValueAt := TRUE;
2  348 -1              END;
2  349 -0 A   END {TAssociation.ValueAt};
2  350 --
2  351 --      {$IFC qDebug}
2  352 --          {-----------------------------------+
2  353 --          |   Fields                          |
2  354 --          +-----------------------------------}
2  355 -- A       PROCEDURE TAssociation.Fields (PROCEDURE DoToField (fieldName: Str255;
2  356 --                                                             fieldAddr: Ptr;
2  357 --                                                             fieldType: INTEGER)); OVERRIDE;
2  358 --
2  359 0- A       BEGIN
2  360 --              DoToField('TAssociation', NIL, bClass);
2  361 --              DoToField('fEntries', @fEntries, bObject);
2  362 --              INHERITED Fields(DoToField);
2  363 -0 A       END {TAssociation.Fields};
2  364 --      {$ENDC}
2  365 --
1   97 --
1   98 --      END.
```

```
3    1 --    {$P}
3    2 --    { UBusyCursor.p }
3    3 --    { Copyright © 1985 - 1988 Apple Computer, Inc.  All rights reserved. }
3    4 --
3    5 --    { Conditional compilation flags used by this unit:
3    6 --
3    7 --        qNeedsROM128K   True skips code for Mac XL, false leaves it in.
3    8 --
3    9 --        qTrace          True surrounds trap patches with $D+ and $D++ (because
3   10 --                        the default is $D++), false skips these directives.
3   11 --
3   12 --        The settings of $D, $R and $N are imported from the UMAUtil interface.
3   13 --    }
3   14 --
3   15 --    UNIT UBusyCursor;
3   16 --
3   17 --        {
3   18 --                        T H E O R Y   O F   O P E R A T I O N
3   19 --
3   20 --        The UBusyCursor unit implements a mechanism for automatically setting
3   21 --        the cursor to the watch when the application is "busy."
3   22 --
3   23 --        An application is considered "busy" if it hasn't called GetNextEvent
3   24 --        or EventAvail for a given number of ticks, where 60 ticks equals one
3   25 --        second.  The default time is defined by kWatchDelay, which is 120 ticks
3   26 --        or 2 seconds--it can be changed by calling BusyDelay.  After this period
3   27 --        of time has elapsed with no call to GetNextEvent or EventAvail, the
3   28 --        cursor is changed to the watch.  On the next call to GetNextEvent or
3   29 --        EventAvail, the cursor is restored to its state before it was changed to
3   30 --        the watch.
3   31 --
3   32 --        Changing the cursor to the watch is done by the VBL task AWatchTask.
3   33 --        AWatchTask's execution frequency is set to kWatchDelay, or by calling
3   34 --        BusyDelay.  When AWatchTask is executed it sets the cursor to the
3   35 --        watch and resets AWatchTask's vblCount.
3   36 --
3   37 --        The traps InitCursor, SetCursor and SetCCursor are patched so that
3   38 --        they "remember" the cursor being set before executing the trap.  This
3   39 --        will allow us to restore the cursor after it has been changed to the
3   40 --        busy watch.
3   41 --
3   42 --        The EventAvail and GetNextEvent traps are patched such that, before
3   43 --        executing the trap, AWatchTask's vblCount is reset, and if the busy
3   44 --        cursor is displayed, then the cursor is restored to the last cursor
3   45 --        set by SetCursor or SetCCursor.
3   46 --
3   47 --        Note that this unit supports the Macintosh XL, though the XL implement-
3   48 --        ation differs slightly from that described above.  See the source code.
3   49 --        }
3   50 --
3   51 --    INTERFACE
3   52 --
3   53 --    {-----------------------------------+
3   54 --    |   Uses                            |
3   55 --    +-----------------------------------}
3   56 --    USES
3   57 --        {$LOAD MacIntf.LOAD}
3   58 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
3   59 --        {$LOAD}
3   60 --            UMAUtil,
3   61 --            UPatch,
3   62 --            UFailure;
3   63 --
3   64 --    {-----------------------------------+
3   65 --    |   Constants                       |
3   66 --    +-----------------------------------}
3   67 --    CONST
3   68 --            kWatchDelay         =       2*60;   {default # of 1/60 sec. ticks before
3   69 --                                                    cursor changes to a watch}
3   70 --
3   71 --    {-----------------------------------+
3   72 --    |   Global Declarations             |
3   73 --    +-----------------------------------}
3   74 --    PROCEDURE BusyActivate(entering: BOOLEAN);
3   75 --        { Call BusyActivate if you want to activate or deactivate the busy cursor
3   76 --            mechanism.  This is call by MacApp when losing/gaining control to a
3   77 --            desk accessory or switcher partition. }
3   78 --
3   79 --    PROCEDURE BusyDelay(newDelay: INTEGER; forceBusy: BOOLEAN);
3   80 --        { Call BusyDelay if you want to change the busy cursor delay.
3   81 --            newDelay should be in 1/60 seconds; a value <= 0 means don't change the
3   82 --            delay.  If forceBusy is TRUE, then the watch is put up immediately,
3   83 --            otherwise it doesn't go up until the required time has passed.
3   84 --            (BusyDelay respects the state flags in the cursor info record (ie.,
3   85 --            changeToWatch and inControl.) }
3   86 --
3   87 --    PROCEDURE ResetBusyCursor;
3   88 --        { Call ResetBusyCursor if you want to change the cursor back to an arrow
3   89 --            and reset the time before changing back to a watch.  This is not
3   90 --            usually called directly by the application.  Instead, it
3   91 --            is called each time GetNextEvent and EventAvail is called, by
3   92 --            patches installed by BusyInstall. }
3   93 --
3   94 --    PROCEDURE BusyInstall;
3   95 --        { BusyInstall installs the busy cursor mechanism.  Typically this is
3   96 --            called once during program initialization.  It installs the
3   97 --            VBL task and patches the traps GetNextEvent, EventAvail, InitCursor,
3   98 --            SetCursor and SetCCursor. }
```

```
3   99 --
3  100 --      PROCEDURE BusyRemove;
3  101 --          { BusyRemove uninstalls the busy cursor mechanism.  Typically this is
3  102 --              called once during program termination.  It removes the VBL task
3  103 --              and unpatches the patched traps. }
3  104 --
3  105 --      IMPLEMENTATION
3  106 --
3  107 --      {$I UBusyCursor.incl.p}
```

```
4    1 --     {$P}
4    2 --     { UBusyCursor.incl.p }
4    3 --     { Copyright 1985-1988 Apple Computer, Inc.  All rights reserved. }
4    4 --
4    5 --     {-----------------------------------+
4    6 --     |   Local types                     |
4    7 --     +-----------------------------------}
4    8 --     TYPE
4    9 --         CursorInfo = RECORD
4   10 --             inColor:        BOOLEAN;        { Is the saved cursor in color? }
4   11 --             origCursor:     Cursor;         { Cursor at the time the busy cursor was
4   12 --                                                 put up, if not in color }
4   13 --             origCCursor:    CCrsrHandle;    { Cursor at the time the busy cursor was
4   14 --                                                 put up, if in color }
4   15 --             watchDelay:     INTEGER;        { time in 1/60 second before cursor
4   16 --                                                 changes to watch }
4   17 --             inControl:      BOOLEAN;        { managed by MacApp; TRUE iff MacApp is
4   18 --                                                 in control; if FALSE we don't
4   19 --                                                 change the cursor at all }
4   20 --             changeToWatch:  BOOLEAN;        { if TRUE, we automagically switch to
4   21 --                                                 the watch in the VBL task and switch
4   22 --                                                 to origCursor on a call to
4   23 --                                                 GetNextEvent or EventAvail;
4   24 --                                                 applications can changed this as
4   25 --                                                 necessary }
4   26 --             watchOn:        BOOLEAN;        { TRUE if the busy cursor on }
4   27 --             spare:          BOOLEAN;        { reserved for future use }
4   28 --             watchCursor:    Cursor;         { the watch cursor }
4   29 --             q:              QElem;          { vbl queue element for changing the
4   30 --                                                 cursor }
4   31 --             END;
4   32 --
4   33 --     {-----------------------------------+
4   34 --     |   Local variables                 |
4   35 --     +-----------------------------------}
4   36 --     VAR
4   37 --         gCursorInfo:        CursorInfo;
4   38 --         gEAPatch:           TrapPatch;      { patch for EventAvail }
4   39 --         gGNEPatch:          TrapPatch;      { patch for GetNextEvent }
4   40 --         gICPatch:           TrapPatch;      { patch for InitCursor }
4   41 --         gSCPatch:           TrapPatch;      { patch for SetCursor }
4   42 --         gSCCPatch:          TrapPatch;      { patch for SetCCursor }
4   43 --
4   44 --
4   45 --     {-----------------------------------+
4   46 --     |   External declarations           |
4   47 --     +-----------------------------------}
4   48 -- A   PROCEDURE AWatchTask; EXTERNAL; { busy cursor VBL task }
4   49 --
4   50 --     {$IFC NOT qNeedsROM128K}
4   51 -- A   PROCEDURE XLBusyDelay(millisec: LONGINT); EXTERNAL;
4   52 -- A   PROCEDURE XLBusyImage(c: Cursor); EXTERNAL;
4   53 -- A   PROCEDURE XLCursorInit; EXTERNAL;
4   54 --     {$ENDC}
4   55 --
4   56 --
4   57 --     {-----------------------------------+
4   58 --     |   Forward declarations            |
4   59 --     +-----------------------------------}
4   60 -- A   PROCEDURE BusyReset(delayTicks: INTEGER); FORWARD;
4   61 -- A   PROCEDURE SetMacAppCursor (VAR theCursor: Cursor); FORWARD;
4   62 -- A   PROCEDURE SetCMacAppCursor (theCCursor: CCrsrHandle); FORWARD;
4   63 -- A   PROCEDURE InitMacAppCursor; FORWARD;
4   64 --
4   65 --
4   66 --     {$S MAMain}
4   67 --     {-----------------------------------+
4   68 --     |   BusyActivate                    |
4   69 --     +-----------------------------------}
4   70 -- A   PROCEDURE BusyActivate(entering: BOOLEAN);
4   71 --
4   72 0- A   BEGIN
4   73 --         BusyReset(gCursorInfo.watchDelay);
4   74 --         gCursorInfo.inControl := entering;
4   75 --
4   76 --     {$IFC NOT qNeedsROM128K}
4   77 --         IF gConfiguration.machineType = envXL THEN
4   78 --             IF NOT entering THEN
4   79 --                 XLCursorInit;
4   80 --     {$ENDC}
4   81 -0 A   END { BusyActivate };
4   82 --
4   83 --
4   84 --     {$S MAMain}
4   85 --     {-----------------------------------+
4   86 --     |   BusyDelay                       |
4   87 --     +-----------------------------------}
4   88 -- A   PROCEDURE BusyDelay(newDelay: INTEGER; forceBusy: BOOLEAN);
4   89 --
4   90 0- A   BEGIN
4   91 --         WITH gCursorInfo DO
4   92 1-             BEGIN
4   93 --             IF newDelay > 0 THEN        { save new delay time }
4   94 --                 watchDelay := newDelay;
4   95 --
4   96 --             IF forceBusy THEN
4   97 --                 newDelay := 1;          { trigger on next tick }
4   98 --
```

```
4   99 --              IF newDelay > 0 THEN        { reset timer }
4  100 --                  BusyReset(newDelay);
4  101 -1              END;
4  102 -0 A    END { BusyDelay };
4  103 --
4  104 --
4  105 --      {$S MAInit}
4  106 --      {-----------------------------------+
4  107 --      |   BusyInstall                     |
4  108 --      +-----------------------------------}
4  109 -- A    PROCEDURE BusyInstall;
4  110 --
4  111 0- A    BEGIN
4  112 --          UseROMMap(TRUE);
4  113 --
4  114 --          { Setup the gCursorInfo record }
4  115 --
4  116 --          gCursorInfo.watchCursor := GetCursor(watchCursor)^^;
4  117 --          WITH gCursorInfo DO
4  118 1-              BEGIN
4  119 --              inControl := TRUE; { we are in control during initialization }
4  120 --              changeToWatch := TRUE;
4  121 --              watchOn := FALSE;
4  122 --              watchDelay := kWatchDelay;
4  123 --              inColor := FALSE;
4  124 --              origCursor := arrow;
4  125 --
4  126 --      {$IFC NOT qNeedsROM128K}
4  127 --          IF gConfiguration.machineType = envXL THEN
4  128 --                  XLBusyImage(gCursorInfo.watchCursor)
4  129 --              ELSE
4  130 --      {$ENDC}
4  131 2-                  BEGIN
4  132 --
4  133 --                  { Setup the VBL task }
4  134 --                  WITH q.vblQElem DO
4  135 3-                      BEGIN
4  136 --                      qType := ORD(vType);
4  137 --                      vblAddr := @AWatchTask;
4  138 --                      vblCount := kWatchDelay;
4  139 --                      vblPhase := 0;
4  140 -3                      END;
4  141 --
4  142 --                  { Patch the necessary traps for non-XL operation }
4  143 --                  FailOSErr(TaillPatch(gSCPatch, $A851, @SetMacappCursor));   { SetCursor }
4  144 --                  FailOSErr(TailPatch(gICPatch, $A850, @InitMacappCursor));   { InitCursor }
4  145 --                  IF gConfiguration.hasColorQD THEN
4  146 --                      FailOSErr(TaillPatch(gSCCPatch, $AA1C, @SetCMacappCursor)); { SetCCursor }
4  147 --
4  148 --                  { Install the VBL task }
4  149 --                  FailOSErr(VInstall(@q));
4  150 -2                  END;
4  151 -1              END;
4  152 --
4  153 --          { Patch the traps applicable to Mac and XL operation }
4  154 --          FailOSErr(TailPatch(gGNEPatch, $A970, @ResetBusyCursor));   { GetNextEvent }
4  155 --          FailOSErr(TailPatch(gEAPatch, $A971, @ResetBusyCursor));    { EventAvail }
4  156 --
4  157 --          { Turn on busy cursor right now; init watchDelay }
4  158 --          BusyDelay(kWatchDelay, TRUE);
4  159 -0 A    END { BusyInstall };
4  160 --
4  161 --
4  162 --      {$S MATerminate}
4  163 --      {-----------------------------------+
4  164 --      |   BusyRemove                      |
4  165 --      +-----------------------------------}
4  166 -- A    PROCEDURE BusyRemove;
4  167 --
4  168 --          { Undo changes for automatic busy cursor--removes the VBL task and
4  169 --              unpatches the patched traps supporting the busy cursor. }
4  170 --
4  171 --      VAR
4  172 --          e:  OSErr;
4  173 0- A    BEGIN
4  174 --      {$IFC NOT qNeedsROM128K}
4  175 --          IF gConfiguration.machineType = envXL THEN
4  176 1-              BEGIN
4  177 --              XLCursorInit;
4  178 -1              END
4  179 --          ELSE
4  180 --      {$ENDC}
4  181 1-              BEGIN
4  182 --              { Remove the VBL task }
4  183 --              e := VRemove(@gCursorInfo.q);
4  184 --
4  185 --              { Unpatch the non-XL patches }
4  186 --              UnpatchTrap(gICPatch);
4  187 --              UnpatchTrap(gSCPatch);
4  188 --              UnpatchTrap(gSCCPatch);
4  189 -1              END;
4  190 --
4  191 --          { These patches apply to all Macs }
4  192 --          UnpatchTrap(gGNEPatch);
4  193 --          UnpatchTrap(gEAPatch);
4  194 -0 A    END { BusyRemove };
4  195 --
4  196 --
4  197 --      {$S MAMain}
```

```
4  198 --      {-----------------------------------+
4  199 --      |   BusyReset                       |
4  200 --      +-----------------------------------}
4  201 -- A    PROCEDURE BusyReset(delayTicks: INTEGER);
4  202 --
4  203 0- A    BEGIN
4  204 --          WITH gCursorInfo DO
4  205 --              IF inControl THEN { ??? should we make this test ??? }
4  206 --                  IF changeToWatch THEN
4  207 --      {$IFC NOT qNeedsROM128K}
4  208 --                      IF gConfiguration.machineType = envXL THEN
4  209 --                          XLBusyDelay((Ord4(delayTicks) * 1000) DIV 60)
4  210 --                      ELSE
4  211 --      {$ENDC}
4  212 1-                           BEGIN
4  213 --                          IF watchOn THEN
4  214 --                              IF inColor THEN
4  215 --                                  SetCCursor(origCCursor)
4  216 --                              ELSE
4  217 --                                  SetCursor(origCursor);
4  218 --                          q.vblQElem.vblCount := delayTicks;
4  219 -1                           END;
4  220 -0 A    END { BusyReset };
4  221 --
4  222 --
4  223 --      {$S Main}
4  224 --      {$IFC qTrace}{$D+}{$ENDC}
4  225 --      {-----------------------------------+
4  226 --      |   BusyTurnOff                     |
4  227 --      +-----------------------------------}
4  228 -- A    PROCEDURE BusyTurnOff;
4  229 --
4  230 --          { This is called from InitMacAppCursor, SetMacAppCursor and SetCMacAppCursor.
4  231 --              It sets gCursorInfo fields to indicate that the cursor is not the
4  232 --              busy watch. }
4  233 --
4  234 0- A    BEGIN
4  235 --          WITH gCursorInfo DO
4  236 --              IF inControl THEN
4  237 --                  IF changeToWatch THEN
4  238 1-                       BEGIN
4  239 --                      watchOn := FALSE; { anyone that sets the busy cursor should set this TRUE explicitly }
4  240 --                      q.vblQElem.vblCount := gCursorInfo.watchDelay;
4  241 -1                       END;
4  242 -0 A    END { BusyTurnOff };
4  243 --      {$IFC qTrace}{$D++}{$ENDC}
4  244 --
4  245 --
4  246 --      {$S Main}
4  247 --      {$IFC qTrace}{$D+}{$ENDC}
4  248 --      {-----------------------------------+
4  249 --      |   InitMacAppCursor                |
4  250 --      +-----------------------------------}
4  251 -- A    PROCEDURE InitMacAppCursor;
4  252 --
4  253 --          { Called when the InitCursor trap is executed.  After completion, we jump
4  254 --              to the ROM InitCursor. }
4  255 --
4  256 0- A    BEGIN
4  257 --          SetMacAppCursor(arrow);
4  258 -0 A    END { InitMacAppCursor };
4  259 --      {$IFC qTrace}{$D++}{$ENDC}
4  260 --
4  261 --
4  262 --      {$IFC qTrace}{$D+}{$ENDC}
4  263 --      {$S MAMain}
4  264 --      {-----------------------------------+
4  265 --      |   ResetBusyCursor                 |
4  266 --      +-----------------------------------}
4  267 -- A    PROCEDURE ResetBusyCursor;
4  268 --
4  269 --          { Patches GetNextEvent and EventAvail to reset the busy cursor.
4  270 --              Also callable from a program. }
4  271 --
4  272 0- A    BEGIN
4  273 --          SetupA5;        { ***** Called from trap patches *****}
4  274 --          BusyReset(gCursorInfo.watchDelay);
4  275 --          RestoreA5;
4  276 -0 A    END { ResetBusyCursor };
4  277 --      {$IFC qTrace}{$D++}{$ENDC}
4  278 --
4  279 --
4  280 --      {$S Main}
4  281 --      {$IFC qTrace}{$D+}{$ENDC}
4  282 --      {-----------------------------------+
4  283 --      |   SetCMacAppCursor                |
4  284 --      +-----------------------------------}
4  285 -- A    PROCEDURE SetCMacAppCursor (theCCursor: CCrsrHandle);
4  286 --
4  287 --          { The SetCCursor patch, used to remember the color cursor being set.
4  288 --              Installed as a "tail" patch, meaning the original SetCCursor trap
4  289 --              is called after this code has completed.}
4  290 --
4  291 0- A    BEGIN
4  292 --          SetupA5;        { ***** Called from trap patches *****}
4  293 --          BusyTurnOff;
4  294 --          WITH gCursorInfo DO
4  295 1-               BEGIN
4  296 --              inColor := TRUE;
```

```
4  297 --              { Save a copy of the color cursor }
4  298 --              gCursorInfo.origCCursor := theCCursor;
4  299 -1             END;
4  300 --
4  301 --          RestoreA5;
4  302 -0 A    END { SetCMacAppCursor };
4  303 --      {$IFC qTrace}{$D++}{$ENDC}
4  304 --
4  305 --
4  306 --      {$S Main}    { must be in Main segment, and cannot call to any other segment
4  307 --                     because SetCursor is called from the AWatchTask VBL task }
4  308 --      {$IFC qTrace}{$D+}{$ENDC}
4  309 --      {-----------------------------------+
4  310 --      |   SetMacAppCursor                 |
4  311 --      +-----------------------------------}
4  312 -- A    PROCEDURE SetMacAppCursor (VAR theCursor: Cursor);
4  313 --
4  314 --          { Patches SetCursor to remember the cursor being set.  Installed as a
4  315 --              "tail" patch, meaning the original SetCursor trap is called after
4  316 --              this code has completed.  Also called from InitMacAppCursor. }
4  317 --
4  318 0- A    BEGIN
4  319 --          SetupA5;        { ***** Called from trap patches *****}
4  320 --          BusyTurnOff;
4  321 --          { If we are setting the cursor to the busy watch, then don't save it }
4  322 --          IF @theCursor <> @gCursorInfo.watchCursor THEN
4  323 1-              BEGIN
4  324 --              gCursorInfo.inColor := FALSE;
4  325 --              gCursorInfo.origCursor := theCursor;
4  326 -1              END
4  327 --          ELSE
4  328 --              gCursorInfo.watchOn := TRUE;    { because BusyTurnOff set it to FALSE }
4  329 --          RestoreA5;
4  330 -0 A    END { SetMacAppCursor };
4  331 --      {$IFC qTrace}{$D++}{$ENDC}
3  108 --
3  109 --      END {UBusyCursor}.
```

```
5     1 --    {$P}
5     2 --    {UDialog.p}
5     3 --    {Copyright © 1988 by Apple Computer Inc.  All rights reserved.}
5     4 --
5     5 --    UNIT UDialog;
5     6 --
5     7 --    INTERFACE
5     8 --
5     9 --    {-----------------------------------+
5    10 --    |   Uses                            |
5    11 --    +-----------------------------------}
5    12 --    USES
5    13 --        {$LOAD MacIntf.LOAD}
5    14 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
5    15 --        {$LOAD UMacApp.LOAD}
5    16 --            UMAUtil, UViewCoords, UFailure,
5    17 --            UMemory, UMenuSetup, UObject, UList, UAssociation, UMacApp,
5    18 --        {$LOAD}
5    19 --            UTEView,
5    20 --            Sane,
5    21 --            Script;
5    22 --
5    23 --    {-----------------------------------+
5    24 --    |   Constants                       |
5    25 --    +-----------------------------------}
5    26 --    CONST
5    27 --
5    28 --        { Miscellaneous Constants}
5    29 --
5    30 --        kPreferColor    =   TRUE;               { argument to IIcon                          }
5    31 --        kControlOn      =   TRUE;               { argument to ICheckBox and IRadio           }
5    32 --
5    33 --        kFrame          =   [adnLineTop, adnLineLeft, adnLineBottom, adnLineRight];
5    34 --
5    35 --        Chr00           =   CHR(0);             { Needed for ControlCharSet                  }
5    36 --        Chr1F           =   CHR($1F);
5    37 --
5    38 --        kMaxTEWidth     =   $2B0;               { Maximum width for TEditText autoscrolling  }
5    39 --
5    40 --    {-----------------------------------+
5    41 --    |   Types                           |
5    42 --    +-----------------------------------}
5    43 --    TYPE
5    44 --
5    45 --        ControlCharSet =    SET OF Chr00..Chr1F;
5    46 --
5    47 --        {-----------------------------------+
5    48 --        |   TDialogView                     |
5    49 --        +-----------------------------------}
5    50 --        DialogViewTemplatePtr = ^DialogViewTemplate;
5    51 --        DialogViewTemplate =    RECORD
5    52 --                                    defaultItem:    IDType;
5    53 --                                    cancelItem:     IDType;
5    54 --                                END;
5    55 --
5    56 --        TDialogView =   OBJECT(TView)
5    57 --            fDefaultItem:       IDType;         { The identifier of the default item         }
5    58 --            fCancelItem:        IDType;         { The identifier of item that cancels dialog }
5    59 --            fParamTxt:          TAssociation;   { the list of TEntrys for replacement        }
5    60 --            fCurrentEditText:   TEditText;      { Currently selected edit text item          }
5    61 --            fTEView:            TDialogTEView;  { Used for editing in EditText views         }
5    62 --            fDismissed:         BOOLEAN;        { Has the dialog been dismissed?             }
5    63 --            fDismisser:         IDType;         { ID of view that caused dismissal           }
5    64 --
5    65 --        {$IFC qProceduralViews}
5    66 --            PROCEDURE TDialogView.IDialogView (itsDocument: TDocument; itsSuperView: TView;
5    67 --                                               itsLocation, itsSize: VPoint;
5    68 --                                               itsHSizeDet, itsVSizeDet: SizeDeterminer;
5    69 --                                               itsDefItemID, itsCancelItemID: IDType);
5    70 --        {$ENDC}
5    71 --
5    72 --        {$IFC qTemplateViews}
5    73 --            PROCEDURE TDialogView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5    74 --        {$ENDC}
5    75 --
5    76 --        {$IFC qWriteTemplates}
5    77 --            PROCEDURE TDialogView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5    78 --
5    79 --            PROCEDURE TDialogView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5    80 --        {$ENDC}
5    81 --
5    82 --            PROCEDURE TDialogView.Free; OVERRIDE;
5    83 --
5    84 --            FUNCTION  TDialogView.CanDismiss (dismissing: IDType): BOOLEAN;
5    85 --
5    86 --            PROCEDURE TDialogView.CantDeselect (theEditText: TEditText);
5    87 --
5    88 --            PROCEDURE TDialogView.Close; OVERRIDE;
5    89 --
5    90 --            PROCEDURE TDialogView.DismissDialog (dismisser: IDType);
5    91 --
5    92 --            FUNCTION  TDialogView.DoCommandKey(ch: CHAR; VAR info: EventInfo): TCommand; OVERRIDE;
5    93 --                { Called when a keyDown event is received and the command key is down. }
5    94 --
5    95 --            FUNCTION  TDialogView.DoKeyCommand (ch: Char; aKeyCode: INTEGER;
5    96 --                                               VAR info: EventInfo): TCommand; OVERRIDE;
5    97 --
5    98 --            PROCEDURE TDialogView.DoSelectEditText (theEditText: TEditText; selectChars: BOOLEAN);
```

```
5   99 --
5  100 --              PROCEDURE TDialogView.EachEditText (PROCEDURE DoToEditText (theEditText: TEditText));
5  101 --
5  102 --              FUNCTION  TDialogView.GetDialogView: TView; OVERRIDE;
5  103 --
5  104 --              FUNCTION  TDialogView.HandleCR (callingView: TView; VAR info: EventInfo): TCommand;
5  105 --
5  106 --              FUNCTION  TDialogView.HandleTab (callingView: TView; VAR info: EventInfo): TCommand;
5  107 --
5  108 --              PROCEDURE TDialogView.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
5  109 --
5  110 --              FUNCTION  TDialogView.MakeTEView: TDialogTEView;
5  111 --
5  112 --              PROCEDURE TDialogView.ParamTxt (keyStr, valueStr: Str255);
5  113 --
5  114 --              FUNCTION  TDialogView.PoseModally: IDType;
5  115 --
5  116 --              PROCEDURE TDialogView.ReplaceText (theText: Handle);
5  117 --
5  118 --              PROCEDURE TDialogView.SelectEditText (itsIdentifier: IDType; selectChars: BOOLEAN);
5  119 --
5  120 --              PROCEDURE TDialogView.SurveyEditText (VAR first, last, next, previous: TEditText);
5  121 --
5  122 --          {$IFC qDebug}
5  123 --              PROCEDURE TDialogView.Fields (PROCEDURE DoToField (fieldName: Str255;
5  124 --                                                 fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  125 --          {$ENDC}
5  126 --
5  127 --              END {TDialogView};
5  128 --
5  129 --
5  130 --          {-----------------------------------+
5  131 --          |   TButton                         |
5  132 --          +-----------------------------------}
5  133 --          ButtonTemplatePtr = ^ButtonTemplate;
5  134 --          ButtonTemplate =    RECORD
5  135 --                                  itsLabel:   Str255;     { Actually, variable length                 }
5  136 --                              END;
5  137 --
5  138 --          TButton =       OBJECT (TCtlMgr)
5  139 --
5  140 --          {$IFC qProceduralViews}
5  141 --              PROCEDURE TButton.IButton (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  142 --                                         itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  143 --                                         itsLabel: Str255);
5  144 --          {$ENDC}
5  145 --
5  146 --          {$IFC qTemplateViews}
5  147 --              PROCEDURE TButton.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  148 --          {$ENDC}
5  149 --
5  150 --          {$IFC qWriteTemplates}
5  151 --              PROCEDURE TButton.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  152 --
5  153 --              PROCEDURE TButton.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  154 --          {$ENDC}
5  155 --
5  156 --              END {TButton};
5  157 --
5  158 --
5  159 --          {-----------------------------------+
5  160 --          |   TCheckBox                       |
5  161 --          +-----------------------------------}
5  162 --          CheckBoxTemplatePtr = ^CheckBoxTemplate;
5  163 --          CheckBoxTemplate =  RECORD
5  164 --                                  isOn:       BOOLEAN;
5  165 --                                  itsLabel:   Str255;     { Actually, variable length                 }
5  166 --                              END;
5  167 --
5  168 --          TCheckBox =     OBJECT (TCtlMgr)
5  169 --
5  170 --          {$IFC qProceduralViews}
5  171 --              PROCEDURE TCheckBox.ICheckBox (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  172 --                                             itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  173 --                                             itsLabel: Str255; isTurnedOn: BOOLEAN);
5  174 --          {$ENDC}
5  175 --
5  176 --          {$IFC qTemplateViews}
5  177 --              PROCEDURE TCheckBox.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  178 --          {$ENDC}
5  179 --
5  180 --          {$IFC qWriteTemplates}
5  181 --              PROCEDURE TCheckBox.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  182 --
5  183 --              PROCEDURE TCheckBox.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  184 --          {$ENDC}
5  185 --
5  186 --              PROCEDURE TCheckBox.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
5  187 --
5  188 --              FUNCTION  TCheckBox.IsOn: BOOLEAN;
5  189 --
5  190 --              PROCEDURE TCheckBox.SetState (state, redraw: BOOLEAN);
5  191 --
5  192 --              PROCEDURE TCheckBox.Toggle (redraw: BOOLEAN);
5  193 --
5  194 --              PROCEDURE TCheckBox.ToggleIf (matchState, redraw: BOOLEAN);
5  195 --
5  196 --              END {TCheckBox};
5  197 --
```

```
5  198 --
5  199 --          {-----------------------------------+
5  200 --          |   TRadio                          |
5  201 --          +-----------------------------------}
5  202 --          RadioTemplatePtr = ^RadioTemplate;
5  203 --          RadioTemplate =     RECORD
5  204 --                                  isOn:       BOOLEAN;
5  205 --                                  itsLabel:   Str255;         { Actually, variable length                 }
5  206 --                              END;
5  207 --
5  208 --          TRadio =        OBJECT (TCtlMgr)
5  209 --
5  210 --          {$IFC qProceduralViews}
5  211 --              PROCEDURE TRadio.IRadio (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  212 --                                       itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  213 --                                       itsLabel: Str255; isTurnedOn: BOOLEAN);
5  214 --          {$ENDC}
5  215 --
5  216 --          {$IFC qTemplateViews}
5  217 --              PROCEDURE TRadio.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  218 --          {$ENDC}
5  219 --
5  220 --          {$IFC qWriteTemplates}
5  221 --              PROCEDURE TRadio.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  222 --
5  223 --              PROCEDURE TRadio.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  224 --          {$ENDC}
5  225 --
5  226 --              PROCEDURE TRadio.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
5  227 --
5  228 --              FUNCTION  TRadio.IsOn: BOOLEAN;
5  229 --
5  230 --              PROCEDURE TRadio.SetState (state, redraw: BOOLEAN);
5  231 --
5  232 --              PROCEDURE TRadio.Toggle (redraw: BOOLEAN);
5  233 --
5  234 --              PROCEDURE TRadio.ToggleIf (matchState, redraw: BOOLEAN);
5  235 --
5  236 --              END {TRadio};
5  237 --
5  238 --
5  239 --          {-----------------------------------+
5  240 --          |   TCluster                        |
5  241 --          +-----------------------------------}
5  242 --          ClusterTemplatePtr = ^ClusterTemplate;
5  243 --          ClusterTemplate =   RECORD
5  244 --                                  itsLabel:   Str255;     { Actually, variable length                 }
5  245 --                              END;
5  246 --
5  247 --          TCluster =      OBJECT (TControl)
5  248 --
5  249 --              fRsrcID:        INTEGER;
5  250 --              fIndex:         INTEGER;
5  251 --              fDataHandle:    StringHandle;
5  252 --
5  253 --          {$IFC qProceduralViews}
5  254 --              PROCEDURE TCluster.ICluster (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  255 --                                       itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  256 --                                       itsRsrcID, itsIndex: INTEGER);
5  257 --          {$ENDC}
5  258 --
5  259 --          {$IFC qTemplateViews}
5  260 --              PROCEDURE TCluster.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  261 --          {$ENDC}
5  262 --
5  263 --          {$IFC qWriteTemplates}
5  264 --              PROCEDURE TCluster.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  265 --
5  266 --              PROCEDURE TCluster.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  267 --          {$ENDC}
5  268 --
5  269 --              PROCEDURE TCluster.Free; OVERRIDE;
5  270 --
5  271 --              PROCEDURE TCluster.Draw (area: Rect); OVERRIDE;
5  272 --
5  273 --              PROCEDURE TCluster.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
5  274 --
5  275 --              PROCEDURE TCluster.GetLabel (VAR theLabel: Str255);
5  276 --
5  277 --              PROCEDURE TCluster.ReleaseLabel;
5  278 --
5  279 --              FUNCTION  TCluster.ReportCurrent: IDType;
5  280 --
5  281 --              PROCEDURE TCluster.SetLabel(theLabel: Str255; redraw: BOOLEAN);
5  282 --
5  283 --          {$IFC qDebug}
5  284 --              PROCEDURE TCluster.Fields (PROCEDURE DoToField (fieldName: Str255;
5  285 --                                          fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  286 --          {$ENDC}
5  287 --              END {TCluster};
5  288 --
5  289 --
5  290 --          {-----------------------------------+
5  291 --          |   TIcon                           |
5  292 --          +-----------------------------------}
5  293 --          IconTemplatePtr = ^IconTemplate;
5  294 --          IconTemplate =  RECORD
5  295 --                              preferColor:    BOOLEAN;
5  296 --                              rsrcID:         INTEGER;
```

```
5  297 --                          END;
5  298 --
5  299 --          TIcon =         OBJECT (TControl)
5  300 --
5  301 --              fPreferColor:   BOOLEAN;
5  302 --              fRsrcID:        INTEGER;
5  303 --              fDataHandle:    Handle;
5  304 --
5  305 --          {$IFC qProceduralViews}
5  306 --              PROCEDURE TIcon.IIcon (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  307 --                                     itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  308 --                                     itsRsrcID: INTEGER; preferColor: BOOLEAN);
5  309 --          {$ENDC}
5  310 --
5  311 --          {$IFC qTemplateViews}
5  312 --              PROCEDURE TIcon.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  313 --          {$ENDC}
5  314 --
5  315 --          {$IFC qWriteTemplates}
5  316 --              PROCEDURE TIcon.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  317 --
5  318 --              PROCEDURE TIcon.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  319 --          {$ENDC}
5  320 --
5  321 --              PROCEDURE TIcon.Free; OVERRIDE;
5  322 --
5  323 --              PROCEDURE TIcon.Draw (area: Rect); OVERRIDE;
5  324 --
5  325 --              PROCEDURE TIcon.ReleaseIcon;
5  326 --
5  327 --              PROCEDURE TIcon.SetIcon (theIcon: Handle; redraw: BOOLEAN);
5  328 --
5  329 --          {$IFC qDebug}
5  330 --              PROCEDURE TIcon.Fields (PROCEDURE DoToField (fieldName: Str255;
5  331 --                                                           fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  332 --          {$ENDC}
5  333 --              END {TIcon};
5  334 --
5  335 --
5  336 --          {-----------------------------------+
5  337 --          |   TPattern                        |
5  338 --          +-----------------------------------}
5  339 --          PatternTemplatePtr = ^PatternTemplate;
5  340 --          PatternTemplate =   RECORD
5  341 --                                  preferColor:    BOOLEAN;
5  342 --                                  rsrcID:         INTEGER;
5  343 --                              END;
5  344 --
5  345 --          TPattern =          OBJECT (TControl)
5  346 --
5  347 --              fPreferColor:   BOOLEAN;
5  348 --              fRsrcID:        INTEGER;
5  349 --              fDataHandle:    Handle;
5  350 --
5  351 --          {$IFC qProceduralViews}
5  352 --              PROCEDURE TPattern.IPattern (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  353 --                                           itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  354 --                                           itsRsrcID: INTEGER; preferColor: BOOLEAN);
5  355 --          {$ENDC}
5  356 --
5  357 --          {$IFC qTemplateViews}
5  358 --              PROCEDURE TPattern.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  359 --          {$ENDC}
5  360 --
5  361 --          {$IFC qWriteTemplates}
5  362 --              PROCEDURE TPattern.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  363 --
5  364 --              PROCEDURE TPattern.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  365 --          {$ENDC}
5  366 --
5  367 --              PROCEDURE TPattern.Free; OVERRIDE;
5  368 --
5  369 --              PROCEDURE TPattern.Draw (area: Rect); OVERRIDE;
5  370 --
5  371 --              PROCEDURE TPattern.ReleasePattern;
5  372 --
5  373 --              PROCEDURE TPattern.SetPattern (thePattern: Handle; redraw: BOOLEAN);
5  374 --
5  375 --          {$IFC qDebug}
5  376 --              PROCEDURE TPattern.Fields (PROCEDURE DoToField (fieldName: Str255;
5  377 --                                                              fieldAddr: Ptr;
5  378 --                                                              fieldType: INTEGER)); OVERRIDE;
5  379 --          {$ENDC}
5  380 --              END {TPattern};
5  381 --
5  382 --
5  383 --          {-----------------------------------+
5  384 --          |   TPicture                        |
5  385 --          +-----------------------------------}
5  386 --          PictureTemplatePtr = ^PictureTemplate;
5  387 --          PictureTemplate =   RECORD
5  388 --                                  rsrcID:     INTEGER;
5  389 --                              END;
5  390 --
5  391 --          TPicture =      OBJECT (TControl)
5  392 --
5  393 --              fRsrcID:        INTEGER;
5  394 --              fDataHandle:    PicHandle;
5  395 --
```

```
5  396 --          {$IFC qProceduralViews}
5  397 --              PROCEDURE TPicture.IPicture (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  398 --                                           itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  399 --                                           itsRsrcID: INTEGER);
5  400 --          {$ENDC}
5  401 --
5  402 --          {$IFC qTemplateViews}
5  403 --              PROCEDURE TPicture.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  404 --          {$ENDC}
5  405 --
5  406 --          {$IFC qWriteTemplates}
5  407 --              PROCEDURE TPicture.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  408 --
5  409 --              PROCEDURE TPicture.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  410 --          {$ENDC}
5  411 --
5  412 --              PROCEDURE TPicture.Free; OVERRIDE;
5  413 --
5  414 --              PROCEDURE TPicture.Draw (area: Rect); OVERRIDE;
5  415 --
5  416 --              PROCEDURE TPicture.ReleasePicture;
5  417 --
5  418 --              PROCEDURE TPicture.SetPicture (thePicture: PicHandle; redraw: BOOLEAN);
5  419 --
5  420 --          {$IFC qDebug}
5  421 --              PROCEDURE TPicture.Fields (PROCEDURE DoToField (fieldName: Str255;
5  422 --                                         fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  423 --          {$ENDC}
5  424 --              END {TPicture};
5  425 --
5  426 --
5  427 --          {-----------------------------------+
5  428 --          |   TPopup                          |
5  429 --          +-----------------------------------}
5  430 --          PopupTemplatePtr = ^PopupTemplate;
5  431 --          PopupTemplate = RECORD
5  432 --                              rsrcID:         INTEGER;
5  433 --                              currentItem:    INTEGER;
5  434 --                              itemOffset:     INTEGER;
5  435 --                          END;
5  436 --
5  437 --          TPopup =        OBJECT (TControl)
5  438 --
5  439 --              fRsrcID:        INTEGER;
5  440 --              fMenuID:        INTEGER;
5  441 --              fMenuHandle:    MenuHandle;
5  442 --              fCurrentItem:   INTEGER;
5  443 --              fItemOffset:    INTEGER;
5  444 --
5  445 --          {$IFC qProceduralViews}
5  446 --              PROCEDURE TPopup.IPopup (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  447 --                                       itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  448 --                                       itsRsrcID, itsCurrentItem, itsItemOffset: INTEGER);
5  449 --          {$ENDC}
5  450 --
5  451 --          {$IFC qTemplateViews}
5  452 --              PROCEDURE TPopup.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  453 --          {$ENDC}
5  454 --
5  455 --          {$IFC qWriteTemplates}
5  456 --              PROCEDURE TPopup.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  457 --
5  458 --              PROCEDURE TPopup.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  459 --          {$ENDC}
5  460 --
5  461 --              PROCEDURE TPopup.Free; OVERRIDE;
5  462 --
5  463 --              PROCEDURE TPopup.AdjustBotRight;
5  464 --
5  465 --              PROCEDURE TPopup.CalcLabelRect (VAR theRect: Rect);
5  466 --
5  467 --              PROCEDURE TPopup.CalcMenuRect (VAR theRect: Rect);
5  468 --
5  469 --              FUNCTION  TPopup.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
5  470 --                                               VAR hysteresis: Point): TCommand; OVERRIDE;
5  471 --
5  472 --              PROCEDURE TPopup.Draw (area: Rect); OVERRIDE;
5  473 --
5  474 --              PROCEDURE TPopup.DrawLabel (area: Rect);
5  475 --
5  476 --              PROCEDURE TPopup.DrawPopupBox (area: Rect);
5  477 --
5  478 --              FUNCTION  TPopup.GetCurrentItem: INTEGER;
5  479 --
5  480 --              PROCEDURE TPopup.GetItemText (item: INTEGER; VAR theText: Str255);
5  481 --
5  482 --              PROCEDURE TPopup.ReleasePopup;
5  483 --
5  484 --              PROCEDURE TPopup.SetCurrentItem (item: INTEGER; redraw: BOOLEAN);
5  485 --
5  486 --              PROCEDURE TPopup.SetPopup (theMenu: MenuHandle; theRsrcID, currentItem: INTEGER;
5  487 --                                         redraw: BOOLEAN);
5  488 --
5  489 --          {$IFC qDebug}
5  490 --              PROCEDURE TPopup.Fields (PROCEDURE DoToField (fieldName: Str255;
5  491 --                                       fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  492 --          {$ENDC}
5  493 --              END {TPopup};
5  494 --
```

```
5  495 --
5  496 --          {-----------------------------------+
5  497 --          |   TStaticText                     |
5  498 --          +-----------------------------------}
5  499 --          StaticTextTemplatePtr = ^StaticTextTemplate;
5  500 --          StaticTextTemplate =    RECORD
5  501 --                                      just:       INTEGER;
5  502 --                                      data:       Str255;     { Actually, variable length                    }
5  503 --                                  END;
5  504 --
5  505 --          TStaticText =   OBJECT (TControl)
5  506 --
5  507 --              fRsrcID:        INTEGER;
5  508 --              fIndex:         INTEGER;
5  509 --              fJust:          INTEGER;
5  510 --              fDataHandle:    Handle;
5  511 --
5  512 --          {$IFC qProceduralViews}
5  513 --              PROCEDURE TStaticText.IStaticText (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  514 --                                                 itsHSizeDet, itsVSizeDet: SizeDeterminer;
5  515 --                                                 itsRsrcID, itsIndex: INTEGER);
5  516 --          {$ENDC}
5  517 --
5  518 --          {$IFC qTemplateViews}
5  519 --              PROCEDURE TStaticText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  520 --          {$ENDC}
5  521 --
5  522 --          {$IFC qWriteTemplates}
5  523 --              PROCEDURE TStaticText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  524 --
5  525 --              PROCEDURE TStaticText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  526 --          {$ENDC}
5  527 --
5  528 --              PROCEDURE TStaticText.Free; OVERRIDE;
5  529 --
5  530 --              FUNCTION  TStaticText.DoSubstitution (VAR realText: Handle): BOOLEAN;
5  531 --
5  532 --              PROCEDURE TStaticText.Draw (area: Rect); OVERRIDE;
5  533 --
5  534 --              PROCEDURE TStaticText.GetText (VAR theText: Str255);
5  535 --
5  536 --              PROCEDURE TStaticText.ImageText (text: Ptr; length: LONGINT; box: Rect;
5  537 --                                               just: INTEGER);
5  538 --
5  539 --              PROCEDURE TStaticText.ReleaseText;
5  540 --
5  541 --              PROCEDURE TStaticText.SetJustification (theJust: INTEGER; redraw: BOOLEAN);
5  542 --
5  543 --              PROCEDURE TStaticText.SetText (theText: Str255; redraw: BOOLEAN);
5  544 --
5  545 --          {$IFC qDebug}
5  546 --              PROCEDURE TStaticText.Fields (PROCEDURE DoToField (fieldName: Str255;
5  547 --                                            fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  548 --          {$ENDC}
5  549 --              END {TStaticText};
5  550 --
5  551 --
5  552 --          {-----------------------------------+
5  553 --          |   TEditText                       |
5  554 --          +-----------------------------------}
5  555 --          EditTextTemplatePtr = ^EditTextTemplate;
5  556 --          EditTextTemplate =  RECORD
5  557 --                                  maxChars:       INTEGER;
5  558 --                                  controlChars:   ControlCharSet;
5  559 --                              END;
5  560 --
5  561 --          TEditText =     OBJECT (TStaticText)
5  562 --
5  563 --              fControlChars:  ControlCharSet;
5  564 --              fMaxChars:      INTEGER;
5  565 --              fTEView:        TTEView;
5  566 --
5  567 --          {$IFC qProceduralViews}
5  568 --              PROCEDURE TEditText.IEditText (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  569 --                                             itsMaxChars: INTEGER);
5  570 --          {$ENDC}
5  571 --
5  572 --          {$IFC qTemplateViews}
5  573 --              PROCEDURE TEditText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  574 --          {$ENDC}
5  575 --
5  576 --          {$IFC qWriteTemplates}
5  577 --              PROCEDURE TEditText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  578 --
5  579 --              PROCEDURE TEditText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  580 --          {$ENDC}
5  581 --
5  582 --              PROCEDURE TEditText.DoIdle (phase: IdlePhase); OVERRIDE;
5  583 --
5  584 --              FUNCTION  TEditText.DoKeyCommand (ch: CHAR; aKeyCode: INTEGER; VAR info: EventInfo): TCommand; OVERRIDE;
5  585 --
5  586 --              FUNCTION  TEditText.DoMenuCommand (aCmdNumber: INTEGER): TCommand; OVERRIDE;
5  587 --
5  588 --              FUNCTION  TEditText.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
5  589 --                                                  VAR hysteresis: Point): TCommand; OVERRIDE;
5  590 --
5  591 --              PROCEDURE TEditText.DoSetupMenus; OVERRIDE;
5  592 --
5  593 --              FUNCTION  TEditText.DoSubstitution (VAR realText: Handle): BOOLEAN; OVERRIDE;
```

```
5  594 --
5  595 --              PROCEDURE TEditText.Draw (area: Rect); OVERRIDE;
5  596 --
5  597 --              PROCEDURE TEditText.GetText (VAR theText: Str255); OVERRIDE;
5  598 --
5  599 --              PROCEDURE TEditText.ImageText (text: Ptr; length: LONGINT; box: Rect;
5  600 --                                             just: INTEGER); OVERRIDE;
5  601 --
5  602 --              PROCEDURE TEditText.InstallSelection (wasActive, beActive: BOOLEAN); OVERRIDE;
5  603 --
5  604 --              PROCEDURE TEditText.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
5  605 --
5  606 --              PROCEDURE TEditText.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
5  607 --
5  608 --              PROCEDURE TEditText.RestartEdit;
5  609 --
5  610 --              PROCEDURE TEditText.SetJustification (theJust: INTEGER; redraw: BOOLEAN); OVERRIDE;
5  611 --
5  612 --              PROCEDURE TEditText.SetSelection (selStart, selEnd: INTEGER; redraw: BOOLEAN);
5  613 --
5  614 --              PROCEDURE TEditText.SetText (theText: Str255; redraw: BOOLEAN); OVERRIDE;
5  615 --
5  616 --              PROCEDURE TEditText.StartEdit (selectChars: BOOLEAN; theTEView: TDialogTEView);
5  617 --
5  618 --              FUNCTION  TEditText.StopEdit: BOOLEAN;
5  619 --
5  620 --              FUNCTION  TEditText.Validate: BOOLEAN; OVERRIDE;
5  621 --
5  622 --          {$IFC qDebug}
5  623 --              PROCEDURE TEditText.Fields (PROCEDURE DoToField (fieldName: Str255;
5  624 --                                          fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  625 --          {$ENDC}
5  626 --              END {TEditText};
5  627 --
5  628 --          {-----------------------------------+
5  629 --          |   TNumberText                     |
5  630 --          +-----------------------------------}
5  631 --          NumberTextTemplatePtr = ^NumberTextTemplate;
5  632 --          NumberTextTemplate =    RECORD
5  633 --                                      value:          LONGINT;
5  634 --                                      minimum:        LONGINT;
5  635 --                                      maximum:        LONGINT;
5  636 --                                  END;
5  637 --
5  638 --          TNumberText =   OBJECT (TEditText)
5  639 --
5  640 --              fMinimum:       LONGINT;
5  641 --              fMaximum:       LONGINT;
5  642 --
5  643 --          {$IFC qProceduralViews}
5  644 --              PROCEDURE TNumberText.INumberText (itsSuperView: TView; itsLocation, itsSize: VPoint;
5  645 --                                                 itsValue, itsMinimum, itsMaximum: LONGINT);
5  646 --          {$ENDC}
5  647 --
5  648 --          {$IFC qTemplateViews}
5  649 --              PROCEDURE TNumberText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
5  650 --          {$ENDC}
5  651 --
5  652 --          {$IFC qWriteTemplates}
5  653 --              PROCEDURE TNumberText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  654 --
5  655 --              PROCEDURE TNumberText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
5  656 --          {$ENDC}
5  657 --
5  658 --              FUNCTION  TNumberText.GetValue: LONGINT;
5  659 --
5  660 --              PROCEDURE TNumberText.SetValue (newValue: LONGINT; redraw: BOOLEAN);
5  661 --
5  662 --              FUNCTION  TNumberText.Validate: BOOLEAN; OVERRIDE;
5  663 --
5  664 --          {$IFC qDebug}
5  665 --              PROCEDURE TNumberText.Fields (PROCEDURE DoToField (fieldName: Str255;
5  666 --                                           fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  667 --          {$ENDC}
5  668 --              END {TNumberText};
5  669 --
5  670 --          {-----------------------------------+
5  671 --          |   TDialogTEView                   |
5  672 --          +-----------------------------------}
5  673 --          TDialogTEView =     OBJECT (TTEView)
5  674 --
5  675 --              fEditText:      TEditText;
5  676 --
5  677 --              PROCEDURE TDialogTEView.DrawContents; OVERRIDE;
5  678 --
5  679 --              PROCEDURE TDialogTEView.InstallEditText (theEditText: TEditText; selectChars: BOOLEAN);
5  680 --
5  681 --              PROCEDURE TDialogTEView.InstallSelection(wasActive, beActive: BOOLEAN); OVERRIDE;
5  682 --
5  683 --              PROCEDURE TDialogTEView.RecalcText; OVERRIDE;
5  684 --
5  685 --              PROCEDURE TDialogTEView.ScrollSelectionIntoView; OVERRIDE;
5  686 --
5  687 --          {$IFC qDebug}
5  688 --              PROCEDURE TDialogTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
5  689 --                                          fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
5  690 --          {$ENDC}
5  691 --              END {TDialogTEView};
5  692 --
```

```
5  693 --      {-------------------------------------+
5  694 --      |   Global routines                   |
5  695 --      +-------------------------------------}
5  696 --          PROCEDURE InitUDialog;
5  697 --
5  698 --      IMPLEMENTATION
5  699 --
5  700 --      {$I UDialog.inc1.p}
```

```
6    1 --     {$P}
6    2 --     { UDDialog.p }
6    3 --     { Copyright © 1988 Apple Computer Inc.  All rights reserved. }
6    4 --
6    5 --     (*****************************************************************************************)
6    6 --     (*     G l o b a l   R o u t i n e s                                                     *)
6    7 --     (*****************************************************************************************)
6    8 --     {$D DlgInit}
6    9 --     {-----------------------------------+
6   10 --     |   InitUDialog                     |
6   11 --     +-----------------------------------}
6   12 -- A   PROCEDURE InitUDialog;
6   13 --     VAR
6   14 --         aDialogView:    TDialogView;
6   15 --         aControl:       TControl;
6   16 --         aCtlMgr:        TCtlMgr;
6   17 --         aButton:        TButton;
6   18 --         aCheckBox:      TCheckBox;
6   19 --         aRadio:         TRadio;
6   20 --         aCluster:       TCluster;
6   21 --         aIcon:          TIcon;
6   22 --         aPicture:       TPicture;
6   23 --         aPopup:         TPopup;
6   24 --         aStaticText:    TStaticText;
6   25 --         anEditText:     TEditText;
6   26 --         aNumberText:    TNumberText;
6   27 --         aPattern:       TPattern;
6   28 --
6   29 0- A   BEGIN
6   30 --     {$IFC qTemplateViews}
6   31 --         NEW(aDialogView);   FailNIL(aDialogView);   RegisterStdType('TDialogView', 'dlog', aDialogView);
6   32 --         NEW(aControl);      FailNIL(aControl);      RegisterStdType('TControl', 'cntl', aControl);
6   33 --         NEW(aCtlMgr);       FailNIL(aCtlMgr);       RegisterType('TCtlMgr', aCtlMgr);
6   34 --         NEW(aButton);       FailNIL(aButton);       RegisterStdType('TButton', 'butn', aButton);
6   35 --         NEW(aCheckBox);     FailNIL(aCheckBox);     RegisterStdType('TCheckBox', 'chkb', aCheckBox);
6   36 --         NEW(aRadio);        FailNIL(aRadio);        RegisterStdType('TRadio', 'radb', aRadio);
6   37 --         NEW(aCluster);      FailNIL(aCluster);      RegisterStdType('TCluster', 'clus', aCluster);
6   38 --         NEW(aIcon);         FailNIL(aIcon);         RegisterStdType('TIcon', 'icon', aIcon);
6   39 --         NEW(aPicture);      FailNIL(aPicture);      RegisterStdType('TPicture', 'pict', aPicture);
6   40 --         NEW(aPopup);        FailNIL(aPopup);        RegisterStdType('TPopup', 'popp', aPopup);
6   41 --         NEW(aStaticText);   FailNIL(aStaticText);   RegisterStdType('TStaticText', 'stat', aStaticText);
6   42 --         NEW(anEditText);    FailNIL(anEditText);    RegisterStdType('TEditText', 'edit', anEditText);
6   43 --         NEW(aNumberText);   FailNIL(aNumberText);   RegisterStdType('TNumberText', 'nmbr', aNumberText);
6   44 --         NEW(aPattern);      FailNIL(aPattern);      RegisterStdType('TPattern', 'patn', aPattern);
6   45 --     {$ENDC}
6   46 -0 A   END;
6   47 --
6   48 --
6   49 --     {$S DlgRes}
6   50 --     {-----------------------------------+
6   51 --     |   FlashControl                    |
6   52 --     +-----------------------------------}
6   53 -- A   PROCEDURE FlashControl (theControl: TControl);
6   54 --     VAR
6   55 --         dontCare:       LONGINT;
6   56 0- A   BEGIN
6   57 --         IF theControl <> NIL THEN
6   58 1-             BEGIN
6   59 --             theControl.HiliteState(TRUE, kRedraw);
6   60 --             Delay(8, dontCare);
6   61 --             theControl.HiliteState(FALSE, kRedraw);
6   62 -1             END;
6   63 -0 A   END;
6   64 --
6   65 --     {$S DlgRes}
6   66 --     {-----------------------------------+
6   67 --     |   GetIfBkColor                    |
6   68 --     +-----------------------------------}
6   69 -- A   PROCEDURE GetIfBkColor (VAR aColor: RGBColor);
6   70 --     CONST
6   71 --         BlackBit    = 5;
6   72 --         YellowBit   = 6;
6   73 --         MagentaBit  = 7;
6   74 --         CyanBit     = 8;
6   75 --     VAR
6   76 --         oldColor:   LONGINT;
6   77 0- A   BEGIN
6   78 --         IF qConfiguration.hasColorQD THEN
6   79 --             GetBackColor(aColor)
6   80 --         ELSE
6   81 1-             BEGIN                                  { Map old, dumb CMYB system to RGB color    }
6   82 --             (*                          xxxxxxx C.MY B rgb w b  = RGB
6   83 --             blackColor      =  33 = 0000000 0.00 1 000 0 1  = 000
6   84 --             whiteColor      =  30 = 0000000 0.00 0 111 1 0  = 111
6   85 --             redColor        = 205 = 0000000 0.11 0 011 0 1  = 100
6   86 --             greenColor      = 341 = 0000000 1.01 0 101 0 1  = 010
6   87 --             blueColor       = 409 = 0000000 1.10 0 110 0 1  = 001
6   88 --             cyanColor       = 273 = 0000000 1.00 0 100 0 1  = 011
6   89 --             magentaColor    = 137 = 0000000 0.10 0 010 0 1  = 101
6   90 --             yellowColor     =  69 = 0000000 0.01 0 001 0 1  = 110
6   91 --             *)
6   92 --             oldColor := thePort^.bkColor;          { Fetch old color                           }
6   93 --             aColor := gRGBBlack;                   { Prime returned color to black             }
6   94 --             IF BTST(oldColor, BlackBit) THEN       { If color isn't black, force CMY = 111      }
6   95 --                 oldColor := BOR(oldColor, $1C0);
6   96 --             IF NOT BTST(oldColor, CyanBit) THEN    { Absence of cyan = presence of red         }
6   97 --                 aColor.red := $FFFF;
6   98 --             IF NOT BTST(oldColor, MagentaBit) THEN { Absence of magenta = presence of green    }
```

```
6   99 --                aColor.green := $FFFF;
6  100 --             IF NOT BTST(oldColor, YellowBit) THEN   { Absence of yellow = presence of blue      }
6  101 --                aColor.blue := $FFFF;
6  102 -1             END;
6  103 -0 A   END;
6  104 --
6  105 --     {$S DlgRes}
6  106 --     {-----------------------------------+
6  107 --     |   SetIfBkColor                    |
6  108 --     +-----------------------------------}
6  109 -- A   PROCEDURE SetIfBkColor (aColor: RGBColor);
6  110 --     CONST
6  111 --         SignBit     = 15;
6  112 --     VAR
6  113 --         index:      INTEGER;
6  114 --         oldColor:   LONGINT;
6  115 0- A   BEGIN
6  116 --         IF gConfiguration.hasColorQD THEN
6  117 --             RGBBackColor(aColor)
6  118 --         ELSE
6  119 1-             BEGIN
6  120 --             index := 0;                             { Prime index                             }
6  121 --             IF BTST(aColor.red, SignBit) THEN       { Set bit if red >= $8000                 }
6  122 --                 index := 4;
6  123 --             IF BTST(aColor.green, SignBit) THEN     { Set bit if green >= $8000               }
6  124 --                 index := index + 2;
6  125 --             IF BTST(aColor.blue, SignBit) THEN      { Set bit if blue >= $8000                }
6  126 --                 index := index + 1;
6  127 2-             CASE index OF
6  128 --                 0:  oldColor := blackColor;
6  129 --                 1:  oldColor := blueColor;
6  130 --                 2:  oldColor := greenColor;
6  131 --                 3:  oldColor := cyanColor;
6  132 --                 4:  oldColor := redColor;
6  133 --                 5:  oldColor := magentaColor;
6  134 --                 6:  oldColor := yellowColor;
6  135 --                 7:  oldColor := whiteColor;
6  136 -2                 END;
6  137 --             BackColor(oldColor);
6  138 -1             END;
6  139 -0 A   END;
6  140 --
6  141 --     {$S DlgRes}
6  142 --     {-----------------------------------+   Determines fore and background colors, based on
6  143 --     |   GetMenuColors                   |       screen depth and which device it will come up on
6  144 --     +-----------------------------------}
6  145 -- A   PROCEDURE GetMenuColors (popupRect: Rect; menuID, itemNum: INTEGER;
6  146 --                                          VAR fColor, bColor: RGBColor);
6  147 --     VAR
6  148 --         gotTitle:       BOOLEAN;
6  149 --         gdh:            GDHandle;
6  150 --         mce:            MCEntryPtr;
6  151 --         titleMce:       MCEntry;
6  152 --         globalMce:      MCEntry;
6  153 --
6  154 --         {-------------------------------+
6  155 --         |   SetBadColors                |
6  156 --         +-------------------------------}
6  157 -- B       PROCEDURE SetBadColors;
6  158 --
6  159 0- B       BEGIN
6  160 --             fColor := gRGBBlack;
6  161 --             bColor := gRGBWhite;
6  162 -0 B       END;
6  163 --
6  164 0- A   BEGIN
6  165 --         gotTitle := FALSE;                      { Assume the worst.  We always do.            }
6  166 --
6  167 --         IF gConfiguration.hasColorQD THEN       { First, be sure we have color QD…            }
6  168 1-             BEGIN
6  169 --             LocalToGlobal(popupRect.topLeft);   { Globalize rect, in focused coordinates      }
6  170 --             LocalToGlobal(popupRect.botRight);
6  171 --             gdh := GetMaxDevice(popupRect);     { Get device characteristics for that rect    }
6  172 --
6  173 --             IF gdh^^.gdPMap^^.pixelSize > 1 THEN    { If we have more than two colors         }
6  174 2-                 BEGIN
6  175 --                 mce := GetMCEntry(menuID, 0);   { Always get title entry                      }
6  176 --                 IF mce <> NIL THEN
6  177 3-                     BEGIN
6  178 --                     gotTitle := TRUE;
6  179 --                     titleMce := mce^;           { Future calls could shift memory             }
6  180 -3                     END;
6  181 --
6  182 --                 IF NOT gotTitle THEN            { If we can't get the title entry, then…      }
6  183 3-                     BEGIN
6  184 --                     mce := GetMCEntry(0, 0);    { …we'll need the global entry, too           }
6  185 --                     IF mce <> NIL THEN
6  186 --                         globalMce := mce^
6  187 --                     ELSE
6  188 4-                         BEGIN
6  189 --                         SetBadColors;           { If no title, AND no global entry, punt  *   }
6  190 --                         EXIT(GetMenuColors);    { Even if item guy exists. No title, No washee }
6  191 -4                         END;
6  192 -3                     END;
6  193 --
6  194 --                 { Handle a title color request }
6  195 --                 IF itemNum = 0 THEN
6  196 3-                     BEGIN
6  197 --                     IF gotTitle THEN
```

```
6  198 4-                      BEGIN
6  199 --                      fColor := titleMce.mctRGB1;
6  200 --                      bColor := titleMce.mctRGB2;
6  201 -4                      END
6  202 --                  ELSE { IF gotGlobal <<< has to be, by this point }
6  203 4-                      BEGIN
6  204 --                      fColor := globalMce.mctRGB1;
6  205 --                      bColor := globalMce.mctRGB4;
6  206 -4                      END;
6  207 -3                  END
6  208 --              { Otherwise, it's for an item }
6  209 --              ELSE
6  210 3-                  BEGIN
6  211 --                  mce := GetMCEntry(menuID, itemNum);
6  212 --                  IF mce <> NIL THEN
6  213 --                      fColor := mce^.mctRGB2
6  214 --                  ELSE IF gotTitle THEN
6  215 --                      fColor := titleMce.mctRGB3
6  216 --                  ELSE
6  217 --                      fColor := globalMce.mctRGB3;
6  218 --
6  219 --                  IF gotTitle THEN
6  220 --                      bColor := titleMce.mctRGB4
6  221 --                  ELSE
6  222 --                      bColor := globalMce.mctRGB2;
6  223 -3                  END;
6  224 -2              END
6  225 --          ELSE
6  226 --              SetBadColors;                   { Only one bit depth.  Default to B&W          }
6  227 -1          END
6  228 --      ELSE
6  229 --          SetBadColors;                       { Not using Color QuickDraw.  B&W for sure     }
6  230 --  {$IFC qDebug}
6  231 --      IF gIntenseDebugging THEN
6  232 1-          BEGIN
6  233 --          IF itemNum = 0 THEN WRITE('Title ') ELSE WRITE('Item #', itemNum:0);
6  234 --          WRITELN(' foreground color- R:', fColor.red:0, ', G:', fColor.green:0,
6  235 --            ', B:', fColor.blue:0);
6  236 --          IF itemNum = 0 THEN WRITE('Title ') ELSE WRITE('Item #', itemNum:0);
6  237 --          WRITELN(' background color- R:', bColor.red:0, ', G:', bColor.green:0,
6  238 --            ', B:', bColor.blue:0);
6  239 -1          END;
6  240 --  {$ENDC}
6  241 -0 A END;
6  242 --
6  243 --
6  244 --  (*****************************************************************************************)
6  245 --  (*      T D i a l o g V i e w                                                            *)
6  246 --  (*****************************************************************************************)
6  247 --
6  248 --  {$IFC qProceduralViews}
6  249 --  {$S DlgOpen}
6  250 --  {-----------------------------------+
6  251 --  |   IDialogView                     |
6  252 --  +-----------------------------------}
6  253 -- A PROCEDURE TDialogView.IDialogView (itsDocument: TDocument; itsSuperView: TView;
6  254 --                                      itsLocation, itsSize: VPoint;
6  255 --                                      itsHSizeDet, itsVSizeDet: SizeDeterminer;
6  256 --                                      itsDefItemID, itsCancelItemID: IDType);
6  257 --  VAR
6  258 --      anAssociation:      TAssociation;
6  259 --      aDialogTEView:      TDialogTEView;
6  260 --      fi:                 FailInfo;
6  261 --
6  262 --      {-------------------------------+
6  263 --      |   HandleFailure               |
6  264 --      +-------------------------------}
6  265 -- B    PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6  266 0- B    BEGIN
6  267 --          Free;
6  268 -0 B    END;
6  269 --
6  270 0- A BEGIN
6  271 --      fParamTxt := NIL;                       { In case of a catastrophe                     }
6  272 --      fTEView := NIL;                         { Ditto.                                       }
6  273 --      IView(itsDocument, itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6  274 --
6  275 --      CatchFailures(fi, HandleFailure);
6  276 --      NEW(anAssociation);                     { Okay to allocate list now                    }
6  277 --      FailNIL(anAssociation);
6  278 --      anAssociation.IAssociation;
6  279 --      fParamTxt := anAssociation;
6  280 --      fDefaultItem := itsDefItemID;
6  281 --      fCancelItem := itsCancelItemID;
6  282 --      fCurrentEditText := NIL;
6  283 --      fDismissed := FALSE;
6  284 --      fDismisser := kNoIdentifier;
6  285 --
6  286 --      aDialogTEView := MakeTEView;
6  287 --      fTEView := aDialogTEView;
6  288 --      Success(fi);
6  289 -0 A END;
6  290 --  {$ENDC}
6  291 --
6  292 --
6  293 --  {$IFC qTemplateViews}
6  294 --  {$S DlgOpen}
6  295 --  {-----------------------------------+
6  296 --  |   IRes                            |
```

```
6  297 --      +-----------------------------------}
6  298 -- A    PROCEDURE TDialogView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6  299 --
6  300 --      VAR
6  301 --          anAssociation:      TAssociation;
6  302 --          aDialogTEView:      TDialogTEView;
6  303 --          fi:                 FailInfo;
6  304 --
6  305 --          {-------------------------------+
6  306 --          |   HandleFailure               |
6  307 --          +-------------------------------}
6  308 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6  309 0- B       BEGIN
6  310 --              Free;
6  311 -0 B       END;
6  312 --
6  313 0- A    BEGIN
6  314 --          fParamTxt := NIL;                       { In case of a catastrophe                    }
6  315 --          fTEView := NIL;                         { Ditto.                                      }
6  316 --          INHERITED IRes(itsDocument, itsSuperView, itsParams);
6  317 --          WITH DialogViewTemplatePtr(itsParams)^ DO
6  318 1-              BEGIN
6  319 --              fDefaultItem := defaultItem;
6  320 --              fCancelItem := cancelItem;
6  321 -1              END;
6  322 --          CatchFailures(fi, HandleFailure);
6  323 --          NEW(anAssociation);                     { Okay to allocate list now                   }
6  324 --          FailNIL(anAssociation);
6  325 --          anAssociation.IAssociation;
6  326 --          fParamTxt := anAssociation;
6  327 --          fCurrentEditText := NIL;
6  328 --          fDismissed := FALSE;
6  329 --          fDismisser := kNoIdentifier;
6  330 --
6  331 --          aDialogTEView := MakeTEView;
6  332 --          fTEView := aDialogTEView;
6  333 --          Success(fi);
6  334 --
6  335 --          OffsetPtr(itsParams, SIZEOF(DialogViewTemplate));
6  336 -0 A    END;
6  337 --      {$ENDC}
6  338 --
6  339 --
6  340 --      {$IFC qWriteTemplates}
6  341 --      {$S MAWriteRes}
6  342 --      {-----------------------------------+
6  343 --      |   WRes                            |
6  344 --      +-----------------------------------}
6  345 -- A    PROCEDURE TDialogView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  346 --
6  347 --      VAR
6  348 --          dgPtr:      DialogViewTemplatePtr;
6  349 --
6  350 0- A    BEGIN
6  351 --          INHERITED WRes(theResource, itsParams);
6  352 --
6  353 --          dgPtr := DialogViewTemplatePtr(ExpandPtr(theResource, itsParams,
6  354 --            SIZEOF(DialogViewTemplate)));
6  355 --
6  356 --          WITH dgPtr^ DO
6  357 1-              BEGIN
6  358 --              defaultItem := fDefaultItem;
6  359 --              cancelItem := fCancelItem;
6  360 -1              END;
6  361 -0 A    END;
6  362 --
6  363 --      {$S MAWriteRes}
6  364 --      {-----------------------------------+
6  365 --      |   WriteRes                        |
6  366 --      +-----------------------------------}
6  367 -- A    PROCEDURE TDialogView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  368 0- A    BEGIN
6  369 --          gWResSignature := 'dlog';   gWResType := 'TDialogView';
6  370 --          WRes(theResource, itsParams);
6  371 -0 A    END;
6  372 --      {$ENDC}
6  373 --
6  374 --
6  375 --      {$S DlgClose}
6  376 --      {-----------------------------------+
6  377 --      |   Free                            |
6  378 --      +-----------------------------------}
6  379 -- A    PROCEDURE TDialogView.Free; OVERRIDE;
6  380 --
6  381 0- A    BEGIN
6  382 --          FreeObject(fParamTxt);
6  383 --          FreeObject(fTEView);
6  384 --          INHERITED Free;
6  385 -0 A    END;
6  386 --
6  387 --
6  388 --      {$S DlgRes}
6  389 --      {-----------------------------------+
6  390 --      |   CanDismiss                      |
6  391 --      +-----------------------------------}
6  392 -- A    FUNCTION  TDialogView.CanDismiss (dismissing: IDType): BOOLEAN;
6  393 0- A    BEGIN
6  394 --          CanDismiss := TRUE;
6  395 --          IF (LONGINT(fCancelItem) = LONGINT(kNoIdentifier)) | (dismissing <> fCancelItem) THEN
```

```
6  396 --              IF (fCurrentEditText <> NIL) & NOT fCurrentEditText.Validate THEN
6  397 1-                  BEGIN
6  398 --                  CanDismiss := FALSE;
6  399 --                  CantDeselect(fCurrentEditText);
6  400 -1                  END;
6  401 -0 A    END;
6  402 --
6  403 --
6  404 --      {$S DlgNonRes}
6  405 --      {-----------------------------------+
6  406 --      |   CantDeselect                    |
6  407 --      +-----------------------------------}
6  408 -- A    PROCEDURE TDialogView.CantDeselect (theEditText: TEditText);
6  409 --
6  410 0- A    BEGIN
6  411 --          SysBeep(2);
6  412 --          theEditText.RestartEdit;
6  413 -0 A    END;
6  414 --
6  415 --
6  416 --      {$S DlgClose}
6  417 --      {-----------------------------------+
6  418 --      |   Close                           |
6  419 --      +-----------------------------------}
6  420 -- A    PROCEDURE TDialogView.Close; OVERRIDE;
6  421 --
6  422 0- A    BEGIN
6  423 --          IF LONGINT(fDismisser) = LONGINT(kNoIdentifier) THEN
6  424 --              IF CanDismiss(fDefaultItem) THEN
6  425 --                  DismissDialog(fDefaultItem)
6  426 --              ELSE
6  427 --                  Failure(0, 0);              { Don't close the window }
6  428 -0 A    END;
6  429 --
6  430 --
6  431 --      {$S DlgClose}
6  432 --      {-----------------------------------+
6  433 --      |   DismissDialog                   |
6  434 --      +-----------------------------------}
6  435 -- A    PROCEDURE TDialogView.DismissDialog (dismisser: IDType);
6  436 --
6  437 0- A    BEGIN
6  438 --          { Override to do whatever your dialog is supposed to end up doing }
6  439 --          fDismissed := TRUE;
6  440 --          fDismisser := dismisser;
6  441 -0 A    END;
6  442 --
6  443 --      {$S DlgRes}
6  444 --      {-----------------------------------+
6  445 --      |   DoChoice                        |
6  446 --      +-----------------------------------}
6  447 -- A    PROCEDURE TDialogView.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
6  448 0- A    BEGIN
6  449 1-          CASE itsChoice OF
6  450 --              mEditTextHit:
6  451 2-                  BEGIN
6  452 --              {$IFC qDebug}
6  453 --                  IF NOT Member(origView, TEditText) THEN
6  454 --                      ProgramBreak(': Got mEditTextHit on non-TEditText view.')
6  455 --                  ELSE
6  456 --              {$ENDC}
6  457 --                  DoSelectEditText(TEditText(origView), FALSE);
6  458 -2                  END;
6  459 --              OTHERWISE
6  460 --                  IF Member(origView, TControl) & TControl(origView).fDismissesDialog THEN
6  461 2-                      BEGIN
6  462 --                      IF CanDismiss(origView.fIdentifier) THEN
6  463 --                          DismissDialog(origView.fIdentifier);
6  464 -2                      END
6  465 --                  ELSE
6  466 --                      INHERITED DoChoice(origView, itsChoice);
6  467 -1              END;
6  468 -0 A    END;
6  469 --
6  470 --      {$S DlgRes}
6  471 --      {-----------------------------------+
6  472 --      |   DoCommandKey                    |
6  473 --      +-----------------------------------}
6  474 -- A    FUNCTION  TDialogView.DoCommandKey (ch: CHAR; VAR info: EventInfo): TCommand; OVERRIDE;
6  475 0- A    BEGIN
6  476 --          IF (ch = '.') AND (LONGINT(fCancelItem) <> LONGINT(kNoIdentifier)) THEN
6  477 1-              BEGIN
6  478 --              IF CanDismiss(fCancelItem) THEN
6  479 2-                  BEGIN
6  480 --                  FlashControl(TControl(FindSubView(fCancelItem)));
6  481 --                  DismissDialog(fCancelItem);
6  482 -2                  END;
6  483 --              DoCommandKey := gNoChanges;
6  484 -1              END
6  485 --          ELSE
6  486 --              DoCommandKey := INHERITED DoCommandKey(ch, info);
6  487 -0 A    END;
6  488 --
6  489 --      {$S DlgRes}
6  490 --      {-----------------------------------+
6  491 --      |   DoKeyCommand                    |
6  492 --      +-----------------------------------}
6  493 -- A    FUNCTION  TDialogView.DoKeyCommand (ch: Char; aKeyCode: INTEGER;
6  494 --                                          VAR info: EventInfo): TCommand; OVERRIDE;
```

```
6  495 0- A   BEGIN
6  496 --         { If we get this far, nobody's handled the Tab, Enter, or Return keys, so we will       }
6  497 --         IF ch = chTab THEN
6  498 --             DoKeyCommand := HandleTab(SELF, info)
6  499 --         ELSE IF (ch = chEnter) | (ch = chReturn) THEN
6  500 --             DoKeyCommand := HandleCR(SELF, info)
6  501 --         ELSE
6  502 --             DoKeyCommand := INHERITED DoKeyCommand(ch, aKeyCode, info);
6  503 -0 A   END;
6  504 --
6  505 --     {$S DlgRes}
6  506 --     {-----------------------------------+
6  507 --     |   DoSelectEditText                |
6  508 --     +-----------------------------------}
6  509 -- A   PROCEDURE TDialogView.DoSelectEditText (theEditText: TEditText; selectChars: BOOLEAN);
6  510 --
6  511 --     VAR
6  512 --         itsWindow:      TWindow;
6  513 --
6  514 0- A   BEGIN
6  515 --         IF theEditText <> fCurrentEditText THEN            { If we're not editing this view…      }
6  516 --             IF (fCurrentEditText <> NIL) & NOT fCurrentEditText.StopEdit THEN
6  517 --                 CantDeselect(fCurrentEditText)
6  518 --             ELSE
6  519 1-                 BEGIN
6  520 --                 fCurrentEditText := theEditText;
6  521 --                 IF theEditText <> NIL THEN
6  522 --                     theEditText.StartEdit(selectChars, fTEView)
6  523 --                 ELSE
6  524 2-                     BEGIN
6  525 --                     itsWindow := GetWindow;                { Set the window's target, which sets  }
6  526 --                     IF itsWindow <> NIL THEN
6  527 --                         itsWindow.SetTarget(SELF);
6  528 -2                     END;
6  529 -1                 END
6  530 --         ELSE IF selectChars THEN                           { Make sure all the chars are selected. }
6  531 --             theEditText.SetSelection(0, MAXINT, kRedraw);
6  532 -0 A   END;
6  533 --
6  534 --     {$S DlgRes}
6  535 --     {-----------------------------------+
6  536 --     |   EachEditText                    |
6  537 --     +-----------------------------------}
6  538 -- A   PROCEDURE TDialogView.EachEditText (PROCEDURE DoToEditText (theEditText: TEditText));
6  539 --
6  540 --         {-------------------------------+
6  541 --         |   CheckSubView                |
6  542 --         +-------------------------------}
6  543 -- B       PROCEDURE CheckSubView (theSubView: TView);
6  544 --
6  545 0- B       BEGIN
6  546 --             IF MEMBER(theSubView, TEditText) THEN
6  547 --                 DoToEditText(TEditText(theSubView))
6  548 --             ELSE
6  549 --                 theSubView.EachSubView(CheckSubView);
6  550 -0 B       END;
6  551 --
6  552 0- A   BEGIN
6  553 --         EachSubView(CheckSubView);
6  554 -0 A   END;
6  555 --
6  556 --
6  557 --     {$S DlgRes}
6  558 --     {-----------------------------------+
6  559 --     |   GetDialogView                   |
6  560 --     +-----------------------------------}
6  561 -- A   FUNCTION  TDialogView.GetDialogView: TView; OVERRIDE;
6  562 0- A   BEGIN
6  563 --         GetDialogView := SELF;
6  564 -0 A   END;
6  565 --
6  566 --     {$S DlgRes}
6  567 --     {-----------------------------------+
6  568 --     |   HandleCR                        |
6  569 --     +-----------------------------------}
6  570 -- A   FUNCTION TDialogView.HandleCR (callingView: TView; VAR info: EventInfo): TCommand;
6  571 --     VAR
6  572 --         fDefaultControl:    TControl;
6  573 --         dontCare:           LONGINT;
6  574 0- A   BEGIN
6  575 --         IF LONGINT(fDefaultItem) <> LONGINT(kNoIdentifier) THEN
6  576 --             IF CanDismiss(fDefaultItem) THEN
6  577 1-                 BEGIN
6  578 --                 FlashControl(TControl(FindSubView(fDefaultItem)));
6  579 --                 DismissDialog(fDefaultItem);
6  580 -1                 END;
6  581 --         HandleCR := gNoChanges;
6  582 -0 A   END;
6  583 --
6  584 --     {$S DlgRes}
6  585 --     {-----------------------------------+
6  586 --     |   HandleTab                       |
6  587 --     +-----------------------------------}
6  588 -- A   FUNCTION TDialogView.HandleTab (callingView: TView; VAR info: EventInfo): TCommand;
6  589 --
6  590 --     VAR
6  591 --         first:          TEditText;
6  592 --         last:           TEditText;
6  593 --         next:           TEditText;
```

```
6  594 --          previous:       TEditText;
6  595 --
6  596 0- A    BEGIN
6  597 --          SurveyEditText(first, last, next, previous);
6  598 --
6  599 --          IF info.theShiftKey THEN
6  600 --              next := previous;
6  601 --
6  602 --          IF next <> NIL THEN
6  603 --              DoSelectEditText(next, TRUE);
6  604 --          HandleTab := gNoChanges;
6  605 -0 A    END;
6  606 --
6  607 --      {$S DlgOpen}
6  608 --      {-----------------------------------+
6  609 --      |   MakeTEView                      |
6  610 --      +-----------------------------------}
6  611 -- A    FUNCTION TDialogView.MakeTEView: TDialogTEView;
6  612 --
6  613 --      VAR
6  614 --          aDialogTEView:      TDialogTEView;
6  615 --
6  616 0- A    BEGIN
6  617 --          NEW(aDialogTEView);
6  618 --          FailNIL(aDialogTEView);
6  619 --          aDialogTEView.ITEView(NIL, SELF, gZeroVPt, gZeroVPt, sizeFixed, sizeFixed,
6  620 --                                gZeroRect, gSystemStyle, teJustLeft, kWithoutStyle, FALSE);
6  621 --          aDialogTEView.fHTE^^.clikLoop :=                { We want standard scrolling behavior   }
6  622 --            defClikLoopProc;
6  623 --          aDialogTEView.fEditText := NIL;
6  624 --          aDialogTEView.AutoScrolling(TRUE);
6  625 --          MakeTEView := aDialogTEView;
6  626 -0 A    END;
6  627 --
6  628 --
6  629 --      {$S DlgRes}
6  630 --      {-----------------------------------+
6  631 --      |   ParamTxt                        |
6  632 --      +-----------------------------------}
6  633 -- A    PROCEDURE TDialogView.ParamTxt (keyStr, valueStr: Str255);
6  634 0- A    BEGIN
6  635 --          fParamTxt.InsertEntry(keyStr, valueStr);
6  636 -0 A    END;
6  637 --
6  638 --
6  639 --      {$S DlgRes}
6  640 --      {-----------------------------------+
6  641 --      |   PoseModally                     |
6  642 --      +-----------------------------------}
6  643 -- A    FUNCTION  TDialogView.PoseModally: IDType;
6  644 --
6  645 --      VAR
6  646 --          itsWindow:      TWindow;
6  647 --
6  648 0- A    BEGIN
6  649 --          itsWindow := GetWindow;
6  650 --          IF itsWindow <> NIL THEN
6  651 1-              BEGIN
6  652 --              itsWindow.Open;
6  653 --              fDismissed := FALSE;
6  654 2-              REPEAT
6  655 --                  gApplication.PollEvent;
6  656 -2              UNTIL fDismissed;
6  657 --              PoseModally := fDismisser;
6  658 -1              END
6  659 --          ELSE
6  660 --              PoseModally := kNoIdentifier;
6  661 -0 A    END;
6  662 --
6  663 --      {$S DlgRes}
6  664 --      {-----------------------------------+
6  665 --      |   ReplaceText                     |
6  666 --      +-----------------------------------}
6  667 -- A    PROCEDURE TDialogView.ReplaceText (theText: Handle);
6  668 --
6  669 --          {-------------------------------+
6  670 --          |   ReplaceOnce                 |
6  671 --          +-------------------------------}
6  672 -- B        PROCEDURE ReplaceOnce(item: TObject);
6  673 --          VAR
6  674 --              offset:     LONGINT;
6  675 --              limit:      LONGINT;
6  676 0- B        BEGIN
6  677 --              offset := 1;                        { Prime starting value                          }
6  678 1-              REPEAT
6  679 --                  limit := GetHandleSize(theText) - 1;
6  680 --                  WITH TEntry(item) DO
6  681 --                      offset := Munger(theText, offset,
6  682 --                        Ptr(ORD4(fKey^)+1), LENGTH(fKey^^), Ptr(ORD4(fValue^)+1), LENGTH(fValue^^));
6  683 -1              UNTIL (offset < 0) | (offset >= limit);
6  684 --              {$IFC qRangeCheck}{$R-}{$ENDC}
6  685 --              theText^^ := GetHandleSize(theText) - 1;
6  686 --              {$IFC qRangeCheck}{$R+}{$ENDC}
6  687 -0 B        END;
6  688 --
6  689 0- A    BEGIN
6  690 --          fParamTxt.fEntries.Each(ReplaceOnce);
6  691 -0 A    END;
6  692 --
```

```
6  693 --
6  694 --      {$S DlgRes}
6  695 --      {-----------------------------------+
6  696 --      |   SelectEditText                  |
6  697 --      +-----------------------------------}
6  698 -- A    PROCEDURE TDialogView.SelectEditText (itsIdentifier: IDType; selectChars: BOOLEAN);
6  699 --
6  700 --      VAR
6  701 --          aSubView:       TView;
6  702 --
6  703 0- A    BEGIN
6  704 --          aSubView := FindSubView(itsIdentifier);
6  705 --          IF (aSubView <> NIL) & (Member(aSubView, TEditText)) THEN
6  706 --              DoSelectEditText(TEditText(aSubView), selectChars);
6  707 -0 A    END;
6  708 --
6  709 --      {$S DlgRes}
6  710 --      {-----------------------------------+
6  711 --      |   SurveyEditText                  |
6  712 --      +-----------------------------------}
6  713 -- A    PROCEDURE TDialogView.SurveyEditText (VAR first, last, next, previous: TEditText);
6  714 --
6  715 --      VAR
6  716 --          foundCurrent:           BOOLEAN;
6  717 --
6  718 --          {-------------------------------+
6  719 --          |   Survey                      |
6  720 --          +-------------------------------}
6  721 -- B        PROCEDURE Survey (theEditText: TEditText);
6  722 --
6  723 0- B        BEGIN
6  724 --              IF first = NIL THEN
6  725 --                  first := theEditText;
6  726 --              last := theEditText;
6  727 --              IF theEditText = fCurrentEditText THEN
6  728 --                  foundCurrent := TRUE
6  729 --              ELSE IF foundCurrent & (next = NIL) THEN
6  730 --                  next := theEditText;
6  731 --              IF NOT foundCurrent THEN
6  732 --                  previous := theEditText;
6  733 -0 B        END;
6  734 --
6  735 0- A    BEGIN
6  736 --          foundCurrent := FALSE;
6  737 --          next := NIL;
6  738 --          previous := NIL;
6  739 --          first := NIL;
6  740 --          last := NIL;
6  741 --          EachEditText(Survey);
6  742 --          IF next = NIL THEN
6  743 --              next := first;
6  744 --          IF previous = NIL THEN
6  745 --              previous := last;
6  746 -0 A    END;
6  747 --
6  748 --
6  749 --      {$IFC qDebug}
6  750 --          {$S DlgFields}
6  751 --          {-----------------------------------+
6  752 --          |   Fields                          |
6  753 --          +-----------------------------------}
6  754 -- A        PROCEDURE TDialogView.Fields (PROCEDURE DoToField (fieldName: Str255;
6  755 --                                        fieldAddr: Ptr;
6  756 --                                        fieldType: INTEGER)); OVERRIDE;
6  757 0- A        BEGIN
6  758 --              DoToField('TDialogView', NIL, bClass);
6  759 --              DoToField('fDefaultItem', @fDefaultItem, bOSType);
6  760 --              DoToField('fCancelItem', @fCancelItem, bOSType);
6  761 --              DoToField('fParamTxt', @fParamTxt, bObject);
6  762 --              DoToField('fCurrentEditText', @fCurrentEditText, bObject);
6  763 --              DoToField('fTEView', @fTEView, bObject);
6  764 --              DoToField('fDismissed', @fDismissed, bBoolean);
6  765 --              DoToField('fDismisser', @fDismisser, bOSType);
6  766 --              INHERITED Fields(DoToField);
6  767 -0 A        END;
6  768 --
6  769 --      {$ENDC}
6  770 --
6  771 --      (*****************************************************************************************)
6  772 --      (*      T B u t t o n                                                                    *)
6  773 --      (*****************************************************************************************)
6  774 --
6  775 --      {$IFC qProceduralViews}
6  776 --      {$S DlgOpen}
6  777 --      {-----------------------------------+
6  778 --      |   IButton                         |
6  779 --      +-----------------------------------}
6  780 -- A    PROCEDURE TButton.IButton (itsSuperView: TView; itsLocation, itsSize: VPoint;
6  781 --                                 itsHSizeDet, itsVSizeDet: SizeDeterminer;
6  782 --                                 itsLabel: Str255);
6  783 0- A    BEGIN
6  784 --          ICtlMgr(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet,
6  785 --            itsLabel, 0, 0, 0, pushButProc);
6  786 --          fDefChoice := mButtonHit;
6  787 -0 A    END;
6  788 --      {$ENDC}
6  789 --
6  790 --
6  791 --      {$IFC qTemplateViews}
```

```
6  792 --      {$S DlgOpen}
6  793 --      {-----------------------------------+
6  794 --      |   IRes                            |
6  795 --      +-----------------------------------}
6  796 -- A    PROCEDURE TButton.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6  797 --
6  798 --      VAR
6  799 --          itsArea:        Rect;
6  800 --
6  801 0- A    BEGIN
6  802 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6  803 --
6  804 --          fDefChoice := mButtonHit;
6  805 --          ControlArea(itsArea);
6  806 --          WITH ButtonTemplatePtr(itsParams)^ DO
6  807 --              CreateCMgrControl(itsArea, itsLabel, 0, 0, 0, pushButProc);
6  808 --
6  809 --          OffsetPtrWStr(itsParams, SIZEOF(ButtonTemplate));
6  810 -0 A    END;
6  811 --      {$ENDC}
6  812 --
6  813 --
6  814 --      {$IFC qTemplateViews}
6  815 --      {$S MAWriteRes}
6  816 --      {-----------------------------------+
6  817 --      |   WRes                            |
6  818 --      +-----------------------------------}
6  819 -- A    PROCEDURE TButton.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  820 --
6  821 --      VAR
6  822 --          theLabel:   Str255;
6  823 --          btPtr:      ButtonTemplatePtr;
6  824 --
6  825 0- A    BEGIN
6  826 --          INHERITED WRes(theResource, itsParams);
6  827 --
6  828 --          GetText(theLabel);
6  829 --
6  830 --          btPtr := ButtonTemplatePtr(ExpandPtrWStr(theResource, itsParams,
6  831 --            SIZEOF(ButtonTemplate), LENGTH(theLabel)));
6  832 --
6  833 --          (* btPtr^.itsLabel := theLabel; *)
6  834 --          CopyStr255(theLabel, PRStr(btPtr^.itsLabel));
6  835 -0 A    END;
6  836 --      {$ENDC}
6  837 --
6  838 --
6  839 --      {$IFC qTemplateViews}
6  840 --      {$S MAWriteRes}
6  841 --      {-----------------------------------+
6  842 --      |   WriteRes                        |
6  843 --      +-----------------------------------}
6  844 -- A    PROCEDURE TButton.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  845 0- A    BEGIN
6  846 --          gWResSignature := 'butn';   gWResType := 'TButton';
6  847 --          WRes(theResource, itsParams);
6  848 -0 A    END;
6  849 --      {$ENDC}
6  850 --
6  851 --
6  852 --      (*****************************************************************************************)
6  853 --      (*      T C h e c k B o x                                                                *)
6  854 --      (*****************************************************************************************)
6  855 --
6  856 --      {$IFC qProceduralViews}
6  857 --      {$S DlgOpen}
6  858 --      {-----------------------------------+
6  859 --      |   ICheckBox                       |
6  860 --      +-----------------------------------}
6  861 -- A    PROCEDURE TCheckBox.ICheckBox (itsSuperView: TView; itsLocation, itsSize: VPoint;
6  862 --                                     itsHSizeDet, itsVSizeDet: SizeDeterminer;
6  863 --                                     itsLabel: Str255; isTurnedOn: BOOLEAN);
6  864 0- A    BEGIN
6  865 --          ICtlMgr(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet,
6  866 --            itsLabel, 0, 0, 1, checkBoxProc);
6  867 --          SetState(isTurnedOn, kDontRedraw);
6  868 --          fDefChoice := mCheckBoxHit;
6  869 -0 A    END;
6  870 --      {$ENDC}
6  871 --
6  872 --
6  873 --      {$IFC qTemplateViews}
6  874 --      {$S DlgOpen}
6  875 --      {-----------------------------------+
6  876 --      |   IRes                            |
6  877 --      +-----------------------------------}
6  878 -- A    PROCEDURE TCheckBox.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6  879 --
6  880 --      VAR
6  881 --          itsArea:        Rect;
6  882 --
6  883 0- A    BEGIN
6  884 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6  885 --
6  886 --          fDefChoice := mCheckBoxHit;
6  887 --          ControlArea(itsArea);
6  888 --          WITH CheckBoxTemplatePtr(itsParams)^ DO
6  889 --              CreateCMgrControl(itsArea, itsLabel, ORD(isOn), 0, 1, checkBoxProc);
6  890 --
```

```
6  891 --          OffsetPtrWStr(itsParams, SIZEOF(CheckBoxTemplate));
6  892 -0 A    END;
6  893 --      {$ENDC}
6  894 --
6  895 --
6  896 --      {$IFC qWriteTemplates}
6  897 --      {$S MAWriteRes}
6  898 --      {-----------------------------------+
6  899 --      |   WRes                            |
6  900 --      +-----------------------------------}
6  901 -- A    PROCEDURE TCheckBox.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  902 --
6  903 --      VAR
6  904 --          theLabel:   Str255;
6  905 --          cbPtr:      CheckBoxTemplatePtr;
6  906 --
6  907 0- A    BEGIN
6  908 --          INHERITED WRes(theResource, itsParams);
6  909 --
6  910 --          GetText(theLabel);
6  911 --
6  912 --          cbPtr := CheckBoxTemplatePtr(ExpandPtrWStr(theResource, itsParams,
6  913 --            SIZEOF(CheckBoxTemplate), LENGTH(theLabel)));
6  914 --
6  915 --          cbPtr^.isOn := IsOn;
6  916 --          (* cbPtr^.itsLabel := theLabel; *)
6  917 --          CopyStr255(theLabel, PRStr(cbPtr^.itsLabel));
6  918 -0 A    END;
6  919 --
6  920 --      {$S MAWriteRes}
6  921 --      {-----------------------------------+
6  922 --      |   WriteRes                        |
6  923 --      +-----------------------------------}
6  924 -- A    PROCEDURE TCheckBox.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6  925 0- A    BEGIN
6  926 --          gWResSignature := 'chkb';   gWResType := 'TCheckBox';
6  927 --          WRes(theResource, itsParams);
6  928 -0 A    END;
6  929 --      {$ENDC}
6  930 --
6  931 --
6  932 --      {$S DlgRes}
6  933 --      {-----------------------------------+
6  934 --      |   DoChoice                        |
6  935 --      +-----------------------------------}
6  936 -- A    PROCEDURE TCheckBox.DoChoice(origView: TView; itsChoice: INTEGER);
6  937 0- A    BEGIN
6  938 --          IF itsChoice = mCheckBoxHit THEN
6  939 --              Toggle(kRedraw);
6  940 --          INHERITED DoChoice(origView, itsChoice);
6  941 -0 A    END;
6  942 --
6  943 --      {$S DlgRes}
6  944 --      {-----------------------------------+
6  945 --      |   IsOn                            |
6  946 --      +-----------------------------------}
6  947 -- A    FUNCTION  TCheckBox.IsOn: BOOLEAN;
6  948 0- A    BEGIN
6  949 --          IsOn := GetVal <> 0;
6  950 -0 A    END;
6  951 --
6  952 --      {$S DlgRes}
6  953 --      {-----------------------------------+
6  954 --      |   SetState                        |
6  955 --      +-----------------------------------}
6  956 -- A    PROCEDURE TCheckBox.SetState (state, redraw: BOOLEAN);
6  957 0- A    BEGIN
6  958 --          SetVal(ORD(state), redraw);
6  959 -0 A    END;
6  960 --
6  961 --      {$S DlgRes}
6  962 --      {-----------------------------------+
6  963 --      |   Toggle                          |
6  964 --      +-----------------------------------}
6  965 -- A    PROCEDURE TCheckBox.Toggle (redraw: BOOLEAN);
6  966 0- A    BEGIN
6  967 --          SetVal(ORD(NOT IsOn), redraw);
6  968 -0 A    END;
6  969 --
6  970 --      {$S DlgRes}
6  971 --      {-----------------------------------+
6  972 --      |   ToggleIf                        |
6  973 --      +-----------------------------------}
6  974 -- A    PROCEDURE TCheckBox.ToggleIf (matchState, redraw: BOOLEAN);
6  975 0- A    BEGIN
6  976 --          IF IsOn = matchState THEN
6  977 --              SetVal(ORD(NOT IsOn), redraw);
6  978 -0 A    END;
6  979 --
6  980 --      (*****************************************************************************************)
6  981 --      (*      T R a d i o                                                                      *)
6  982 --      (*****************************************************************************************)
6  983 --
6  984 --      {$IFC qProceduralViews}
6  985 --      {$S DlgOpen}
6  986 --      {-----------------------------------+
6  987 --      |   IRadio                          |
6  988 --      +-----------------------------------}
6  989 -- A    PROCEDURE TRadio.IRadio (itsSuperView: TView; itsLocation, itsSize: VPoint;
```

```
6  990 --                              itsHSizeDet, itsVSizeDet: SizeDeterminer;
6  991 --                              itsLabel: Str255; isTurnedOn: BOOLEAN);
6  992 0- A   BEGIN
6  993 --         ICtlMgr(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet,
6  994 --           itsLabel, 0, 0, 1, radioButProc);
6  995 --         SetState(isTurnedOn, kDontRedraw);
6  996 --         fDefChoice := mRadioHit;
6  997 -0 A   END;
6  998 --     {$ENDC}
6  999 --
6 1000 --
6 1001 --     {$IFC qTemplateViews}
6 1002 --     {$S DlgOpen}
6 1003 --     {-----------------------------------+
6 1004 --     |   IRes                            |
6 1005 --     +-----------------------------------}
6 1006 -- A   PROCEDURE TRadio.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 1007 --
6 1008 --     VAR
6 1009 --         itsArea:    Rect;
6 1010 --
6 1011 0- A   BEGIN
6 1012 --         INHERITED IRes(NIL, itsSuperView, itsParams);
6 1013 --
6 1014 --         fDefChoice := mRadioHit;
6 1015 --         ControlArea(itsArea);
6 1016 --         WITH RadioTemplatePtr(itsParams)^ DO
6 1017 --             CreateCMgrControl(itsArea, itsLabel, ORD(isOn), 0, 1, radioButProc);
6 1018 --         OffsetPtrWStr(itsParams, SIZEOF(RadioTemplate));
6 1019 -0 A   END;
6 1020 --     {$ENDC}
6 1021 --
6 1022 --
6 1023 --     {$IFC qWriteTemplates}
6 1024 --     {$S MAWriteRes}
6 1025 --     {-----------------------------------+
6 1026 --     |   WRes                            |
6 1027 --     +-----------------------------------}
6 1028 -- A   PROCEDURE TRadio.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1029 --
6 1030 --     VAR
6 1031 --         theLabel:   Str255;
6 1032 --         rdPtr:      RadioTemplatePtr;
6 1033 --
6 1034 0- A   BEGIN
6 1035 --         INHERITED WRes(theResource, itsParams);
6 1036 --
6 1037 --         GetText(theLabel);
6 1038 --
6 1039 --         rdPtr := RadioTemplatePtr(ExpandPtrWStr(theResource, itsParams,
6 1040 --           SIZEOF(RadioTemplate), LENGTH(theLabel)));
6 1041 --
6 1042 --         rdPtr^.isOn := IsOn;
6 1043 --         (* rdPtr^.itsLabel := theLabel; *)
6 1044 --         CopyStr255(theLabel, PRStr(rdPtr^.itsLabel));
6 1045 -0 A   END;
6 1046 --
6 1047 --     {$S MAWriteRes}
6 1048 --     {-----------------------------------+
6 1049 --     |   WriteRes                        |
6 1050 --     +-----------------------------------}
6 1051 -- A   PROCEDURE TRadio.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1052 0- A   BEGIN
6 1053 --         gWResSignature := 'radb';   gWResType := 'TRadio';
6 1054 --         WRes(theResource, itsParams);
6 1055 -0 A   END;
6 1056 --     {$ENDC}
6 1057 --
6 1058 --
6 1059 --     {$S DlgRes}
6 1060 --     {-----------------------------------+
6 1061 --     |   DoChoice                        |
6 1062 --     +-----------------------------------}
6 1063 -- A   PROCEDURE TRadio.DoChoice(origView: TView; itsChoice: INTEGER);
6 1064 0- A   BEGIN
6 1065 --         IF (itsChoice = mRadioHit) & NOT IsOn THEN
6 1066 1-             BEGIN
6 1067 --             Toggle(kRedraw);
6 1068 --             INHERITED DoChoice(origView, itsChoice);    { Should this be called regardless??? }
6 1069 -1             END;
6 1070 -0 A   END;
6 1071 --
6 1072 --     {$S DlgRes}
6 1073 --     {-----------------------------------+
6 1074 --     |   IsOn                            |
6 1075 --     +-----------------------------------}
6 1076 -- A   FUNCTION  TRadio.IsOn: BOOLEAN;
6 1077 0- A   BEGIN
6 1078 --         IsOn := GetVal <> 0;
6 1079 -0 A   END;
6 1080 --
6 1081 --     {$S DlgRes}
6 1082 --     {-----------------------------------+
6 1083 --     |   SetState                        |
6 1084 --     +-----------------------------------}
6 1085 -- A   PROCEDURE TRadio.SetState (state, redraw: BOOLEAN);
6 1086 0- A   BEGIN
6 1087 --         SetVal(ORD(state), redraw);
6 1088 -0 A   END;
```

```
6 1089 --
6 1090 --      {$S DlgRes}
6 1091 --      {-----------------------------------+
6 1092 --      |   Toggle                          |
6 1093 --      +-----------------------------------}
6 1094 -- A    PROCEDURE TRadio.Toggle (redraw: BOOLEAN);
6 1095 0- A    BEGIN
6 1096 --          SetVal(ORD(NOT IsOn), redraw);
6 1097 -0 A    END;
6 1098 --
6 1099 --      {$S DlgRes}
6 1100 --      {-----------------------------------+
6 1101 --      |   ToggleIf                        |
6 1102 --      +-----------------------------------}
6 1103 -- A    PROCEDURE TRadio.ToggleIf (matchState, redraw: BOOLEAN);
6 1104 0- A    BEGIN
6 1105 --          IF IsOn = matchState THEN
6 1106 --              SetVal(ORD(NOT IsOn), redraw);
6 1107 -0 A    END;
6 1108 --
6 1109 --
6 1110 --      (*****************************************************************************************)
6 1111 --      (*      T C l u s t e r                                                                  *)
6 1112 --      (*****************************************************************************************)
6 1113 --
6 1114 --      {$IFC qProceduralViews}
6 1115 --      {$S DlgOpen}
6 1116 --      {-----------------------------------+
6 1117 --      |   ICluster                        |
6 1118 --      +-----------------------------------}
6 1119 -- A    PROCEDURE TCluster.ICluster (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 1120 --                                   itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 1121 --                                   itsRsrcID, itsIndex: INTEGER);
6 1122 --      VAR
6 1123 --          aString:        Str255;
6 1124 --          fi:             FailInfo;
6 1125 --
6 1126 --          {--------------------------------+
6 1127 --          |   HandleFailure                |
6 1128 --          +--------------------------------}
6 1129 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1130 0- B        BEGIN
6 1131 --              Free;
6 1132 -0 B        END;
6 1133 --
6 1134 0- A    BEGIN
6 1135 --          fDataHandle := NIL;
6 1136 --          IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 1137 --          fRsrcID := itsRsrcID;
6 1138 --          fIndex := itsIndex;
6 1139 --          IF fRsrcID <> kNoResource THEN
6 1140 1-              BEGIN
6 1141 --              CatchFailures(fi, HandleFailure);
6 1142 --              GetIndString(aString, fRsrcID, fIndex);
6 1143 --              FailResError;
6 1144 --              Success(fi);
6 1145 --              SetLabel(aString, kDontRedraw);
6 1146 -1              END;
6 1147 --          ViewEnable(FALSE, kDontRedraw);             { Default is not to enable hit testing          }
6 1148 --          fDefChoice := mClusterHit;
6 1149 -0 A    END;
6 1150 --      {$ENDC}
6 1151 --
6 1152 --
6 1153 --      {$IFC qTemplateViews}
6 1154 --      {$S DlgOpen}
6 1155 --      {-----------------------------------+
6 1156 --      |   IRes                            |
6 1157 --      +-----------------------------------}
6 1158 -- A    PROCEDURE TCluster.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 1159 --
6 1160 0- A    BEGIN
6 1161 --          fDataHandle := NIL;
6 1162 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6 1163 --          fDefChoice := mClusterHit;
6 1164 --
6 1165 --          WITH ClusterTemplatePtr(itsParams)^ DO
6 1166 --              SetLabel(itsLabel, kDontRedraw);
6 1167 --
6 1168 --          OffsetPtrWStr(itsParams, SIZEOF(ClusterTemplate));
6 1169 -0 A    END;
6 1170 --      {$ENDC}
6 1171 --
6 1172 --
6 1173 --      {$IFC qWriteTemplates}
6 1174 --      {$S MAWriteRes}
6 1175 --      {-----------------------------------+
6 1176 --      |   WRes                            |
6 1177 --      +-----------------------------------}
6 1178 -- A    PROCEDURE TCluster.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1179 --
6 1180 --      VAR
6 1181 --          theLabel:   Str255;
6 1182 --          clPtr:      ClusterTemplatePtr;
6 1183 --
6 1184 0- A    BEGIN
6 1185 --          INHERITED WRes(theResource, itsParams);
6 1186 --
6 1187 --          GetLabel(theLabel);
```

```
6 1188 --
6 1189 --          clPtr := ClusterTemplatePtr(ExpandPtrWStr(theResource, itsParams,
6 1190 --            SIZEOF(ClusterTemplate), LENGTH(theLabel)));
6 1191 --
6 1192 --          (* clPtr^.itsLabel := theLabel; *)
6 1193 --          CopyStr255(theLabel, PRStr(clPtr^.itsLabel));
6 1194 -0 A    END;
6 1195 --
6 1196 --      {$S MAWriteRes}
6 1197 --      {-----------------------------------+
6 1198 --      |   WriteRes                        |
6 1199 --      +-----------------------------------}
6 1200 -- A    PROCEDURE TCluster.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1201 0- A    BEGIN
6 1202 --          gWResSignature := 'clus';   gWResType := 'TCluster';
6 1203 --          WRes(theResource, itsParams);
6 1204 -0 A    END;
6 1205 --      {$ENDC}
6 1206 --
6 1207 --
6 1208 --      {$S DlgClose}
6 1209 --      {-----------------------------------+
6 1210 --      |   Free                            |
6 1211 --      +-----------------------------------}
6 1212 -- A    PROCEDURE TCluster.Free; OVERRIDE;
6 1213 0- A    BEGIN
6 1214 --          ReleaseLabel;
6 1215 --          INHERITED Free;
6 1216 -0 A    END;
6 1217 --
6 1218 --      {$S DlgRes}
6 1219 --      {-----------------------------------+
6 1220 --      |   DoChoice                        |
6 1221 --      +-----------------------------------}
6 1222 -- A    PROCEDURE TCluster.DoChoice (origView: TView; itsChoice: INTEGER); OVERRIDE;
6 1223 --
6 1224 --          {-------------------------------+
6 1225 --          |   ResetRadios                 |
6 1226 --          +-------------------------------}
6 1227 -- B        PROCEDURE ResetRadios(aView: TView);
6 1228 0- B        BEGIN
6 1229 --              IF MEMBER(aView, TRadio) AND        { If the subview is a TRadio, and…          }
6 1230 --                (aView <> origView) THEN          { …it's not the calling radio…              }
6 1231 --                  TRadio(aView).SetState(FALSE, kRedraw); { …set it off and redraw it         }
6 1232 -0 B        END;
6 1233 --
6 1234 0- A    BEGIN
6 1235 --          IF (itsChoice = mRadioHit) AND          { If we got this far, a radio's changed state }
6 1236 --            (origView.fSuperView = SELF) THEN     { Only worry about it if it's _our_ subview!  }
6 1237 --              EachSubView(ResetRadios);           { Reset everybody except the calling radio    }
6 1238 --          INHERITED DoChoice(origView, itsChoice);
6 1239 -0 A    END;
6 1240 --
6 1241 --      {$S DlgRes}
6 1242 --      {-----------------------------------+
6 1243 --      |   Draw                            |
6 1244 --      +-----------------------------------}
6 1245 -- A    PROCEDURE TCluster.Draw (area: Rect); OVERRIDE;
6 1246 --      VAR
6 1247 --          fontHt:             INTEGER;
6 1248 --          labelWd:            INTEGER;
6 1249 --          oldTop:             INTEGER;
6 1250 --          err:                OSErr;
6 1251 --          fInfo:              FontInfo;
6 1252 --          theFrame:           Rect;
6 1253 --          realText:           StringHandle;
6 1254 --          aDialogView:        TDialogView;
6 1255 --          oldColor:           RGBColor;
6 1256 --
6 1257 0- A    BEGIN
6 1258 --          IF fDataHandle <> NIL THEN
6 1259 1-              BEGIN
6 1260 --              PenNormal;                          { Normalcy in most things                   }
6 1261 --              PenSize(fPenSize.h, fPenSize.v); (* PenSize(2, 2); *)
6 1262 --              GetIfColor(oldColor);               { Save the original pen color               }
6 1263 --              SetPortTextStyle(fTextStyle);       { Install the object's TextStyle            }
6 1264 --              GetFontInfo(fInfo);                 { Determine label's height                  }
6 1265 --              fontHt := fInfo.ascent + fInfo.descent + fInfo.leading;
6 1266 --              ControlArea(theFrame);              { Get the control's extent                  }
6 1267 --              oldTop := theFrame.top;
6 1268 --              InsetRect(theFrame,                 { Inset the frame a little                  }
6 1269 --                fPenSize.h + 1, fPenSize.v + 1);
6 1270 --              theFrame.top := oldTop + BSR(fontHt, 1); { Bump top so it cuts label in half    }
6 1271 --
6 1272 --              FrameRect(theFrame);                { Draw the frame                            }
6 1273 --
6 1274 --              realText := fDataHandle;            { Make a copy of the original text          }
6 1275 --              err := HandToHand(Handle(realText));{ We won't die just because we can't copy   }
6 1276 --              IF err <> noErr THEN                { We'll just use original in an emergency   }
6 1277 --                  realText := fDataHandle
6 1278 --              ELSE
6 1279 2-                  BEGIN
6 1280 --                  aDialogView := TDialogView(GetDialogView);
6 1281 --                  IF aDialogView <> NIL THEN
6 1282 --                      aDialogView.ReplaceText(Handle(realText));
6 1283 -2                  END;
6 1284 --              labelWd := StringWidth(realText^^) + 8;
6 1285 --              SetRect(theFrame, 16, 0, labelWd + 16, fontHt);
6 1286 --              HLock(Handle(realText));
```

```
6 1287 --              TextBox(Ptr(ORD4(realText^) + 1),   { Draw the label                          }
6 1288 --                LENGTH(realText^^), theFrame, teJustCenter);
6 1289 --              HUnlock(Handle(realText));
6 1290 --              IF err = noErr THEN                 { Dispose the handle if we could create it }
6 1291 --                  DisposHandle(Handle(realText));
6 1292 --              SetIfColor(oldColor);
6 1293 -1              END;
6 1294 --          INHERITED Draw(area);                   { Let parents have a chance to draw too    }
6 1295 -0 A    END;
6 1296 --
6 1297 --      {$S DlgNonRes}
6 1298 --      {-----------------------------------+
6 1299 --      |   GetLabel                        |
6 1300 --      +-----------------------------------}
6 1301 -- A    PROCEDURE TCluster.GetLabel (VAR theLabel: Str255);
6 1302 0- A    BEGIN
6 1303 --          IF fDataHandle <> NIL THEN
6 1304 --              theLabel := fDataHandle^^
6 1305 --          ELSE
6 1306 --              theLabel := '';
6 1307 -0 A    END;
6 1308 --
6 1309 --      {$S DlgNonRes}
6 1310 --      {-----------------------------------+
6 1311 --      |   ReleaseLabel                    |
6 1312 --      +-----------------------------------}
6 1313 -- A    PROCEDURE TCluster.ReleaseLabel;
6 1314 0- A    BEGIN
6 1315 --          DisposIfHandle(fDataHandle);
6 1316 --          fRsrcID := kNoResource;
6 1317 --          fDataHandle := NIL;
6 1318 -0 A    END;
6 1319 --
6 1320 --      {$S DlgRes}
6 1321 --      {-----------------------------------+
6 1322 --      |   ReportCurrent                   |
6 1323 --      +-----------------------------------}
6 1324 -- A    FUNCTION  TCluster.ReportCurrent: IDType;
6 1325 --      VAR
6 1326 --          rView:      TView;
6 1327 --
6 1328 --          {-------------------------------+
6 1329 --          |   FindRadio                   |
6 1330 --          +-------------------------------}
6 1331 -- B        FUNCTION  FindRadio(aView: TView): BOOLEAN;
6 1332 0- B        BEGIN
6 1333 --              FindRadio := MEMBER(aView, TRadio) AND TRadio(aView).IsOn;
6 1334 -0 B        END;
6 1335 --
6 1336 0- A    BEGIN
6 1337 --          rView := FirstSubViewThat(FindRadio);
6 1338 --          IF rView <> NIL THEN
6 1339 --              ReportCurrent := rView.fIdentifier
6 1340 --          ELSE
6 1341 --              ReportCurrent := kNoIdentifier;
6 1342 -0 A    END;
6 1343 --
6 1344 --      {$S DlgNonRes}
6 1345 --      {-----------------------------------+
6 1346 --      |   SetLabel                        |
6 1347 --      +-----------------------------------}
6 1348 -- A    PROCEDURE TCluster.SetLabel(theLabel: Str255; redraw: BOOLEAN);
6 1349 --
6 1350 0- A    BEGIN
6 1351 --          ReleaseLabel;
6 1352 --          IF theLabel <> '' THEN
6 1353 1-              BEGIN
6 1354 --              fDataHandle := NewString(theLabel);
6 1355 --              IF MemError <> noErr THEN
6 1356 --                  fDataHandle := NIL;
6 1357 -1              END;
6 1358 --          IF redraw THEN
6 1359 --              ForceRedraw;
6 1360 -0 A    END;
6 1361 --
6 1362 --      {$IFC qDebug}
6 1363 --          {$S DlgFields}
6 1364 --          {-----------------------------------+
6 1365 --          |   Fields                          |
6 1366 --          +-----------------------------------}
6 1367 -- A        PROCEDURE TCluster.Fields (PROCEDURE DoToField (fieldName: Str255;
6 1368 --                                                         fieldAddr: Ptr;
6 1369 --                                                         fieldType: INTEGER)); OVERRIDE;
6 1370 0- A        BEGIN
6 1371 --              DoToField('TCluster', NIL, bClass);
6 1372 --              DoToField('fRsrcID', @fRsrcID, bInteger);
6 1373 --              DoToField('fIndex', @fIndex, bInteger);
6 1374 --              DoToField('fDataHandle', @fDataHandle, bHandle);
6 1375 --              INHERITED Fields(DoToField);
6 1376 -0 A        END;
6 1377 --
6 1378 --      {$ENDC}
6 1379 --
6 1380 --
6 1381 --      (*****************************************************************************************)
6 1382 --      (*      T I c o n                                                                        *)
6 1383 --      (*****************************************************************************************)
6 1384 --
6 1385 --      {$IFC qProceduralViews}
```

```
6 1386 --      {$S DlgOpen}
6 1387 --      {-----------------------------------+
6 1388 --      |   IIcon                           |
6 1389 --      +-----------------------------------}
6 1390 -- A    PROCEDURE TIcon.IIcon (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 1391 --                             itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 1392 --                             itsRsrcID: INTEGER; preferColor: BOOLEAN);
6 1393 --      VAR
6 1394 --          fi:                 FailInfo;
6 1395 --
6 1396 --          {-------------------------------+
6 1397 --          |   HandleFailure               |
6 1398 --          +-------------------------------}
6 1399 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1400 0- B        BEGIN
6 1401 --              Free;
6 1402 -0 B        END;
6 1403 --
6 1404 0- A    BEGIN
6 1405 --          fDataHandle := NIL;
6 1406 --          IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 1407 --          fPreferColor := preferColor;
6 1408 --          fRsrcID := itsRsrcID;
6 1409 --          IF fRsrcID <> kNoResource THEN
6 1410 1-              BEGIN
6 1411 --              CatchFailures(fi, HandleFailure);
6 1412 --              IF fPreferColor AND gConfiguration.hasColorQD THEN
6 1413 --                  fDataHandle := Handle(GetCIcon(fRsrcID));
6 1414 --              IF fDataHandle = NIL THEN
6 1415 2-                  BEGIN
6 1416 --                  fPreferColor := NOT kPreferColor;   { Either can't or won't                    }
6 1417 --                  fDataHandle := GetIcon(fRsrcID);
6 1418 -2                  END;
6 1419 --              FailResError;
6 1420 --              Success(fi);
6 1421 -1              END;
6 1422 --          ViewEnable(FALSE, kDontRedraw);             { Default is to not enable hit testing          }
6 1423 --          fDefChoice := mIconHit;
6 1424 -0 A    END;
6 1425 --      {$ENDC}
6 1426 --
6 1427 --
6 1428 --      {$IFC qTemplateViews}
6 1429 --      {$S DlgOpen}
6 1430 --      {-----------------------------------+
6 1431 --      |   IRes                            |
6 1432 --      +-----------------------------------}
6 1433 -- A    PROCEDURE TIcon.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 1434 --
6 1435 --      VAR
6 1436 --          fi:                 FailInfo;
6 1437 --
6 1438 --          {-------------------------------+
6 1439 --          |   HandleFailure               |
6 1440 --          +-------------------------------}
6 1441 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1442 0- B        BEGIN
6 1443 --              Free;
6 1444 -0 B        END;
6 1445 --
6 1446 0- A    BEGIN
6 1447 --          fDataHandle := NIL;
6 1448 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6 1449 --
6 1450 --          WITH IconTemplatePtr(itsParams)^ DO
6 1451 1-              BEGIN
6 1452 --              fPreferColor := preferColor;
6 1453 --              frsrcID := rsrcID;
6 1454 -1              END;
6 1455 --          IF fRsrcID <> kNoResource THEN
6 1456 1-              BEGIN
6 1457 --              CatchFailures(fi, HandleFailure);
6 1458 --              IF fPreferColor AND gConfiguration.hasColorQD THEN
6 1459 --                  fDataHandle := Handle(GetCIcon(fRsrcID));
6 1460 --              IF fDataHandle = NIL THEN
6 1461 2-                  BEGIN
6 1462 --                  fPreferColor := NOT kPreferColor;   { Either can't or won't                    }
6 1463 --                  fDataHandle := GetIcon(fRsrcID);
6 1464 -2                  END;
6 1465 --              { Don't die because resource not found - just return NIL handle }
6 1466 --              FailResError;
6 1467 --              Success(fi);
6 1468 -1              END;
6 1469 --          fDefChoice := mIconHit;
6 1470 --
6 1471 --          OffsetPtr(itsParams, SIZEOF(IconTemplate));
6 1472 -0 A    END;
6 1473 --      {$ENDC}
6 1474 --
6 1475 --
6 1476 --      {$IFC qWriteTemplates}
6 1477 --      {$S MAWriteRes}
6 1478 --      {-----------------------------------+
6 1479 --      |   WRes                            |
6 1480 --      +-----------------------------------}
6 1481 -- A    PROCEDURE TIcon.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1482 --
6 1483 --      VAR
6 1484 --          icPtr:      IconTemplatePtr;
```

```
6 1485 --
6 1486 0- A   BEGIN
6 1487 --          INHERITED WRes(theResource, itsParams);
6 1488 --
6 1489 --          icPtr := IconTemplatePtr(ExpandPtr(theResource, itsParams, SIZEOF(IconTemplate)));
6 1490 --
6 1491 --          WITH icPtr^ DO
6 1492 1-              BEGIN
6 1493 --              preferColor := fPreferColor;
6 1494 --      {$IFC qDebug}
6 1495 --              IF fRsrcID = kNoResource THEN
6 1496 --                  WriteLn('Tried to write TIcon with no resource ID.');
6 1497 --      {$ENDC}
6 1498 --              rsrcID := fRsrcID;
6 1499 -1              END;
6 1500 -0 A   END;
6 1501 --      {$ENDC}
6 1502 --
6 1503 --
6 1504 --      {$IFC qWriteTemplates}
6 1505 --      {$S MAWriteRes}
6 1506 --      {-----------------------------------+
6 1507 --      |   WriteRes                        |
6 1508 --      +-----------------------------------}
6 1509 -- A   PROCEDURE TIcon.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1510 0- A   BEGIN
6 1511 --          gWResSignature := 'icon';   gWResType := 'TIcon';
6 1512 --          WRes(theResource, itsParams);
6 1513 -0 A   END;
6 1514 --      {$ENDC}
6 1515 --
6 1516 --
6 1517 --      {$S DlgClose}
6 1518 --      {-----------------------------------+
6 1519 --      |   Free                            |
6 1520 --      +-----------------------------------}
6 1521 -- A   PROCEDURE TIcon.Free; OVERRIDE;
6 1522 0- A   BEGIN
6 1523 --          ReleaseIcon;
6 1524 --          INHERITED Free;
6 1525 -0 A   END;
6 1526 --
6 1527 --      {$S DlgRes}
6 1528 --      {-----------------------------------+
6 1529 --      |   Draw                            |
6 1530 --      +-----------------------------------}
6 1531 -- A   PROCEDURE TIcon.Draw(area: Rect); OVERRIDE;
6 1532 --      VAR
6 1533 --          oldState:       SignedByte;
6 1534 --          theRect:        Rect;
6 1535 --
6 1536 0- A   BEGIN
6 1537 --          IF fDataHandle <> NIL THEN
6 1538 1-              BEGIN
6 1539 --              PenNormal; { NECESSARY? }
6 1540 --              ControlArea(theRect);
6 1541 --              IF fRsrcID <> kNoResource THEN
6 1542 --                  LoadResource(fDataHandle);
6 1543 --              IF fDataHandle = NIL THEN
6 1544 --                  ReleaseIcon
6 1545 --              ELSE IF NOT fPreferColor THEN
6 1546 2-                  BEGIN
6 1547 --                  oldState := GetHandleBits(fDataHandle);
6 1548 --                  HNoPurge(fDataHandle);
6 1549 --                  PlotIcon(theRect, fDataHandle);
6 1550 --                  SetHandleBits(fDataHandle, oldState);
6 1551 -2                  END
6 1552 --              ELSE
6 1553 --                  PlotCIcon(theRect, CIconHandle(fDataHandle));
6 1554 -1              END;
6 1555 --
6 1556 --          INHERITED Draw(area);
6 1557 -0 A   END;
6 1558 --
6 1559 --      {$S DlgNonRes}
6 1560 --      {-----------------------------------+
6 1561 --      |   ReleaseIcon                     |
6 1562 --      +-----------------------------------}
6 1563 -- A   PROCEDURE TIcon.ReleaseIcon;
6 1564 0- A   BEGIN
6 1565 --          fRsrcID := kNoResource;
6 1566 --          IF fPreferColor THEN
6 1567 --              DisposCIcon(CIconHandle(fDataHandle))
6 1568 --          ELSE
6 1569 --              HPurge(fDataHandle);
6 1570 --          fDataHandle := NIL;
6 1571 -0 A   END;
6 1572 --
6 1573 --      {$S DlgNonRes}
6 1574 --      {-----------------------------------+
6 1575 --      |   SetIcon                         |
6 1576 --      +-----------------------------------}
6 1577 -- A   PROCEDURE TIcon.SetIcon(theIcon: Handle; redraw: BOOLEAN);
6 1578 0- A   BEGIN
6 1579 --          ReleaseIcon;
6 1580 --          fDataHandle := theIcon;
6 1581 --          IF redraw THEN
6 1582 --              ForceRedraw;
6 1583 -0 A   END;
```

```
6 1584 --
6 1585 --      {$IFC qDebug}
6 1586 --          {$S DlgFields}
6 1587 --          {-----------------------------------+
6 1588 --          |   Fields                          |
6 1589 --          +-----------------------------------}
6 1590 -- A       PROCEDURE TIcon.Fields (PROCEDURE DoToField (fieldName: Str255;
6 1591 --                                                      fieldAddr: Ptr;
6 1592 --                                                      fieldType: INTEGER)); OVERRIDE;
6 1593 0- A       BEGIN
6 1594 --              DoToField('TIcon', NIL, bClass);
6 1595 --              DoToField('fPreferColor', @fPreferColor, bBoolean);
6 1596 --              DoToField('fRsrcID', @fRsrcID, bInteger);
6 1597 --              DoToField('fDataHandle', @fDataHandle, bHandle);
6 1598 --              INHERITED Fields(DoToField);
6 1599 -0 A       END;
6 1600 --
6 1601 --      {$ENDC}
6 1602 --
6 1603 --
6 1604 --      (*****************************************************************************************)
6 1605 --      (*      T P a t t e r n                                                                  *)
6 1606 --      (*****************************************************************************************)
6 1607 --
6 1608 --      {$IFC qProceduralViews}
6 1609 --      {$S DlgOpen}
6 1610 --      {-----------------------------------+
6 1611 --      |   IPattern                        |
6 1612 --      +-----------------------------------}
6 1613 -- A   PROCEDURE TPattern.IPattern (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 1614 --                                      itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 1615 --                                      itsRsrcID: INTEGER; preferColor: BOOLEAN);
6 1616 --      VAR
6 1617 --          fi:             FailInfo;
6 1618 --
6 1619 --          {------------------------------+
6 1620 --          |   HandleFailure              |
6 1621 --          +------------------------------}
6 1622 -- B       PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1623 0- B       BEGIN
6 1624 --              Free;
6 1625 -0 B       END;
6 1626 --
6 1627 0- A   BEGIN
6 1628 --          fDataHandle := NIL;
6 1629 --          IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 1630 --          fPreferColor := preferColor;
6 1631 --          fRsrcID := itsRsrcID;
6 1632 --          IF fRsrcID <> kNoResource THEN
6 1633 1-              BEGIN
6 1634 --              CatchFailures(fi, HandleFailure);
6 1635 --              IF fPreferColor AND gConfiguration.hasColorQD THEN
6 1636 --                  fDataHandle := Handle(GetPixPat(fRsrcID));
6 1637 --              IF fDataHandle = NIL THEN
6 1638 2-                  BEGIN
6 1639 --                  fPreferColor := NOT kPreferColor;   { Either can't or won't                  }
6 1640 --                  fDataHandle := Handle(GetPattern(fRsrcID));
6 1641 -2                  END;
6 1642 --              FailResError;
6 1643 --              Success(fi);
6 1644 -1              END;
6 1645 --          ViewEnable(FALSE, kDontRedraw);             { Default is to not enable hit testing          }
6 1646 --          fDefChoice := mPatternHit;
6 1647 -0 A   END;
6 1648 --      {$ENDC}
6 1649 --
6 1650 --
6 1651 --      {$IFC qTemplateViews}
6 1652 --      {$S DlgOpen}
6 1653 --      {-----------------------------------+
6 1654 --      |   IRes                            |
6 1655 --      +-----------------------------------}
6 1656 -- A   PROCEDURE TPattern.IRes (itsDocument: TDocument; itsSuperView: TView;
6 1657 --                                      VAR itsParams: Ptr); OVERRIDE;
6 1658 --      VAR
6 1659 --          fi:             FailInfo;
6 1660 --
6 1661 --          {------------------------------+
6 1662 --          |   HandleFailure              |
6 1663 --          +------------------------------}
6 1664 -- B       PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1665 0- B       BEGIN
6 1666 --              Free;
6 1667 -0 B       END;
6 1668 --
6 1669 0- A   BEGIN
6 1670 --          fDataHandle := NIL;
6 1671 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6 1672 --
6 1673 --          WITH PatternTemplatePtr(itsParams)^ DO
6 1674 1-              BEGIN
6 1675 --              fPreferColor := preferColor;
6 1676 --              fRsrcID := rsrcID;
6 1677 -1              END;
6 1678 --          IF fRsrcID <> kNoResource THEN
6 1679 1-              BEGIN
6 1680 --              CatchFailures(fi, HandleFailure);
6 1681 --              IF fPreferColor AND gConfiguration.hasColorQD THEN
6 1682 --                  fDataHandle := Handle(GetPixPat(fRsrcID));
```

```
6 1683 --             IF fDataHandle = NIL THEN
6 1684 2-                 BEGIN
6 1685 --                 fPreferColor := NOT kPreferColor;   { Either can't or won't                    }
6 1686 --                 fDataHandle := Handle(GetPattern(fRsrcID));
6 1687 -2                 END;
6 1688 --             FailResError;
6 1689 --             Success(fi);
6 1690 -1             END;
6 1691 --         fDefChoice := mPatternHit;
6 1692 --
6 1693 --         OffsetPtr(itsParams, SIZEOF(PatternTemplate));
6 1694 -0 A   END;
6 1695 --     {$ENDC}
6 1696 --
6 1697 --
6 1698 --     {$IFC qWriteTemplates}
6 1699 --     {$S MAWriteRes}
6 1700 --     {-----------------------------------+
6 1701 --     |   WRes                            |
6 1702 --     +-----------------------------------}
6 1703 -- A   PROCEDURE TPattern.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1704 --
6 1705 --     VAR
6 1706 --         ptPtr:      PatternTemplatePtr;
6 1707 --
6 1708 0- A   BEGIN
6 1709 --         INHERITED WRes(theResource, itsParams);
6 1710 --
6 1711 --         ptPtr := PatternTemplatePtr(ExpandPtr(theResource, itsParams, SIZEOF(PatternTemplate)));
6 1712 --
6 1713 --         WITH ptPtr^ DO
6 1714 1-             BEGIN
6 1715 --             preferColor := fPreferColor;
6 1716 --     {$IFC qDebug}
6 1717 --             IF fRsrcID = kNoResource THEN
6 1718 --                 WriteLn('Tried to write TPattern with no resource ID.');
6 1719 --     {$ENDC}
6 1720 --             rsrcID := fRsrcID;
6 1721 -1             END;
6 1722 -0 A   END;
6 1723 --
6 1724 --     {$S MAWriteRes}
6 1725 --     {-----------------------------------+
6 1726 --     |   WriteRes                        |
6 1727 --     +-----------------------------------}
6 1728 -- A   PROCEDURE TPattern.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1729 0- A   BEGIN
6 1730 --         gWResSignature := 'patn';   gWResType := 'TPattern';
6 1731 --         WRes(theResource, itsParams);
6 1732 -0 A   END;
6 1733 --     {$ENDC}
6 1734 --
6 1735 --
6 1736 --     {$S DlgClose}
6 1737 --     {-----------------------------------+
6 1738 --     |   Free                            |
6 1739 --     +-----------------------------------}
6 1740 -- A   PROCEDURE TPattern.Free; OVERRIDE;
6 1741 0- A   BEGIN
6 1742 --         ReleasePattern;
6 1743 --         INHERITED Free;
6 1744 -0 A   END;
6 1745 --
6 1746 --     {$S DlgRes}
6 1747 --     {-----------------------------------+
6 1748 --     |   Draw                            |
6 1749 --     +-----------------------------------}
6 1750 -- A   PROCEDURE TPattern.Draw (area: Rect); OVERRIDE;
6 1751 --     VAR
6 1752 --         oldState:       SignedByte;
6 1753 --         theRect:        Rect;
6 1754 --
6 1755 0- A   BEGIN
6 1756 --         IF fDataHandle <> NIL THEN
6 1757 1-             BEGIN
6 1758 --             PenNormal; { NECESSARY? }
6 1759 --             ControlArea(theRect);
6 1760 --             IF (fRsrcID <> kNoResource) &
6 1761 --               NOT fPreferColor THEN             { Pixpat handles <> resource handles!!!!        }
6 1762 --                 LoadResource(fDataHandle);
6 1763 --             IF fDataHandle = NIL THEN
6 1764 --                 ReleasePattern
6 1765 --             ELSE IF NOT fPreferColor THEN
6 1766 2-                 BEGIN
6 1767 --                 oldState := GetHandleBits(fDataHandle);
6 1768 --                 HNoPurge(fDataHandle);
6 1769 --                 FillRect(theRect, PatHandle(fDataHandle)^^);
6 1770 --                 SetHandleBits(fDataHandle, oldState);
6 1771 -2                 END
6 1772 --             ELSE
6 1773 --                 FillCRect(theRect, PixPatHandle(fDataHandle));
6 1774 -1             END;
6 1775 --
6 1776 --         INHERITED Draw(area);
6 1777 -0 A   END;
6 1778 --
6 1779 --     {$S DlgNonRes}
6 1780 --     {-----------------------------------+
6 1781 --     |   ReleasePattern                  |
```

```
6 1782 --      +-----------------------------------}
6 1783 -- A    PROCEDURE TPattern.ReleasePattern;
6 1784 0- A    BEGIN
6 1785 --          fRsrcID := kNoResource;
6 1786 --          IF fPreferColor THEN
6 1787 --              DisposPixPat(PixPatHandle(fDataHandle))
6 1788 --          ELSE
6 1789 --              HPurge(fDataHandle);
6 1790 --          fDataHandle := NIL;
6 1791 -0 A    END;
6 1792 --
6 1793 --      {$S DlgNonRes}
6 1794 --      {-----------------------------------+
6 1795 --      |   SetPattern                      |
6 1796 --      +-----------------------------------}
6 1797 -- A    PROCEDURE TPattern.SetPattern (thePattern: Handle; redraw: BOOLEAN);
6 1798 0- A    BEGIN
6 1799 --          ReleasePattern;
6 1800 --          fDataHandle := thePattern;
6 1801 --          IF redraw THEN
6 1802 --              ForceRedraw;
6 1803 -0 A    END;
6 1804 --
6 1805 --      {$IFC qDebug}
6 1806 --          {$S DlgFields}
6 1807 --          {-----------------------------------+
6 1808 --          |   Fields                          |
6 1809 --          +-----------------------------------}
6 1810 -- A        PROCEDURE TPattern.Fields (PROCEDURE DoToField (fieldName: Str255;
6 1811 --                                                          fieldAddr: Ptr;
6 1812 --                                                          fieldType: INTEGER)); OVERRIDE;
6 1813 0- A        BEGIN
6 1814 --              DoToField('TPattern', NIL, bClass);
6 1815 --              DoToField('fPreferColor', @fPreferColor, bBoolean);
6 1816 --              DoToField('fRsrcID', @fRsrcID, bInteger);
6 1817 --              DoToField('fDataHandle', @fDataHandle, bHandle);
6 1818 --              INHERITED Fields(DoToField);
6 1819 -0 A        END;
6 1820 --
6 1821 --      {$ENDC}
6 1822 --
6 1823 --
6 1824 --      {*****************************************************************************************}
6 1825 --      {*      T P i c t u r e                                                                  *}
6 1826 --      {*****************************************************************************************}
6 1827 --
6 1828 --      {$IFC qProceduralViews}
6 1829 --      {$S DlgOpen}
6 1830 --      {-----------------------------------+
6 1831 --      |   IPicture                        |
6 1832 --      +-----------------------------------}
6 1833 -- A    PROCEDURE TPicture.IPicture (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 1834 --                                  itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 1835 --                                  itsRsrcID: INTEGER);
6 1836 --
6 1837 --      VAR
6 1838 --          fi:                 FailInfo;
6 1839 --
6 1840 --          {-------------------------------+
6 1841 --          |   HandleFailure               |
6 1842 --          +-------------------------------}
6 1843 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1844 0- B        BEGIN
6 1845 --              Free;
6 1846 -0 B        END;
6 1847 --
6 1848 0- A    BEGIN
6 1849 --          fDataHandle := NIL;
6 1850 --          IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 1851 --          fRsrcID := itsRsrcID;
6 1852 --          IF fRsrcID <> kNoResource THEN
6 1853 1-              BEGIN
6 1854 --              CatchFailures(fi, HandleFailure);
6 1855 --              fDataHandle := GetPicture(fRsrcID);
6 1856 --              FailResError;
6 1857 --              Success(fi);
6 1858 -1              END;
6 1859 --          ViewEnable(FALSE, kDontRedraw);                 { Default is to not enable hit testing          }
6 1860 --          fDefChoice := mPictureHit;
6 1861 -0 A    END;
6 1862 --      {$ENDC}
6 1863 --
6 1864 --
6 1865 --      {$IFC qTemplateViews}
6 1866 --      {$S DlgOpen}
6 1867 --      {-----------------------------------+
6 1868 --      |   IRes                            |
6 1869 --      +-----------------------------------}
6 1870 -- A    PROCEDURE TPicture.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 1871 --
6 1872 --      VAR
6 1873 --          fi:                 FailInfo;
6 1874 --
6 1875 --          {-------------------------------+
6 1876 --          |   HandleFailure               |
6 1877 --          +-------------------------------}
6 1878 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 1879 0- B        BEGIN
6 1880 --              Free;
```

```
6 1881 -0 B       END;
6 1882 --
6 1883 0- A   BEGIN
6 1884 --         fDataHandle := NIL;
6 1885 --         INHERITED IRes(NIL, itsSuperView, itsParams);
6 1886 --
6 1887 --         fRsrcID := PictureTemplatePtr(itsParams)^.rsrcID;
6 1888 --         IF fRsrcID <> kNoResource THEN
6 1889 1-             BEGIN
6 1890 --             CatchFailures(fi, HandleFailure);
6 1891 --             fDataHandle := GetPicture(fRsrcID);
6 1892 --             FailResError;
6 1893 --             Success(fi);
6 1894 -1              END;
6 1895 --         fDefChoice := mPictureHit;
6 1896 --
6 1897 --         OffsetPtr(itsParams, SIZEOF(PictureTemplate));
6 1898 -0 A   END;
6 1899 --     {$ENDC}
6 1900 --
6 1901 --
6 1902 --     {$IFC qWriteTemplates}
6 1903 --     {$S MAWriteRes}
6 1904 --     {-----------------------------------+
6 1905 --     |   WRes                            |
6 1906 --     +-----------------------------------}
6 1907 -- A   PROCEDURE TPicture.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1908 --
6 1909 --     VAR
6 1910 --         pcPtr:      PictureTemplatePtr;
6 1911 --
6 1912 0- A   BEGIN
6 1913 --         INHERITED WRes(theResource, itsParams);
6 1914 --
6 1915 --         pcPtr := PictureTemplatePtr(ExpandPtr(theResource, itsParams, SIZEOF(PictureTemplate)));
6 1916 --
6 1917 --     {$IFC qDebug}
6 1918 --         IF fRsrcID = kNoResource THEN
6 1919 --             WriteLn('Tried to write TPicture with no resource ID.');
6 1920 --     {$ENDC}
6 1921 --         pcPtr^.rsrcID := fRsrcID;
6 1922 -0 A   END;
6 1923 --
6 1924 --     {$S MAWriteRes}
6 1925 --     {-----------------------------------+
6 1926 --     |   WriteRes                        |
6 1927 --     +-----------------------------------}
6 1928 -- A   PROCEDURE TPicture.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 1929 0- A   BEGIN
6 1930 --         gWResSignature := 'pict';   gWResType := 'TPicture';
6 1931 --         WRes(theResource, itsParams);
6 1932 -0 A   END;
6 1933 --     {$ENDC}
6 1934 --
6 1935 --
6 1936 --     {$S DlgClose}
6 1937 --     {-----------------------------------+
6 1938 --     |   Free                            |
6 1939 --     +-----------------------------------}
6 1940 -- A   PROCEDURE TPicture.Free; OVERRIDE;
6 1941 0- A   BEGIN
6 1942 --         ReleasePicture;
6 1943 --         INHERITED Free;
6 1944 -0 A   END;
6 1945 --
6 1946 --     {$S DlgRes}
6 1947 --     {-----------------------------------+
6 1948 --     |   Draw                            |
6 1949 --     +-----------------------------------}
6 1950 -- A   PROCEDURE TPicture.Draw (area: Rect); OVERRIDE;
6 1951 --     VAR
6 1952 --         oldState:       SignedByte;
6 1953 --         theRect:        Rect;
6 1954 --
6 1955 0- A   BEGIN
6 1956 --         IF fDataHandle <> NIL THEN
6 1957 1-             BEGIN
6 1958 --             ControlArea(theRect);
6 1959 --             IF fRsrcID <> kNoResource THEN
6 1960 --                 LoadResource(Handle(fDataHandle));
6 1961 --             IF fDataHandle = NIL THEN
6 1962 --                 ReleasePicture
6 1963 --             ELSE
6 1964 2-                 BEGIN
6 1965 --                 oldState := GetHandleBits(Handle(fDataHandle));
6 1966 --                 HNoPurge(Handle(fDataHandle));
6 1967 --                 PenNormal; { ??? NECESSARY ??? }
6 1968 --                 DrawPicture(fDataHandle, theRect);
6 1969 --                 SetHandleBits(Handle(fDataHandle), oldState);
6 1970 -2                  END;
6 1971 -1              END;
6 1972 --         INHERITED Draw(area);
6 1973 -0 A   END;
6 1974 --
6 1975 --     {$S DlgNonRes}
6 1976 --     {-----------------------------------+
6 1977 --     |   ReleasePicture                  |
6 1978 --     +-----------------------------------}
6 1979 -- A   PROCEDURE TPicture.ReleasePicture;
```

```
6 1980 0- A   BEGIN
6 1981 --         fRsrcID := kNoResource;
6 1982 --         fDataHandle := NIL;
6 1983 -0 A   END;
6 1984 --
6 1985 --     {$S DlgNonRes}
6 1986 --     {-----------------------------------+
6 1987 --     |   SetPicture                      |
6 1988 --     +-----------------------------------}
6 1989 -- A   PROCEDURE TPicture.SetPicture (thePicture: PicHandle; redraw: BOOLEAN);
6 1990 0- A   BEGIN
6 1991 --         ReleasePicture;
6 1992 --         fDataHandle := thePicture;
6 1993 --         IF redraw THEN
6 1994 --             ForceRedraw;
6 1995 -0 A   END;
6 1996 --
6 1997 --     {$IFC qDebug}
6 1998 --         {$S DlgFields}
6 1999 --         {-----------------------------------+
6 2000 --         |   Fields                          |
6 2001 --         +-----------------------------------}
6 2002 -- A       PROCEDURE TPicture.Fields (PROCEDURE DoToField (fieldName: Str255;
6 2003 --                                                        fieldAddr: Ptr;
6 2004 --                                                        fieldType: INTEGER)); OVERRIDE;
6 2005 0- A       BEGIN
6 2006 --             DoToField('TPicture', NIL, bClass);
6 2007 --             DoToField('fRsrcID', @fRsrcID, bInteger);
6 2008 --             DoToField('fDataHandle', @fDataHandle, bHandle);
6 2009 --             INHERITED Fields(DoToField);
6 2010 -0 A       END;
6 2011 --
6 2012 --     {$ENDC}
6 2013 --
6 2014 --
6 2015 --     (*****************************************************************************************)
6 2016 --     (*      T P o p u p                                                                      *)
6 2017 --     (*****************************************************************************************)
6 2018 --
6 2019 --     {$IFC qProceduralViews}
6 2020 --     {$S DlgOpen}
6 2021 --     {-----------------------------------+
6 2022 --     |   IPopup                          |
6 2023 --     +-----------------------------------}
6 2024 -- A   PROCEDURE TPopup.IPopup (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 2025 --                             itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 2026 --                             itsRsrcID, itsCurrentItem, itsItemOffset: INTEGER);
6 2027 --
6 2028 --     VAR
6 2029 --         fi:                 FailInfo;
6 2030 --         aMenu:              MenuHandle;
6 2031 --
6 2032 --         {-------------------------------+
6 2033 --         |   HandleFailure               |
6 2034 --         +-------------------------------}
6 2035 -- B       PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 2036 0- B       BEGIN
6 2037 --             Free;
6 2038 -0 B       END;
6 2039 --
6 2040 0- A   BEGIN
6 2041 --         fMenuHandle := NIL;
6 2042 --         IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 2043 --
6 2044 --         IF qConfiguration.hasHierarchicalMenus THEN
6 2045 1-             BEGIN
6 2046 --             fCurrentItem := Max(1, itsCurrentItem);
6 2047 --             fItemOffset := itsItemOffset;
6 2048 --             IF itsRsrcID <> kNoResource THEN
6 2049 2-                 BEGIN
6 2050 --                 CatchFailures(fi, HandleFailure);
6 2051 --                 aMenu := GetMenu(itsRsrcID);
6 2052 --                 FailNIL(aMenu);
6 2053 --                 HNoPurge(Handle(aMenu));
6 2054 --                 SetPopup(aMenu, itsRsrcID, itsCurrentItem, FALSE);
6 2055 --                 Success(fi);
6 2056 -2                 END
6 2057 --             ELSE
6 2058 2-                 BEGIN
6 2059 --                 fRsrcID := kNoResource;
6 2060 --                 fMenuID := kNoResource;
6 2061 -2                 END;
6 2062 --             fDefChoice := mPopupHit;
6 2063 -1             END
6 2064 --         ELSE
6 2065 1-             BEGIN
6 2066 --     {$IFC qDebug}
6 2067 --             ProgramBreak('Attempt to use popup menus on machine that doesn''t support them');
6 2068 --     {$ENDC}
6 2069 --             fShown := FALSE;                        { What's reasonable here ??? }
6 2070 -1             END;
6 2071 -0 A   END;
6 2072 --     {$ENDC}
6 2073 --
6 2074 --
6 2075 --     {$IFC qTemplateViews}
6 2076 --     {$S DlgOpen}
6 2077 --     {-----------------------------------+
6 2078 --     |   IRes                            |
```

```
6 2079 --      +-----------------------------------}
6 2080 -- A    PROCEDURE TPopup.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 2081 --
6 2082 --      VAR
6 2083 --          fi:                 FailInfo;
6 2084 --          aMenu:              MenuHandle;
6 2085 --
6 2086 --          {-------------------------------+
6 2087 --          |   HandleFailure               |
6 2088 --          +-------------------------------}
6 2089 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 2090 0- B        BEGIN
6 2091 --              Free;
6 2092 -0 B        END;
6 2093 --
6 2094 0- A    BEGIN
6 2095 --          fMenuHandle := NIL;
6 2096 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6 2097 --
6 2098 --          IF gConfiguration.hasHierarchicalMenus THEN
6 2099 1-              BEGIN
6 2100 --              WITH PopupTemplatePtr(itsParams)^ DO
6 2101 2-                  BEGIN
6 2102 --                  fCurrentItem := Max(1, currentItem);
6 2103 --                  fItemOffset := itemOffset;
6 2104 --                  IF rsrcID <> kNoResource THEN
6 2105 3-                      BEGIN
6 2106 --                      CatchFailures(fi, HandleFailure);
6 2107 --                      aMenu := GetMenu(rsrcID);
6 2108 --                      FailNIL(aMenu);
6 2109 --                      HNoPurge(Handle(aMenu));
6 2110 --                      SetPopup(aMenu, rsrcID, fCurrentItem, FALSE);
6 2111 --                      Success(fi);
6 2112 -3                      END
6 2113 --                  ELSE
6 2114 3-                      BEGIN
6 2115 --                      fRsrcID := kNoResource;
6 2116 --                      fMenuID := kNoResource;
6 2117 -3                      END;
6 2118 -2                  END;
6 2119 --              fDefChoice := mPopupHit;
6 2120 -1              END
6 2121 --          ELSE
6 2122 1-              BEGIN
6 2123 --      {$IFC qDebug}
6 2124 --              ProgramBreak('Attempt to use popup menus on machine that doesn''t support them');
6 2125 --      {$ENDC}
6 2126 --              fShown := FALSE;                        { What's reasonable here ??? }
6 2127 -1              END;
6 2128 --
6 2129 --          OffsetPtr(itsParams, SIZEOF(PopupTemplate));
6 2130 -0 A    END;
6 2131 --      {$ENDC}
6 2132 --
6 2133 --
6 2134 --      {$IFC qWriteTemplates}
6 2135 --      {$S MAWriteRes}
6 2136 --      {-----------------------------------+
6 2137 --      |   WRes                            |
6 2138 --      +-----------------------------------}
6 2139 -- A    PROCEDURE TPopup.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2140 --
6 2141 --      VAR
6 2142 --          poPtr:      PopupTemplatePtr;
6 2143 --
6 2144 0- A    BEGIN
6 2145 --          INHERITED WRes(theResource, itsParams);
6 2146 --
6 2147 --          poPtr := PopupTemplatePtr(ExpandPtr(theResource, itsParams, SIZEOF(PopupTemplate)));
6 2148 --
6 2149 --          WITH poPtr^ DO
6 2150 1-              BEGIN
6 2151 --      {$IFC qDebug}
6 2152 --              IF fRsrcID = kNoResource THEN
6 2153 --                  ProgramBreak('Tried to write TPopup with no resource ID.');
6 2154 --      {$ENDC}
6 2155 --              rsrcID := fRsrcID;
6 2156 --              currentItem := fCurrentItem;
6 2157 --              itemOffset := fItemOffset;
6 2158 -1              END;
6 2159 -0 A    END;
6 2160 --
6 2161 --      {$S MAWriteRes}
6 2162 --      {-----------------------------------+
6 2163 --      |   WriteRes                        |
6 2164 --      +-----------------------------------}
6 2165 -- A    PROCEDURE TPopup.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2166 0- A    BEGIN
6 2167 --          gWResSignature := 'popp';   gWResType := 'TPopup';
6 2168 --          WRes(theResource, itsParams);
6 2169 -0 A    END;
6 2170 --      {$ENDC}
6 2171 --
6 2172 --
6 2173 --      {$S DlgClose}
6 2174 --      {-----------------------------------+
6 2175 --      |   Free                            |
6 2176 --      +-----------------------------------}
6 2177 -- A    PROCEDURE TPopup.Free; OVERRIDE;
```

```
6 2178 0- A   BEGIN
6 2179 --         ReleasePopup;
6 2180 --         INHERITED Free;
6 2181 -0 A   END;
6 2182 --
6 2183 --     {$S DlgOpen}
6 2184 --     {-----------------------------------+
6 2185 --     |   AdjustBotRight                  |
6 2186 --     +-----------------------------------}
6 2187 -- A   PROCEDURE TPopup.AdjustBotRight;
6 2188 --     VAR
6 2189 --         numItems:       INTEGER;
6 2190 --         newHeight:      INTEGER;
6 2191 --         newWidth:       INTEGER;
6 2192 0- A   BEGIN
6 2193 --         IF fMenuHandle <> NIL THEN
6 2194 1-             BEGIN
6 2195 --             CalcMenuSize(fMenuHandle);
6 2196 --             numItems := CountMItems(fMenuHandle);
6 2197 --             newWidth := fMenuHandle^^.menuWidth + fItemOffset + fInset.left + fInset.right + 3;
6 2198 --             newHeight := fMenuHandle^^.menuHeight DIV MAX(1, numItems) + fInset.top + fInset.bottom + 3;
6 2199 --             Resize(newWidth, newHeight, kRedraw);
6 2200 -1             END;
6 2201 -0 A   END;
6 2202 --
6 2203 --     {$S DlgRes}
6 2204 --     {-----------------------------------+
6 2205 --     |   CalcLabelRect                   |
6 2206 --     +-----------------------------------}
6 2207 -- A   PROCEDURE TPopup.CalcLabelRect (VAR theRect: Rect);
6 2208 --
6 2209 0- A   BEGIN
6 2210 --         ControlArea(theRect);
6 2211 --         InsetRect(theRect, 1, 1);
6 2212 --         WITH theRect DO
6 2213 1-             BEGIN
6 2214 --             right := left + fItemOffset;
6 2215 --             bottom := bottom - 1;
6 2216 -1             END;
6 2217 -0 A   END;
6 2218 --
6 2219 --     {$S DlgRes}
6 2220 --     {-----------------------------------+
6 2221 --     |   CalcMenuRect                    |
6 2222 --     +-----------------------------------}
6 2223 -- A   PROCEDURE TPopup.CalcMenuRect (VAR theRect: Rect);
6 2224 --
6 2225 0- A   BEGIN
6 2226 --         ControlArea(theRect);
6 2227 --         InsetRect(theRect, 1, 1);
6 2228 --         WITH theRect DO
6 2229 1-             BEGIN
6 2230 --             left := left + fItemOffset;
6 2231 --             botRight := Point(LONGINT(botRight) - $00010001);
6 2232 -1             END;
6 2233 -0 A   END;
6 2234 --
6 2235 --     {$S DlgRes}
6 2236 --     {-----------------------------------+
6 2237 --     |   DoMouseCommand                  |
6 2238 --     +-----------------------------------}
6 2239 -- A   FUNCTION  TPopup.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
6 2240 --                                       VAR hysteresis: Point): TCommand; OVERRIDE;
6 2241 --     VAR
6 2242 --         newChoice:      INTEGER;
6 2243 --         result:         LONGINT;
6 2244 --         menuPt:         Point;
6 2245 --         aMenuHandle:    MenuHandle;
6 2246 --         labelRect:      Rect;
6 2247 --         menuRect:       Rect;
6 2248 --         oldFColor:      RGBColor;
6 2249 --         oldBkColor:     RGBColor;
6 2250 --         newFColor:      RGBColor;
6 2251 --         newBkColor:     RGBColor;
6 2252 --
6 2253 0- A   BEGIN
6 2254 --         DoMouseCommand := gNoChanges;
6 2255 --         CalcLabelRect(labelRect);
6 2256 --         CalcMenuRect(menuRect);
6 2257 --
6 2258 --         IF fMenuHandle <> NIL THEN
6 2259 1-             BEGIN
6 2260 --             MAInsertMenu(fMenuHandle, hierMenu);    { Use MAInsertMenu to ensure colors are set }
6 2261 --             { Save the old colors, fetch the item colors, and draw the popup box }
6 2262 --             GetIfColor(oldFColor); GetIfBkColor(oldBkColor);
6 2263 --             GetMenuColors(menuRect, fMenuID, 0, newFColor, newBkColor);
6 2264 --             SetIfColor(newBkColor); SetIfBkColor(newFColor);
6 2265 --             DrawLabel(labelRect);
6 2266 --
6 2267 --             IF gConfiguration.hasColorQD & (fRsrcID <> kNoResource) THEN
6 2268 --                 aMenuHandle := GetResMenu(fRsrcID); { Reloads color tables!                    }
6 2269 --             SetPt(menuPt, labelRect.right, labelRect.top);
6 2270 --             LocalToGlobal(menuPt);
6 2271 --             CalcMenuSize(fMenuHandle);          { Fix for Menu Manager bug                 }
6 2272 --             result := PopUpMenuSelect(fMenuHandle, menuPt.v, menuPt.h, fCurrentItem);
6 2273 --             newChoice := LoWord(result);
6 2274 --             SetIfColor(newFColor); SetIfBkColor(newBkColor);
6 2275 --             DrawLabel(labelRect);
6 2276 --             SetIfColor(oldFColor); SetIfBkColor(oldBkColor);
```

```
6 2277 --              IF HiWord(result) <> 0 THEN
6 2278 2-                  BEGIN
6 2279 --                  SetCurrentItem(newChoice, TRUE);
6 2280 --                  DoChoice(SELF, fDefChoice);
6 2281 -2                  END;
6 2282 --              DeleteMenu(fMenuId);
6 2283 --              SetIfColor(oldFColor); SetIfBkColor(oldBkColor);
6 2284 -1              END;
6 2285 -0 A    END;
6 2286 --
6 2287 --      {$S DlgRes}
6 2288 --      {-----------------------------------+
6 2289 --      |   Draw                            |
6 2290 --      +-----------------------------------}
6 2291 -- A    PROCEDURE TPopup.Draw (area: Rect); OVERRIDE;
6 2292 --
6 2293 --      VAR
6 2294 --          aRect:          Rect;
6 2295 --          oldFColor:      RGBColor;
6 2296 --          oldBkColor:     RGBColor;
6 2297 --          newFColor:      RGBColor;
6 2298 --          newBkColor:     RGBColor;
6 2299 --
6 2300 0- A    BEGIN
6 2301 --          IF fMenuHandle <> NIL THEN
6 2302 1-              BEGIN
6 2303 --              MAInsertMenu(fMenuHandle, hierMenu);    { Use MAInsertMenu to ensure colors are set }
6 2304 --              { Erase the whole menu first }
6 2305 --              ControlArea(aRect);
6 2306 --              IF SectRect(area, aRect, aRect) THEN
6 2307 2-                  BEGIN
6 2308 --                  (* EraseRect(aRect); *)
6 2309 --
6 2310 --                  { Save the old colors, fetch the item colors, and draw the popup box }
6 2311 --                  GetIfColor(oldFColor); GetIfBkColor(oldBkColor);
6 2312 --                  CalcMenuRect(aRect);
6 2313 --                  GetMenuColors(aRect, fMenuID, fCurrentItem, newFColor, newBkColor);
6 2314 --                  SetIfColor(newFColor); SetIfBkColor(newBkColor);
6 2315 --                  DrawPopupBox(area);
6 2316 --
6 2317 --                  { Fetch the title colors, and draw it }
6 2318 --                  GetMenuColors(aRect, fMenuID, 0, newFColor, newBkColor);
6 2319 --                  SetIfColor(newFColor); SetIfBkColor(newBkColor);
6 2320 --                  DrawLabel(area);
6 2321 --
6 2322 --                  { Reset colors to their original state }
6 2323 --                  SetIfColor(oldFColor); SetIfBkColor(oldBkColor);
6 2324 -2                  END;
6 2325 --              DeleteMenu(fMenuID);
6 2326 -1              END;
6 2327 --
6 2328 --          INHERITED Draw(area);
6 2329 -0 A    END;
6 2330 --
6 2331 --      {$S DlgRes}
6 2332 --      {-----------------------------------+
6 2333 --      |   DrawLabel                       |
6 2334 --      +-----------------------------------}
6 2335 -- A    PROCEDURE TPopup.DrawLabel (area: Rect);
6 2336 --
6 2337 --      CONST
6 2338 --          labelSlop       =    5;
6 2339 --
6 2340 --      VAR
6 2341 --          labelRect:      Rect;
6 2342 --          theLabel:       Str255;
6 2343 --
6 2344 0- A    BEGIN
6 2345 --          CalcLabelRect(labelRect);
6 2346 --          labelRect.right := labelRect.right - labelSlop;
6 2347 --          IF SectRect(area, labelRect, area) THEN
6 2348 1-              BEGIN
6 2349 --              theLabel := CONCAT(fMenuHandle^^.menuData, ':'); { Fetch the title of the menu      }
6 2350 --              TextBox(Ptr(ORD4(@theLabel)+1), LENGTH(theLabel), labelRect, teJustRight);
6 2351 -1              END;
6 2352 -0 A    END;
6 2353 --
6 2354 --      {$S DlgRes}
6 2355 --      {-----------------------------------+
6 2356 --      |   DrawPopupBox                    |
6 2357 --      +-----------------------------------}
6 2358 -- A    PROCEDURE TPopup.DrawPopupBox (area: Rect);
6 2359 --
6 2360 --      CONST
6 2361 --          ShadowedFrame   = [adnLineTop, adnLineLeft, adnLineBottom, adnLineRight, adnShadow];
6 2362 --          leftSlop        =   12;
6 2363 --          rightSlop       =    0;
6 2364 --          botSlop         =    4;
6 2365 --
6 2366 --      VAR
6 2367 --          wid:            INTEGER;
6 2368 --          newWid:         INTEGER;
6 2369 --          newLen:         INTEGER;
6 2370 --          menuRect:       Rect;
6 2371 --          colorRect:      Rect;
6 2372 --          theLabel:       Str255;
6 2373 --
6 2374 0- A    BEGIN
6 2375 --          CalcMenuRect(menuRect);
```

```
6 2376 --          GetItem(fMenuHandle, fCurrentItem, theLabel);
6 2377 --          WITH menuRect DO
6 2378 1-              BEGIN
6 2379 --              IF SectRect(area, menuRect, colorRect) THEN
6 2380 2-                  BEGIN
6 2381 --                  wid := (right - left) - (leftSlop + rightSlop);
6 2382 --                  newWid := StringWidth(theLabel);
6 2383 --                  IF newWid > wid THEN
6 2384 3-                      BEGIN
6 2385 --                      newLen := LENGTH(theLabel);
6 2386 --                      wid := wid - CharWidth('…');
6 2387 --
6 2388 4-                      REPEAT
6 2389 --                          newWid := newWid - CharWidth(theLabel[newLen]);
6 2390 --                          newLen := PRED(newLen);
6 2391 -4                      UNTIL (newWid <= wid) OR (LENGTH(theLabel) = 0);
6 2392 --
6 2393 --                      newLen := SUCC(newLen);
6 2394 --                      theLabel[newLen] := '…';
6 2395 --                      theLabel[0] := CHR(newLen);
6 2396 -3                      END;
6 2397 --
6 2398 --                  PenNormal;
6 2399 --
6 2400 --                  EraseRect(colorRect);
6 2401 --
6 2402 --                  MoveTo(left + leftSlop, bottom - botSlop);
6 2403 --                  DrawString(theLabel);
6 2404 --                  SetIfColor(gRGBBlack);
6 2405 --                  InsetRect(menuRect, -1, -1);
6 2406 --                  botRight := Point(LONGINT(botRight) - $00010001);
6 2407 --                  FrameRect(menuRect);
6 2408 --                  MoveTo(left + 2, bottom);
6 2409 --                  LineTo(right, bottom);
6 2410 --                  LineTo(right, top + 2);
6 2411 -2                  END;
6 2412 -1              END;
6 2413 -0 A    END;
6 2414 --
6 2415 --      {$S DlgNonRes}
6 2416 --      {-----------------------------------+
6 2417 --      |   GetCurrentItem                  |
6 2418 --      +-----------------------------------}
6 2419 -- A    FUNCTION  TPopup.GetCurrentItem: INTEGER;
6 2420 0- A    BEGIN
6 2421 --          GetCurrentItem := fCurrentItem;
6 2422 -0 A    END;
6 2423 --
6 2424 --      {$S DlgNonRes}
6 2425 --      {-----------------------------------+
6 2426 --      |   GetItemText                     |
6 2427 --      +-----------------------------------}
6 2428 -- A    PROCEDURE TPopup.GetItemText (item: INTEGER; VAR theText: Str255);
6 2429 0- A    BEGIN
6 2430 --          IF fMenuHandle <> NIL THEN
6 2431 --              GetItem(fMenuHandle, item, theText)
6 2432 --          ELSE
6 2433 --              theText := '';
6 2434 -0 A    END;
6 2435 --
6 2436 --      {$S DlgNonRes}
6 2437 --      {-----------------------------------+
6 2438 --      |   ReleasePopup                    |
6 2439 --      +-----------------------------------}
6 2440 -- A    PROCEDURE TPopup.ReleasePopup;
6 2441 0- A    BEGIN
6 2442 --          IF fMenuHandle <> NIL THEN
6 2443 1-              BEGIN
6 2444 --              {$IFC FALSE}
6 2445 --              DeleteMenu(fMenuID);
6 2446 --              {$ENDC}
6 2447 --              HPurge(Handle(fMenuHandle));
6 2448 --              DisposHandle(Handle(fMenuHandle));
6 2449 --              fMenuHandle := NIL;
6 2450 -1              END;
6 2451 --          fRsrcID := kNoResource;
6 2452 --          fMenuID := kNoResource;
6 2453 --          fCurrentItem := 0;
6 2454 -0 A    END;
6 2455 --
6 2456 --      {$S DlgNonRes}
6 2457 --      {-----------------------------------+
6 2458 --      |   SetCurrentItem                  |
6 2459 --      +-----------------------------------}
6 2460 -- A    PROCEDURE TPopup.SetCurrentItem (item: INTEGER; redraw: BOOLEAN);
6 2461 --      VAR
6 2462 --          menuRect:       Rect;
6 2463 0- A    BEGIN
6 2464 --          IF (fMenuHandle <> NIL) AND (item <> fCurrentItem) THEN
6 2465 1-              BEGIN
6 2466 --              IF fCurrentItem <> 0 THEN
6 2467 --                  SetItemMark(fMenuHandle, fCurrentItem, ' ');
6 2468 --              IF item <> 0 THEN
6 2469 --                  SetItemMark(fMenuHandle, item, CHR(checkMark));
6 2470 --              fCurrentItem := item;
6 2471 -1              END;
6 2472 --          IF redraw THEN
6 2473 1-              BEGIN
6 2474 --              CalcMenuRect(menuRect);
```

```
6 2475 --             menuRect.botRight := Point(LONGINT(menuRect.botRight) - $00010001);
6 2476 --             InvalidRect(menuRect);
6 2477 -1             END;
6 2478 -0 A    END;
6 2479 --
6 2480 --      {$S DlgRes}
6 2481 --      {-----------------------------------+   theMenu is assumed to be a handle to a real MENU.  If it
6 2482 --      |   SetPopup                        |       is from a resource, the handle will be detached!
6 2483 --      +-----------------------------------}
6 2484 -- A    PROCEDURE TPopup.SetPopup (theMenu: MenuHandle; theRsrcID, currentItem: INTEGER; redraw: BOOLEAN);
6 2485 --
6 2486 --      VAR
6 2487 --          fi:                 FailInfo;
6 2488 --          aMenu:              MenuHandle;
6 2489 --
6 2490 --          {-------------------------------+
6 2491 --          |   HandleFailure                |
6 2492 --          +-------------------------------}
6 2493 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 2494 0- B        BEGIN
6 2495 --             Free;
6 2496 -0 B        END;
6 2497 --
6 2498 0- A    BEGIN
6 2499 --          ReleasePopup;
6 2500 --          IF theMenu <> NIL THEN
6 2501 1-             BEGIN
6 2502 --             IF theRsrcID <> kNoResource THEN
6 2503 2-                BEGIN
6 2504 --                CatchFailures(fi, HandleFailure);
6 2505 --                DetachResource(Handle(theMenu));
6 2506 --                FailResError;
6 2507 --                Success(fi);
6 2508 -2                END;
6 2509 --             {$IFC FALSE}
6 2510 --             MAInsertMenu(theMenu, hierMenu);    { Use MAInsertMenu to ensure colors are set }
6 2511 --             {$ENDC}
6 2512 --             fMenuHandle := theMenu;
6 2513 --             fRsrcID := theRsrcID;
6 2514 --             fMenuID := theMenu^^.menuID;
6 2515 -1             END;
6 2516 --          SetCurrentItem(MAX(1, currentItem), kDontRedraw);
6 2517 --          AdjustBotRight;
6 2518 --          IF redraw THEN
6 2519 --             ForceRedraw;
6 2520 -0 A    END;
6 2521 --
6 2522 --      {$IFC qDebug}
6 2523 --          {$S DlgFields}
6 2524 --          {-----------------------------------+
6 2525 --          |   Fields                          |
6 2526 --          +-----------------------------------}
6 2527 -- A       PROCEDURE TPopup.Fields (PROCEDURE DoToField (fieldName: Str255;
6 2528 --                                                       fieldAddr: Ptr;
6 2529 --                                                       fieldType: INTEGER)); OVERRIDE;
6 2530 0- A       BEGIN
6 2531 --             DoToField('TPopup', NIL, bClass);
6 2532 --             DoToField('fRsrcID', @fRsrcID, bInteger);
6 2533 --             DoToField('fMenuID', @fMenuID, bInteger);
6 2534 --             DoToField('fMenuHandle', @fMenuHandle, bHandle);
6 2535 --             DoToField('fCurrentItem', @fCurrentItem, bInteger);
6 2536 --             DoToField('fItemOffset', @fItemOffset, bInteger);
6 2537 --             INHERITED Fields(DoToField);
6 2538 -0 A       END;
6 2539 --
6 2540 --      {$ENDC}
6 2541 --
6 2542 --
6 2543 --      (*****************************************************************************************)
6 2544 --      (*      T D i a l o g T E V i e w                                                        *)
6 2545 --      (*****************************************************************************************)
6 2546 --      {$S DlgRes}
6 2547 --      {-----------------------------------+
6 2548 --      |   DrawContents                    |
6 2549 --      +-----------------------------------}
6 2550 -- A    PROCEDURE TDialogTEView.DrawContents; OVERRIDE;
6 2551 0- A    BEGIN
6 2552 --          { Rely on the TEditText view to draw this }
6 2553 -0 A    END;
6 2554 --
6 2555 --
6 2556 --      {$IFC qDebug}
6 2557 --      {$S DlgFields}
6 2558 --      {-----------------------------------+
6 2559 --      |   Fields                          |
6 2560 --      +-----------------------------------}
6 2561 -- A    PROCEDURE TDialogTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
6 2562 --                                                          fieldAddr: Ptr;
6 2563 --                                                          fieldType: INTEGER)); OVERRIDE;
6 2564 0- A    BEGIN
6 2565 --          DoToField('TDialogTEView', NIL, bClass);
6 2566 --          DoToField('fEditText', @fEditText, bObject);
6 2567 --          INHERITED Fields(DoToField);
6 2568 -0 A    END;
6 2569 --      {$ENDC}
6 2570 --
6 2571 --
6 2572 --      {$S DlgNonRes}
6 2573 --      {-----------------------------------+
```

```
6 2574 --      |   InstallEditText                 |
6 2575 --      +-----------------------------------}
6 2576 -- A    PROCEDURE TDialogTEView.InstallEditText (theEditText: TEditText; selectChars: BOOLEAN);
6 2577 --
6 2578 --      VAR
6 2579 --          theText:        Str255;
6 2580 --
6 2581 0- A    BEGIN
6 2582 --          SetJustification(theEditText.fJust, kDontRedraw);
6 2583 --
6 2584 --          IF fSuperView <> NIL THEN
6 2585 --              fSuperView.RemoveSubView(SELF);
6 2586 --          theEditText.fSuperView.AddSubView(SELF);
6 2587 --
6 2588 --          fInset := theEditText.fInset;
6 2589 --          SetOneStyle(0, 0, doAll, theEditText.fTextStyle, kDontRedraw);
6 2590 --          {$H-}
6 2591 --          WITH theEditText.fSize DO
6 2592 --              Resize(h, Max(v, fHTE^^.lineHeight + fInset.top + fInset.bottom), kDontRedraw);
6 2593 --          WITH theEditText.fLocation DO
6 2594 --              Locate(h, v, kDontRedraw);
6 2595 --          {$H+}
6 2596 --
6 2597 --          theEditText.GetText(theText);
6 2598 --          SetText(theText);
6 2599 --          RecalcText;
6 2600 --          IF selectChars THEN
6 2601 --              SetSelect(0, MAXINT, fHTE)
6 2602 --          ELSE
6 2603 --              SetSelect(0, 0, fHTE);              { Caller will set the selection.        }
6 2604 --
6 2605 --          fMaxChars := theEditText.fMaxChars;
6 2606 --          fEditText := theEditText;
6 2607 --          theEditText.fTEView := SELF;
6 2608 --          theEditText.fIdleFreq := fIdleFreq;
6 2609 -0 A    END;
6 2610 --
6 2611 --
6 2612 --      {$S DlgNonRes}
6 2613 --      {-----------------------------------+
6 2614 --      |   InstallSelection                |
6 2615 --      +-----------------------------------}
6 2616 -- A    PROCEDURE TDialogTEView.InstallSelection(wasActive, beActive: BOOLEAN); OVERRIDE;
6 2617 --
6 2618 0- A    BEGIN
6 2619 --          { If we're deselecting a field and it's been horizontally scrolled, invalidate it
6 2620 --              so that it is redrawn correctly.}
6 2621 --          IF NOT beActive THEN
6 2622 --              IF fHTE^^.destRect.left <> fInset.left THEN
6 2623 1-                  BEGIN
6 2624 --                  InvalidRect(fHTE^^.viewRect);
6 2625 --                  fHTE^^.destRect.left := fInset.left;
6 2626 -1                  END;
6 2627 --
6 2628 --          INHERITED InstallSelection(wasActive, beActive);
6 2629 -0 A    END;
6 2630 --
6 2631 --
6 2632 --      {$S DlgRes}
6 2633 --      {-----------------------------------+
6 2634 --      |   RecalcText                      |   To force right margin out to far right
6 2635 --      +-----------------------------------}
6 2636 -- A    PROCEDURE TDialogTEView.RecalcText; OVERRIDE;
6 2637 0- A    BEGIN
6 2638 --          INHERITED RecalcText;
6 2639 --          IF fHTE^^.just = teJustLeft THEN        { This is awful kludgy.  Any better determiner? }
6 2640 --              fHTE^^.destRect.right := kMaxTEWidth;
6 2641 -0 A    END;
6 2642 --
6 2643 --
6 2644 --      {$S DlgRes}
6 2645 --      {-----------------------------------+
6 2646 --      |   ScrollSelectionIntoView         |
6 2647 --      +-----------------------------------}
6 2648 -- A    PROCEDURE TDialogTEView.ScrollSelectionIntoView; OVERRIDE;
6 2649 --
6 2650 0- A    BEGIN
6 2651 --      {$IFC NOT qNeedsROM128K}
6 2652 --          IF gConfiguration.hasROM128K THEN
6 2653 --      {$ENDC}
6 2654 --              IF Focus THEN
6 2655 --                  TESelView(fHTE);                { This assumes, of course, NOT fAutoWrap…       }
6 2656 -0 A    END;
6 2657 --
6 2658 --
6 2659 --      (*****************************************************************************************)
6 2660 --      (*      T S t a t i c T e x t                                                            *)
6 2661 --      (*****************************************************************************************)
6 2662 --
6 2663 --      {$IFC qProceduralViews}
6 2664 --      {$S DlgOpen}
6 2665 --      {-----------------------------------+
6 2666 --      |   IStaticText                     |
6 2667 --      +-----------------------------------}
6 2668 -- A    PROCEDURE TStaticText.IStaticText (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 2669 --                                         itsHSizeDet, itsVSizeDet: SizeDeterminer;
6 2670 --                                         itsRsrcID, itsIndex: INTEGER);
6 2671 --      VAR
6 2672 --          aString:        Str255;
```

```
6 2673 --          fi:             FailInfo;
6 2674 --
6 2675 --          {-------------------------------+
6 2676 --          |   HandleFailure               |
6 2677 --          +-------------------------------}
6 2678 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
6 2679 0- B        BEGIN
6 2680 --              Free;
6 2681 -0 B        END;
6 2682 --
6 2683 0- A    BEGIN
6 2684 --          fDataHandle := NIL;
6 2685 --          IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
6 2686 --          fRsrcID := itsRsrcID;
6 2687 --          fIndex := itsIndex;
6 2688 --          fJust := teJustLeft;                    { Default to system justification              }
6 2689 --          IF fRsrcID <> kNoResource THEN
6 2690 1-              BEGIN
6 2691 --              CatchFailures(fi, HandleFailure);
6 2692 --              GetIndString(aString, fRsrcID, fIndex);
6 2693 --              FailResError;
6 2694 --              Success(fi);
6 2695 --              SetText(aString, kDontRedraw);
6 2696 -1              END;
6 2697 --          ViewEnable(FALSE, kDontRedraw);         { Default is to not enable hit testing         }
6 2698 --          fDefChoice := mStaticTextHit;
6 2699 -0 A    END;
6 2700 --      {$ENDC}
6 2701 --
6 2702 --
6 2703 --      {$IFC qTemplateViews}
6 2704 --      {$S DlgOpen}
6 2705 --      {-----------------------------------+
6 2706 --      |   IRes                            |
6 2707 --      +-----------------------------------}
6 2708 -- A    PROCEDURE TStaticText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 2709 --
6 2710 0- A    BEGIN
6 2711 --          fDataHandle := NIL;
6 2712 --          INHERITED IRes(NIL, itsSuperView, itsParams);
6 2713 --
6 2714 --          fDefChoice := mStaticTextHit;
6 2715 --          WITH StaticTextTemplatePtr(itsParams)^ DO
6 2716 1-              BEGIN
6 2717 --              fJust := just;
6 2718 --              SetText(data, kDontRedraw);
6 2719 -1              END;
6 2720 --
6 2721 --          OffsetPtrWStr(itsParams, SIZEOF(StaticTextTemplate));
6 2722 -0 A    END;
6 2723 --      {$ENDC}
6 2724 --
6 2725 --
6 2726 --      {$IFC qWriteTemplates}
6 2727 --      {$S MAWriteRes}
6 2728 --      {-----------------------------------+
6 2729 --      |   WRes                            |
6 2730 --      +-----------------------------------}
6 2731 -- A    PROCEDURE TStaticText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2732 --
6 2733 --      VAR
6 2734 --          theText:    Str255;
6 2735 --          stPtr:      StaticTextTemplatePtr;
6 2736 --
6 2737 0- A    BEGIN
6 2738 --          INHERITED WRes(theResource, itsParams);
6 2739 --
6 2740 --          GetText(theText);
6 2741 --
6 2742 --          stPtr := StaticTextTemplatePtr(ExpandPtrWStr(theResource, itsParams,
6 2743 --            SIZEOF(StaticTextTemplate), LENGTH(theText)));
6 2744 --
6 2745 --          WITH stPtr^ DO
6 2746 1-              BEGIN
6 2747 --              just := fJust;
6 2748 --              (* data := theText; *)
6 2749 --              CopyStr255(theText, PRStr(data));
6 2750 -1              END;
6 2751 -0 A    END;
6 2752 --
6 2753 --      {$S MAWriteRes}
6 2754 --      {-----------------------------------+
6 2755 --      |   WriteRes                        |
6 2756 --      +-----------------------------------}
6 2757 -- A    PROCEDURE TStaticText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2758 0- A    BEGIN
6 2759 --          gWResSignature := 'stat';   gWResType := 'TStaticText';
6 2760 --          WRes(theResource, itsParams);
6 2761 -0 A    END;
6 2762 --      {$ENDC}
6 2763 --
6 2764 --
6 2765 --      {$S DlgClose}
6 2766 --      {-----------------------------------+
6 2767 --      |   Free                            |
6 2768 --      +-----------------------------------}
6 2769 -- A    PROCEDURE TStaticText.Free; OVERRIDE;
6 2770 0- A    BEGIN
6 2771 --          ReleaseText;
```

```
6 2772 --          INHERITED Free;
6 2773 -0 A    END;
6 2774 --
6 2775 --      {$S DlgRes}
6 2776 --      {-----------------------------------+
6 2777 --      |   DoSubstitution                  |
6 2778 --      +-----------------------------------}
6 2779 -- A    FUNCTION  TStaticText.DoSubstitution (VAR realText: Handle): BOOLEAN;
6 2780 --
6 2781 --      VAR
6 2782 --          err:            OSErr;
6 2783 --          aDialogView:    TDialogView;
6 2784 --
6 2785 0- A    BEGIN
6 2786 --          realText := fDataHandle;              { Make a copy of the original text             }
6 2787 --          err := HandToHand(realText);
6 2788 --          IF err <> noErr THEN                  { We just won't do substitution in an emergency }
6 2789 --              realText := fDataHandle
6 2790 --          ELSE
6 2791 1-              BEGIN
6 2792 --              aDialogView := TDialogView(GetDialogView);
6 2793 --              IF aDialogView <> NIL THEN
6 2794 --                  aDialogView.ReplaceText(realText);
6 2795 -1              END;
6 2796 --          DoSubstitution := (err = noErr);
6 2797 -0 A    END;
6 2798 --
6 2799 --      {$S DlgRes}
6 2800 --      {-----------------------------------+
6 2801 --      |   Draw                            |
6 2802 --      +-----------------------------------}
6 2803 -- A    PROCEDURE TStaticText.Draw (area: Rect); OVERRIDE;
6 2804 --      VAR
6 2805 --          subbed:         BOOLEAN;
6 2806 --          theRect:        Rect;
6 2807 --          oldColor:       RGBColor;
6 2808 --          realText:       Handle;
6 2809 --
6 2810 0- A    BEGIN
6 2811 --          IF fDataHandle <> NIL THEN
6 2812 1-              BEGIN
6 2813 --              subbed := DoSubstitution(realText); { Make the substitution if desired             }
6 2814 --              ControlArea(theRect);
6 2815 --              PenNormal; { ??? NECESSARY ??? }
6 2816 --              GetIfColor(oldColor);
6 2817 --              SetPortTextStyle(fTextStyle);
6 2818 --              HLock(realText);
6 2819 --              ImageText(Ptr(ORD4(realText^) + 1), { Draw the text itself                         }
6 2820 --                LENGTH(StringHandle(realText)^^), theRect, fJust);
6 2821 --              HUnlock(realText);
6 2822 --              IF subbed THEN
6 2823 --                  DisposHandle(realText);
6 2824 --              SetIfColor(oldColor);
6 2825 -1              END;
6 2826 --          INHERITED Draw(area);
6 2827 -0 A    END;
6 2828 --
6 2829 --      {$S DlgRes}
6 2830 --      {-----------------------------------+
6 2831 --      |   GetText                         |
6 2832 --      +-----------------------------------}
6 2833 -- A    PROCEDURE TStaticText.GetText (VAR theText: Str255);
6 2834 --
6 2835 0- A    BEGIN
6 2836 --          IF fDataHandle <> NIL THEN
6 2837 --              theText := StringHandle(fDataHandle)^^
6 2838 --          ELSE
6 2839 --              theText := '';
6 2840 -0 A    END;
6 2841 --
6 2842 --      {$S DlgRes}
6 2843 --      {-----------------------------------+
6 2844 --      |   ImageText                       |
6 2845 --      +-----------------------------------}
6 2846 -- A    PROCEDURE TStaticText.ImageText (text: Ptr; length: LONGINT; box: Rect; just: INTEGER);
6 2847 0- A    BEGIN
6 2848 --          TextBox(text, length, box, just);
6 2849 -0 A    END;
6 2850 --
6 2851 --      {$S DlgNonRes}
6 2852 --      {-----------------------------------+
6 2853 --      |   ReleaseText                     |
6 2854 --      +-----------------------------------}
6 2855 -- A    PROCEDURE TStaticText.ReleaseText;
6 2856 0- A    BEGIN
6 2857 --          DisposIfHandle(fDataHandle);
6 2858 --          fRsrcID := kNoResource;
6 2859 --          fDataHandle := NIL;
6 2860 -0 A    END;
6 2861 --
6 2862 --      {$S DlgNonRes}
6 2863 --      {-----------------------------------+
6 2864 --      |   SetJustification                |
6 2865 --      +-----------------------------------}
6 2866 -- A    PROCEDURE TStaticText.SetJustification (theJust: INTEGER; redraw: BOOLEAN);
6 2867 0- A    BEGIN
6 2868 --          fJust := theJust;
6 2869 --          IF redraw THEN
6 2870 --              ForceRedraw;
```

```
6 2871 -0 A   END;
6 2872 --
6 2873 --     {$S DlgNonRes}
6 2874 --     {-----------------------------------+
6 2875 --     |   SetText                         |
6 2876 --     +-----------------------------------}
6 2877 -- A   PROCEDURE TStaticText.SetText (theText: Str255; redraw: BOOLEAN);
6 2878 --     VAR
6 2879 --         area:       Rect;
6 2880 --
6 2881 0- A   BEGIN
6 2882 --         IF (fDataHandle = NIL) | (theText <> StringHandle(fDataHandle)^^) THEN
6 2883 1-             BEGIN
6 2884 --             ReleaseText;
6 2885 --             fDataHandle := Handle(NewString(theText));
6 2886 --             IF MemError <> noErr THEN
6 2887 --                 fDataHandle := NIL;
6 2888 --             IF redraw & Focus THEN
6 2889 2-                 BEGIN
6 2890 --                 ControlArea(area);
6 2891 --                 EraseRect(area);
6 2892 --                 Draw(area);
6 2893 -2                 END;
6 2894 -1             END;
6 2895 -0 A   END;
6 2896 --
6 2897 --     {$IFC qDebug}
6 2898 --         {$S DlgFields}
6 2899 --         {-----------------------------------+
6 2900 --         |   Fields                          |
6 2901 --         +-----------------------------------}
6 2902 -- A       PROCEDURE TStaticText.Fields (PROCEDURE DoToField (fieldName: Str255;
6 2903 --                                                            fieldAddr: Ptr;
6 2904 --                                                            fieldType: INTEGER)); OVERRIDE;
6 2905 0- A       BEGIN
6 2906 --             DoToField('TStaticText', NIL, bClass);
6 2907 --             DoToField('fRsrcID', @fRsrcID, bInteger);
6 2908 --             DoToField('fIndex', @fIndex, bInteger);
6 2909 --             DoToField('fDataHandle', @fDataHandle, bHandle);
6 2910 --             DoToField('fJust', @fJust, bInteger);
6 2911 --             INHERITED Fields(DoToField);
6 2912 -0 A         END;
6 2913 --     {$ENDC}
6 2914 --
6 2915 --
6 2916 --     (*****************************************************************************************)
6 2917 --     (*      T E d i t T e x t                                                                *)
6 2918 --     (*****************************************************************************************)
6 2919 --
6 2920 --     {$IFC qProceduralViews}
6 2921 --     {$S DlgOpen}
6 2922 --     {-----------------------------------+
6 2923 --     |   IEditText                       |
6 2924 --     +-----------------------------------}
6 2925 -- A   PROCEDURE TEditText.IEditText (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 2926 --                                    itsMaxChars: INTEGER);
6 2927 --     VAR
6 2928 --         itsMargins:     Rect;
6 2929 --
6 2930 0- A   BEGIN
6 2931 --         fTEView := NIL;
6 2932 --         IStaticText(itsSuperView, itsLocation, itsSize, sizeFixed, sizeFixed,
6 2933 --                     kNoResource, 0);
6 2934 --         fMaxChars := itsMaxChars;
6 2935 --         fControlChars := [chLeft, chRight, chUp, chDown, chBackspace];
6 2936 --         fTextStyle := gSystemStyle;
6 2937 --         Inset(2, 2, kDontRedraw);               { Default is a little, teeny inset…          }
6 2938 --         fPenSize := Point($00010001);           { …and a thin frame                          }
6 2939 --         fAdornment := kFrame;
6 2940 --         ViewEnable(TRUE, kDontRedraw);
6 2941 --         fDefChoice := mEditTextHit;
6 2942 -0 A   END;
6 2943 --     {$ENDC}
6 2944 --
6 2945 --
6 2946 --     {$IFC qTemplateViews}
6 2947 --     {$S DlgOpen}
6 2948 --     {-----------------------------------+
6 2949 --     |   IRes                            |
6 2950 --     +-----------------------------------}
6 2951 -- A   PROCEDURE TEditText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 2952 --
6 2953 0- A   BEGIN
6 2954 --         fTEView := NIL;
6 2955 --         INHERITED IRes(NIL, itsSuperView, itsParams);
6 2956 --
6 2957 --         WITH EditTextTemplatePtr(itsParams)^ DO
6 2958 1-             BEGIN
6 2959 --             fMaxChars := maxChars;
6 2960 --             fControlChars := controlChars;
6 2961 -1             END;
6 2962 --         fDefChoice := mEditTextHit;
6 2963 --
6 2964 --         OffsetPtr(itsParams, SIZEOF(EditTextTemplate));
6 2965 -0 A   END;
6 2966 --     {$ENDC}
6 2967 --
6 2968 --
6 2969 --     {$IFC qWriteTemplates}
```

```
6 2970 --      {$S MAWriteRes}
6 2971 --      {-----------------------------------+
6 2972 --      |   WRes                            |
6 2973 --      +-----------------------------------}
6 2974 -- A    PROCEDURE TEditText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2975 --
6 2976 --      VAR
6 2977 --          edPtr:      EditTextTemplatePtr;
6 2978 --
6 2979 0- A    BEGIN
6 2980 --          INHERITED WRes(theResource, itsParams);
6 2981 --
6 2982 --          edPtr := EditTextTemplatePtr(ExpandPtr(theResource, itsParams, SIZEOF(EditTextTemplate)));
6 2983 --
6 2984 --          WITH edPtr^ DO
6 2985 1-              BEGIN
6 2986 --              maxChars := fmaxChars;
6 2987 --              controlChars := fControlChars;
6 2988 -1              END;
6 2989 -0 A    END;
6 2990 --
6 2991 --      {$S MAWriteRes}
6 2992 --      {-----------------------------------+
6 2993 --      |   WriteRes                        |
6 2994 --      +-----------------------------------}
6 2995 -- A    PROCEDURE TEditText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 2996 0- A    BEGIN
6 2997 --          gWResSignature := 'edit';   gWResType := 'TEditText';
6 2998 --          WRes(theResource, itsParams);
6 2999 -0 A    END;
6 3000 --      {$ENDC}
6 3001 --
6 3002 --
6 3003 --      {$S DlgRes}
6 3004 --      {-----------------------------------+
6 3005 --      |   DoIdle                          |
6 3006 --      +-----------------------------------}
6 3007 -- A    PROCEDURE TEditText.DoIdle (phase: IdlePhase); OVERRIDE;
6 3008 --
6 3009 0- A    BEGIN
6 3010 --          IF fTEView <> NIL THEN
6 3011 --              fTEView.DoIdle(phase);
6 3012 -0 A    END;
6 3013 --
6 3014 --      {$S DlgRes}
6 3015 --      {-----------------------------------+
6 3016 --      |   DoKeyCommand                    |
6 3017 --      +-----------------------------------}
6 3018 -- A    FUNCTION TEditText.DoKeyCommand (ch: CHAR; aKeyCode: INTEGER; VAR info: EventInfo): TCommand; OVERRIDE;
6 3019 --
6 3020 --      VAR
6 3021 --          aCommand:       TCommand;
6 3022 --
6 3023 0- A    BEGIN
6 3024 --          IF (fTEView <> NIL) & ((ch >= ' ') | (ch IN fControlChars)) THEN
6 3025 1-              BEGIN
6 3026 --              aCommand := fTEView.DoKeyCommand(ch, aKeyCode, info);
6 3027 --              IF aCommand <> gNoChanges THEN
6 3028 --                  aCommand.fCanUndo := FALSE;     { !!! Temporary until commands have targets}
6 3029 --              DoKeyCommand := aCommand;
6 3030 -1              END
6 3031 --          ELSE
6 3032 --              DoKeyCommand := INHERITED DoKeyCommand(ch, aKeyCode, info);
6 3033 -0 A    END;
6 3034 --
6 3035 --      {$S DlgRes}
6 3036 --      {-----------------------------------+
6 3037 --      |   DoMenuCommand                   |
6 3038 --      +-----------------------------------}
6 3039 -- A    FUNCTION  TEditText.DoMenuCommand (aCmdNumber: CmdNumber): TCommand; OVERRIDE;
6 3040 --
6 3041 --      VAR
6 3042 --          aCommand:       TCommand;
6 3043 --
6 3044 0- A    BEGIN
6 3045 --          IF fTEView <> NIL THEN
6 3046 1-              BEGIN
6 3047 --              aCommand := fTEView.DoMenuCommand(aCmdNumber);
6 3048 --              IF aCommand <> gNoChanges THEN
6 3049 --                  aCommand.fCanUndo := FALSE;     { !!! Temporary until commands have targets}
6 3050 --              DoMenuCommand := aCommand;
6 3051 -1              END
6 3052 --          ELSE
6 3053 --              DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
6 3054 -0 A    END;
6 3055 --
6 3056 --
6 3057 --      {$S DlgRes}
6 3058 --      {-----------------------------------+
6 3059 --      |   DoMouseCommand                  |
6 3060 --      +-----------------------------------}
6 3061 -- A    FUNCTION  TEditText.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
6 3062 --                                          VAR hysteresis: Point): TCommand; OVERRIDE;
6 3063 --
6 3064 --      VAR
6 3065 --          aCommand:       TCommand;
6 3066 --
6 3067 0- A    BEGIN
6 3068 --          IF gTarget <> SELF THEN
```

```
6 3069 --             DoChoice(SELF, mEditTextHit);
6 3070 --
6 3071 --         { Make sure we were in fact selected before passing along the mouse click }
6 3072 --         IF gTarget = SELF THEN
6 3073 1-             BEGIN
6 3074 --             aCommand := fTEView.DoMouseCommand(theMouse, info, hysteresis);
6 3075 --             IF aCommand <> gNoChanges THEN
6 3076 --                 aCommand.fCanUndo := FALSE;     { !!! Temporary until commands have targets}
6 3077 --             DoMouseCommand := aCommand;
6 3078 -1             END
6 3079 --         ELSE
6 3080 --             DoMquseCommand := gNoChanges;
6 3081 -0 A    END;
6 3082 --
6 3083 --
6 3084 --     {$S DlgRes}
6 3085 --     {-----------------------------------+
6 3086 --     |   DoSetupMenus                    |
6 3087 --     +-----------------------------------}
6 3088 -- A   PROCEDURE TEditText.DoSetupMenus; OVERRIDE;
6 3089 --
6 3090 0- A   BEGIN
6 3091 --         IF fTEView <> NIL THEN
6 3092 --             fTEView.DoSetupMenus
6 3093 --         ELSE
6 3094 --             INHERITED DoSetupMenus;
6 3095 -0 A    END;
6 3096 --
6 3097 --
6 3098 --     {$S DlgRes}
6 3099 --     {-----------------------------------+
6 3100 --     |   DoSubstitution                  |
6 3101 --     +-----------------------------------}
6 3102 -- A   FUNCTION  TEditText.DoSubstitution (VAR realText: Handle): BOOLEAN; OVERRIDE;
6 3103 0- A   BEGIN
6 3104 --         { Default action is for editable text items is not to do any substitions }
6 3105 --         realText := fDataHandle;              { Simply return our stock text              }
6 3106 --         DoSubstitution := FALSE;              { We didn't substitute                      }
6 3107 -0 A    END;
6 3108 --
6 3109 --     {$S DlgRes}
6 3110 --     {-----------------------------------+
6 3111 --     |   Draw                            |
6 3112 --     +-----------------------------------}
6 3113 -- A   PROCEDURE TEditText.Draw (area: Rect); OVERRIDE;
6 3114 --
6 3115 --     VAR
6 3116 --         theRect:        Rect;
6 3117 --
6 3118 0- A   BEGIN
6 3119 --         IF fTEView <> NIL THEN
6 3120 1-             BEGIN
6 3121 --             fTEView.Draw(area);
6 3122 --             GetQDExtent(theRect);
6 3123 --             Adorn(theRect, fPenSize, fAdornment);
6 3124 -1             END
6 3125 --         ELSE
6 3126 --             INHERITED Draw(area);
6 3127 -0 A    END;
6 3128 --
6 3129 --     {$IFC qDebug}
6 3130 --         {$S DlgFields}
6 3131 --         {-----------------------------------+
6 3132 --         |   Fields                          |
6 3133 --         +-----------------------------------}
6 3134 -- A       PROCEDURE TEditText.Fields (PROCEDURE DoToField (fieldName: Str255;
6 3135 --                                                          fieldAddr: Ptr;
6 3136 --                                                          fieldType: INTEGER)); OVERRIDE;
6 3137 0- A       BEGIN
6 3138 --             DoToField('TEditText', NIL, bClass);
6 3139 --             DoToField('fMaxChars', @fMaxChars, bInteger);
6 3140 --             DoToField('fControlChars', @fControlChars, bHexLongInt);
6 3141 --             DoToField('fTEView', @fTEView, bObject);
6 3142 --             INHERITED Fields(DoToField);
6 3143 -0 A          END;
6 3144 --     {$ENDC}
6 3145 --
6 3146 --
6 3147 --     {$S DlgRes}
6 3148 --     {-----------------------------------+
6 3149 --     |   GetText                         |
6 3150 --     +-----------------------------------}
6 3151 -- A   PROCEDURE TEditText.GetText (VAR theText: Str255); OVERRIDE;
6 3152 --
6 3153 --     VAR
6 3154 --         theChars:       Handle;
6 3155 --         numberOfChars:  INTEGER;
6 3156 --
6 3157 0- A   BEGIN
6 3158 --         IF fTEView = NIL THEN
6 3159 --             INHERITED GetText(theText)
6 3160 --         ELSE
6 3161 1-             BEGIN
6 3162 --             theChars := fTEView.ExtractText;
6 3163 --             numberOfChars := MIN(255, GetHandleSize(theChars));
6 3164 --     {$IFC qRangeCheck}{$R-}{$ENDC}
6 3165 --             theText[0] := CHR(numberOfChars);
6 3166 --     {$IFC qRangeCheck}{$R+}{$ENDC}
6 3167 --             BlockMove(Ptr(theChars^), Ptr(ORD4(@theText)+1), numberOfChars);
```

```
6 3168 -1              END;
6 3169 -0 A    END;
6 3170 --
6 3171 --
6 3172 --      {-------------------------------+
6 3173 --      |   ImageText                   |
6 3174 --      +-------------------------------}
6 3175 -- A    PROCEDURE TEditText.ImageText (text: Ptr; length: LONGINT; box: Rect; just: INTEGER); OVERRIDE;
6 3176 --
6 3177 --      CONST
6 3178 --          TESysJust = $BAC;
6 3179 --
6 3180 --      VAR
6 3181 --          tmpHTE:         TEHandle;
6 3182 --
6 3183 0- A    BEGIN
6 3184 --          IF length >= 0 THEN
6 3185 1-              BEGIN
6 3186 --              IF just = teJustLeft THEN
6 3187 2-                  BEGIN
6 3188 --                  IF gConfiguration.hasScriptManager THEN
6 3189 --                      just := GetSysJust
6 3190 --                  ELSE
6 3191 --      {$IFC NOT qNeedsROM128K}
6 3192 --                      IF gConfiguration.hasROM128K THEN
6 3193 --      {$ENDC}
6 3194 --                          just := PInteger(TESysJust)^;
6 3195 -2                  END
6 3196 --              ELSE IF just = teForceLeft THEN
6 3197 --                  just := teJustLeft;
6 3198 --              { Could optimize for single lines, here… }
6 3199 --              EraseRect(box);
6 3200 --              tmpHTE := TENew(box, box);
6 3201 --              tmpHTE^^.just := just;
6 3202 --              IF just = teJustLeft THEN
6 3203 --                  WITH tmpHTE^^ DO
6 3204 2-                      BEGIN
6 3205 --                      crOnly := -1;
6 3206 --                      destRect.right := kMaxTEWidth;
6 3207 -2                      END;
6 3208 --              TESetText(text, length, tmpHTE);
6 3209 --              TEUpdate(box, tmpHTE);
6 3210 --              TEDispose(tmpHTE);
6 3211 -1              END;
6 3212 -0 A    END;
6 3213 --
6 3214 --      {$S DlgRes}
6 3215 --      {-----------------------------------+
6 3216 --      |   Locate                          |
6 3217 --      +-----------------------------------}
6 3218 -- A    PROCEDURE TEditText.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
6 3219 --
6 3220 0- A    BEGIN
6 3221 --          INHERITED Locate(h, v, invalidate);
6 3222 --          IF fTEView <> NIL THEN
6 3223 --              fTEView.Locate(h, v, FALSE);
6 3224 -0 A    END;
6 3225 --
6 3226 --
6 3227 --      {$S DlgRes}
6 3228 --      {-----------------------------------+
6 3229 --      |   Resize                          |
6 3230 --      +-----------------------------------}
6 3231 -- A    PROCEDURE TEditText.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
6 3232 --
6 3233 0- A    BEGIN
6 3234 --          INHERITED Resize(width, height, invalidate);
6 3235 --          IF fTEView <> NIL THEN
6 3236 --              fTEView.Resize(width, height, FALSE);
6 3237 -0 A    END;
6 3238 --
6 3239 --
6 3240 --      {$S DlgNonRes}
6 3241 --      {-----------------------------------+
6 3242 --      |   RestartEdit                     |
6 3243 --      +-----------------------------------}
6 3244 -- A    PROCEDURE TEditText.RestartEdit;
6 3245 --
6 3246 0- A    BEGIN
6 3247 --          IF fTEView.Focus THEN
6 3248 --              TESetSelect(0, MAXINT, fTEView.fHTE);
6 3249 -0 A    END;
6 3250 --
6 3251 --
6 3252 --      {$S DlgNonRes}
6 3253 --      {-----------------------------------+
6 3254 --      |   SetJustification                |
6 3255 --      +-----------------------------------}
6 3256 -- A    PROCEDURE TEditText.SetJustification (theJust: INTEGER; redraw: BOOLEAN);
6 3257 0- A    BEGIN
6 3258 --          IF fTEView <> NIL THEN
6 3259 --              fTEView.SetJustification(theJust, kDontRedraw);
6 3260 --          INHERITED SetJustification(theJust, redraw);
6 3261 -0 A    END;
6 3262 --
6 3263 --      {$S DlgRes}
6 3264 --      {-----------------------------------+
6 3265 --      |   SetSelection                    |
6 3266 --      +-----------------------------------}
```

```
6 3267 -- A   PROCEDURE TEditText.SetSelection (selStart, selEnd: INTEGER; redraw: BOOLEAN);
6 3268 --
6 3269 0- A   BEGIN
6 3270 --         IF fTEView <> NIL THEN
6 3271 --             IF redraw & fTEView.Focus THEN
6 3272 --                 TESetSelect(selStart, selEnd, fTEView.fHTE)
6 3273 --             ELSE
6 3274 --                 SetSelect(selStart, selEnd, fTEView.fHTE);
6 3275 -0 A   END;
6 3276 --
6 3277 --     {$S DlgNonRes}
6 3278 --     {-----------------------------------+
6 3279 --     |   SetText                         |
6 3280 --     +-----------------------------------}
6 3281 -- A   PROCEDURE TEditText.SetText (theText: Str255; redraw: BOOLEAN); OVERRIDE;
6 3282 --
6 3283 --     VAR
6 3284 --         currentText:    Str255;
6 3285 --         area:           Rect;
6 3286 --
6 3287 0- A   BEGIN
6 3288 --         IF fTEView <> NIL THEN
6 3289 1-             BEGIN
6 3290 --             GetText(currentText);
6 3291 --             IF currentText <> theText THEN
6 3292 2-                 BEGIN
6 3293 --                 fTEView.SetText(theText);
6 3294 --                 fTEView.RecalcText;
6 3295 --                 IF redraw & Focus THEN
6 3296 3-                     BEGIN
6 3297 --                     ControlArea(area);
6 3298 --                     EraseRect(area);
6 3299 --                     Draw(area);
6 3300 -3                     END;
6 3301 -2                 END;
6 3302 -1             END
6 3303 --         ELSE
6 3304 --             INHERITED SetText(theText, redraw);
6 3305 -0 A   END;
6 3306 --
6 3307 --
6 3308 --     {$S DlgRes}
6 3309 --     {-----------------------------------+
6 3310 --     |   InstallSelection                |
6 3311 --     +-----------------------------------}
6 3312 -- A   PROCEDURE TEditText.InstallSelection (wasActive, beActive: BOOLEAN); OVERRIDE;
6 3313 --
6 3314 --     VAR
6 3315 --         itsDialogView:  TDialogView;
6 3316 --         myExtent:       VRect;
6 3317 --         minToSee:       Point;
6 3318 --         aString:        Str255;
6 3319 --
6 3320 0- A   BEGIN
6 3321 --         IF fTEView <> NIL THEN
6 3322 --             fTEView.InstallSelection(wasActive, beActive);
6 3323 -0 A   END;
6 3324 --
6 3325 --
6 3326 --     {$S DlgNonRes}
6 3327 --     {-----------------------------------+
6 3328 --     |   StartEdit                       |
6 3329 --     +-----------------------------------}
6 3330 -- A   PROCEDURE TEditText.StartEdit (selectChars: BOOLEAN; theTEView: TDialogTEView);
6 3331 --
6 3332 --     VAR
6 3333 --         myExtent:       VRect;
6 3334 --         minToSee:       Point;
6 3335 --         itsWindow:      TWindow;
6 3336 --
6 3337 0- A   BEGIN
6 3338 --         theTEView.InstallEditText(SELF, selectChars);
6 3339 --
6 3340 --         itsWindow := GetWindow;                 { Set the window's target, which sets   }
6 3341 --         IF itsWindow <> NIL THEN                { ...the application's target if it is  }
6 3342 --             itsWindow.SetTarget(SELF);          { ...the front window.                  }
6 3343 --
6 3344 --         GetExtent(myExtent);
6 3345 --         InsetVRect(myExtent, -10, -10);
6 3346 --         minToSee.h := MIN(fSize.h + 10, kMaxCoord);
6 3347 --         minToSee.v := MIN(fSize.v + 10, kMaxCoord);
6 3348 --
6 3349 --         RevealRect(myExtent, minToSee, kVisible);
6 3350 -0 A   END;
6 3351 --
6 3352 --
6 3353 --     {$S DlgNonRes}
6 3354 --     {-----------------------------------+
6 3355 --     |   StopEdit                        |
6 3356 --     +-----------------------------------}
6 3357 -- A   FUNCTION TEditText.StopEdit: BOOLEAN;
6 3358 --
6 3359 --     VAR
6 3360 --         aString:        Str255;
6 3361 --
6 3362 0- A   BEGIN
6 3363 --         StopEdit := FALSE;                      { Assume the worst              }
6 3364 --         IF Validate THEN
6 3365 1-             BEGIN
```

```
6 3366 --             GetText(aString);
6 3367 --             fTEView.InstallSelection(TRUE, FALSE);
6 3368 --             fTEView := NIL;
6 3369 --             SetText(aString, kDontRedraw);
6 3370 --             StopEdit := TRUE;
6 3371 -1             END;
6 3372 -0 A   END;
6 3373 --
6 3374 --
6 3375 --     {$S DlgRes}
6 3376 --     {-----------------------------------+
6 3377 --     |   Validate                        |
6 3378 --     +-----------------------------------}
6 3379 -- A   FUNCTION  TEditText.Validate: BOOLEAN;
6 3380 --
6 3381 0- A   BEGIN
6 3382 --         Validate := TRUE;
6 3383 -0 A   END;
6 3384 --
6 3385 --
6 3386 --     (*****************************************************************************************)
6 3387 --     (*      T N u m b e r T e x t                                                            *)
6 3388 --     (*****************************************************************************************)
6 3389 --
6 3390 --     {$IFC qProceduralViews}
6 3391 --     {$S DlgOpen}
6 3392 --     {-----------------------------------+
6 3393 --     |   INumberText                     |
6 3394 --     +-----------------------------------}
6 3395 -- A   PROCEDURE TNumberText.INumberText (itsSuperView: TView; itsLocation, itsSize: VPoint;
6 3396 --                                       itsValue, itsMinimum, itsMaximum: LONGINT);
6 3397 --     VAR
6 3398 --         aString:        Str255;
6 3399 --
6 3400 0- A   BEGIN
6 3401 --         IEditText(itsSuperView, itsLocation, itsSize, 255);
6 3402 --         fMinimum := itsMinimum;
6 3403 --         fMaximum := itsMaximum;
6 3404 --         NumToString(itsValue, aString);
6 3405 --         SetText(aString, kDontRedraw);
6 3406 -0 A   END;
6 3407 --     {$ENDC}
6 3408 --
6 3409 --
6 3410 --     {$IFC qTemplateViews}
6 3411 --     {$S DlgOpen}
6 3412 --     {-----------------------------------+
6 3413 --     |   IRes                            |
6 3414 --     +-----------------------------------}
6 3415 -- A   PROCEDURE TNumberText.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
6 3416 --
6 3417 --     VAR
6 3418 --         aString:        Str255;
6 3419 --
6 3420 0- A   BEGIN
6 3421 --         INHERITED IRes(NIL, itsSuperView, itsParams);
6 3422 --
6 3423 --         WITH NumberTextTemplatePtr(itsParams)^ DO
6 3424 1-             BEGIN
6 3425 --             NumToString(value, aString);
6 3426 --             SetText(aString, kDontRedraw);
6 3427 --             fMinimum := minimum;
6 3428 --             fMaximum := maximum;
6 3429 -1             END;
6 3430 --
6 3431 --         OffsetPtr(itsParams, SIZEOF(NumberTextTemplate));
6 3432 -0 A   END;
6 3433 --     {$ENDC}
6 3434 --
6 3435 --
6 3436 --     {$IFC qWriteTemplates}
6 3437 --     {$S MAWriteRes}
6 3438 --     {-----------------------------------+
6 3439 --     |   WRes                            |
6 3440 --     +-----------------------------------}
6 3441 -- A   PROCEDURE TNumberText.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
6 3442 --
6 3443 --     VAR
6 3444 --         nmPtr:      NumberTextTemplatePtr;
6 3445 --
6 3446 0- A   BEGIN
6 3447 --         INHERITED WRes(theResource, itsParams);
6 3448 --
6 3449 --         nmPtr := NumberTextTemplatePtr(ExpandPtr(theResource, itsParams,
6 3450 --           SIZEOF(NumberTextTemplate)));
6 3451 --
6 3452 --         WITH nmPtr^ DO
6 3453 1-             BEGIN
6 3454 --             value := GetValue;
6 3455 --             minimum := fMinimum;
6 3456 --             maximum := fMaximum;
6 3457 -1             END;
6 3458 -0 A   END;
6 3459 --
6 3460 --     {$S MAWriteRes}
6 3461 --     {-----------------------------------+
6 3462 --     |   WriteRes                        |
6 3463 --     +-----------------------------------}
6 3464 -- A   PROCEDURE TNumberText.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
```

```
6 3465 0- A   BEGIN
6 3466 --         gWResSignature := 'nmbr';   gWResType := 'TNumberText';
6 3467 --         WRes(theResource, itsParams);
6 3468 -0 A   END;
6 3469 --     {$ENDC}
6 3470 --
6 3471 --
6 3472 --     {$S DlgRes}
6 3473 --     {-----------------------------------+
6 3474 --     |   GetValue                        |
6 3475 --     +-----------------------------------}
6 3476 -- A   FUNCTION  TNumberText.GetValue: LONGINT;
6 3477 --
6 3478 --     VAR
6 3479 --         aString:        Str255;
6 3480 --         theValue:       LONGINT;
6 3481 --
6 3482 0- A   BEGIN
6 3483 --         GetText(aString);
6 3484 --         StringToNum(aString, theValue);
6 3485 --         GetValue := theValue;
6 3486 -0 A   END;
6 3487 --
6 3488 --
6 3489 --     {$S DlgNonRes}
6 3490 --     {-----------------------------------+
6 3491 --     |   SetValue                        |
6 3492 --     +-----------------------------------}
6 3493 -- A   PROCEDURE TNumberText.SetValue (newValue: LONGINT; redraw: BOOLEAN);
6 3494 --
6 3495 --     VAR
6 3496 --         aString:        Str255;
6 3497 --
6 3498 0- A   BEGIN
6 3499 --         newValue := MAX(fMinimum, MIN(fMaximum, newValue));
6 3500 --         NumToString(newValue, aString);
6 3501 --         SetText(aString, redraw);
6 3502 -0 A   END;
6 3503 --
6 3504 --
6 3505 --     {$S DlgNonRes}
6 3506 --     {-----------------------------------+
6 3507 --     |   Validate                        |
6 3508 --     +-----------------------------------}
6 3509 -- A   FUNCTION  TNumberText.Validate: BOOLEAN; OVERRIDE;
6 3510 --
6 3511 --     VAR
6 3512 --         theString:      Str255;
6 3513 --         decRec:         Decimal;
6 3514 --         extValue:       Extended;
6 3515 --         index:          INTEGER;
6 3516 --         validPrefix:    BOOLEAN;
6 3517 --
6 3518 0- A   BEGIN
6 3519 --         GetText(theString);
6 3520 --         IF LENGTH(theString) > 0 THEN
6 3521 1-             BEGIN
6 3522 --             Validate := FALSE;
6 3523 --             index := 1;
6 3524 --             Str2Dec(theString, index, decRec, validPrefix);
6 3525 --             IF validPrefix AND (index > LENGTH(theString)) AND (decRec.exp >= 0) THEN
6 3526 2-                 BEGIN
6 3527 --                 extValue := Dec2Num(decRec);
6 3528 --                 Validate := (extValue >= fMinimum) & (extValue <= fMaximum);
6 3529 -2                 END;
6 3530 -1             END
6 3531 --         ELSE
6 3532 --             Validate := TRUE;
6 3533 -0 A   END;
6 3534 --
6 3535 --     {$IFC qDebug}
6 3536 --         {$S DlgFields}
6 3537 --         {-----------------------------------+
6 3538 --         |   Fields                          |
6 3539 --         +-----------------------------------}
6 3540 -- A       PROCEDURE TNumberText.Fields (PROCEDURE DoToField (fieldName: Str255;
6 3541 --                                                            fieldAddr: Ptr;
6 3542 --                                                            fieldType: INTEGER)); OVERRIDE;
6 3543 0- A       BEGIN
6 3544 --             DoToField('TNumberText', NIL, bClass);
6 3545 --             DoToField('fMinimum', @fMinimum, bLongInt);
6 3546 --             DoToField('fMaximum', @fMaximum, bLongInt);
6 3547 --             INHERITED Fields(DoToField);
6 3548 -0 A       END;
6 3549 --     {$ENDC}
5  701 --
5  702 --     END.
```

```
7    1 --    {$P}
7    2 --    { UFailure.p }
7    3 --    { Copyright 1985-1988 by Apple Computer, Inc.  All rights reserved. }
7    4 --
7    5 --    { Conditional compile flags used by this unit:
7    6 --
7    7 --        qDebug          True includes debugging code, false ignores debugging code.
7    8 --
7    9 --        qTrace          True surrounds trap patches with $D+ and $D++ (because
7   10 --                        the default is $D++), false skips these directives.
7   11 --
7   12 --        The settings of $D, $R and $N are imported from UMAUtil.
7   13 --     }
7   14 --
7   15 --
7   16 --    UNIT UFailure;
7   17 --
7   18 --    (*
7   19 --                        T H E O R Y   O F   O P E R A T I O N
7   20 --
7   21 --        This unit implements the MacApp failure mechanism.
7   22 --
7   23 --        The failure mechanism is built around exception handlers.  An exception
7   24 --        handler is a routine, generally local to some other routine, that is
7   25 --        called when a failure occurs and takes action to handle the failure.
7   26 --        An exception handler is of the form
7   27 --
7   28 --            PROCEDURE ExceptionHandler (error: OSErr; message: LONGINT);
7   29 --
7   30 --        where error is the error that caused the failure, and message identifies
7   31 --        the error message that may be displayed.  Consider a routine that opens
7   32 --        a file, reads its contents, and closes the file.  If a failure occured
7   33 --        while reading the file, an exception handler would be needed to close the
7   34 --        file, as the rest of the routine will not be executed.  (See the example
7   35 --        at the end of these comments.)
7   36 --
7   37 --        References to expection handlers are defined by the FailInfo record.
7   38 --        The exception handlers form a linked-list via the nextInfo field of
7   39 --        FailInfo.  The linked list is a stack since new exception handlers
7   40 --        are added to the front of the list.
7   41 --
7   42 --        New exception handlers are added to the stack with the CatchFailures
7   43 --        procedure, and removed from the stack with the Success procedure.  In
7   44 --        general you call CatchFailures to post an exception handler when an
7   45 --        error the application should handle might occur, and call Success to
7   46 --        remove the handler from the stack, after the handler is no longer
7   47 --        needed (i.e. the possibility of error no longer exists).  Any failure
7   48 --        detected within the limits of the CatchFailures and subsequent Success
7   49 --        call results in the execution of the exception handler.  (Failure does
7   50 --        not have to occur in the same routine as your call to CatchFailures.
7   51 --        The failure may occur in any routine called after CatchFailures but
7   52 --        before Success.)
7   53 --
7   54 --        When MacApp (or your code) determines that a failure has occured, it
7   55 --        calls Failure.  As a convenience, several procedures are provided
7   56 --        to check for standard kinds of failures and call Failure if needed.
7   57 --        These procedures are:
7   58 --
7   59 --            FailNIL         Calls Failure if its parameter is NIL.
7   60 --            FailOSErr       Calls Failure if its parameter is not noErr.
7   61 --            FailMemError    Calls Failure if MemError returns other than noErr.
7   62 --            FailResError    Calls Failure if ResError returns other than noErr.
7   63 --
7   64 --        When Failure is called, execution of the routine that called Failure is
7   65 --        terminated and the exception handler at the top of the stack is popped.
7   66 --        For each routine that was called after the handler was posted
7   67 --        to the stack, execution is terminated as though from an EXIT statement.
7   68 --        Then the exception handler is called.  It generally cleans up for the
7   69 --        routine in which it is nested.  Upon completion the next exception handler
7   70 --        is popped from the stack, repeating the process.
7   71 --
7   72 --        The error causing the failure, and a message code is passed to Failure.
7   73 --        For MacApp, the last exception handler on the stack is the one in
7   74 --        TApplication.PollEvent.  It calls TApplication.ShowError, which calls
7   75 --        ErrorAlert, which decodes the message and displays an alert.  You exception
7   76 --        handlers may set the message code to one more specific to your application
7   77 --        by calling FailNewMessage at the end of your exception handler.
7   78 --        FailNewMessage changes the message only if the current one is non-zero.
7   79 --        This has the effect of allowing those exception handlers closest to the
7   80 --        source of the error to set the message.
7   81 --
7   82 --        One last note about exception handlers:  It is possible for an exception
7   83 --        handler to terminate exception processing by using a non-local GOTO to
7   84 --        jump back into the routine in which the exception handler is nested.  This
7   85 --        is how MacApp keeps the application running when a failure occurs.  The
7   86 --        last exception handler on the stack, in TApplication.PollEvent, uses a
7   87 --        GOTO to continue event processing.
7   88 --
7   89 --        The following is an example showing the use of exception handlers.
7   90 --        The ReadTheFile procedure reads the contents of a file, signalling
7   91 --        failure if the open, read, or close returns an error.
7   92 --
7   93 --            PROCEDURE ReadTheFile (fileName: Str255; volRefNum: INTEGER;
7   94 --                                   VAR contents: ContentsRecord);
7   95 --            VAR
7   96 --                fi:         FailInfo;
7   97 --                count:      LONGINT;
7   98 --
```

```
7   99 --                PROCEDURE ExceptionHandler (error: OSErr; message: LONGINT);
7  100 --                VAR
7  101 --                    err:    OSErr;
7  102 --                BEGIN
7  103 --                    err := FSClose(fileRefNum);     { Make sure the file is closed }
7  104 --                    { Pass on my own error message... }
7  105 --                    FailNewMessage(error, message, kMyErrorMessage);
7  106 --                END;
7  107 --
7  108 --            BEGIN
7  109 --                { If FSOpen fails we don't need to do any failure handling,
7  110 --                    though the guy that called us probably will. }
7  111 --                FailOSErr(FSOpen(fileName, volRefNum, fileRefNum));
7  112 --
7  113 --                { Now that the file is open, if the read fails we must make
7  114 --                    sure to close the file. }
7  115 --                CatchFailures(fi, Exceptionhandler);
7  116 --
7  117 --                count := SIZEOF(ContentsRecord);
7  118 --                FailOSErr(FSRead(fileRefNum, count, @contents));
7  119 --
7  120 --                { The file's been read so we don't need our exception handler
7  121 --                    anymore. }
7  122 --                Success(fi);
7  123 --
7  124 --                { Again, the guy that called us will have to deal with this. }
7  125 --                FailOSErr(FSClose(fileRefNum));
7  126 --            END;
7  127 --    *)
7  128 --
7  129 --    INTERFACE
7  130 --
7  131 --    {-----------------------------------+
7  132 --    |   Uses                            |
7  133 --    +-----------------------------------}
7  134 --    USES
7  135 --        {$LOAD MacIntf.LOAD}
7  136 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
7  137 --        {$LOAD}
7  138 --            UMAUtil
7  139 --        {$IFC qDebug}
7  140 --            , UWritelnWindow
7  141 --            , UTrace
7  142 --        {$ENDC}
7  143 --            ;
7  144 --
7  145 --    {-----------------------------------+
7  146 --    |   Global types                    |
7  147 --    +-----------------------------------}
7  148 --    TYPE
7  149 --            { used for the exception handling mechanism }
7  150 --        PFailInfo =     ^FailInfo;
7  151 --        FailInfo =      RECORD
7  152 --                        regs:           ARRAY[1..11] OF LONGINT;
7  153 --                        error:          INTEGER;
7  154 --                        message:        LONGINT;
7  155 --                        failA6:         LONGINT;
7  156 --                        failPC:         LONGINT;
7  157 --                        nextInfo:       PFailInfo;
7  158 --                        {$IFC qDebug}
7  159 --                        whoPC:          LONGINT;
7  160 --                        {$ENDC}
7  161 --                        END;
7  162 --
7  163 --    {-----------------------------------+
7  164 --    |   Global variables                |
7  165 --    +-----------------------------------}
7  166 --    {$IFC qDebug}
7  167 --    VAR
7  168 --        gAskAboutAlloc:     BOOLEAN;    { if TRUE ask about each allocation to force
7  169 --                                            failure }
7  170 --        gAskFailure:        BOOLEAN;    { if TRUE ask about calls to
7  171 --                                            FailOSErr, FailResError, FailMemError,
7  172 --                                            and give user a chance to force error }
7  173 --    {$ENDC qDebug}
7  174 --
7  175 --
7  176 --    {-----------------------------------+
7  177 --    |   Global routines                 |
7  178 --    +-----------------------------------}
7  179 --    PROCEDURE InitUFailure;
7  180 --        { Initializes this unit.  Must be called before using any features of
7  181 --            this unit. }
7  182 --
7  183 --    FUNCTION  BuildMessage(lowWord, highWord: INTEGER): LONGINT;
7  184 --                INLINE $2E9F;   { MOVE.L        (A7)+,(A7)    }
7  185 --        { Takes the 2 integers and combines them into a LONGINT.  Note that the
7  186 --            low-order word is the first parameter. }
7  187 --
7  188 --    PROCEDURE CatchFailures(VAR fi: FailInfo;
7  189 --                            PROCEDURE Handler(e: INTEGER; m: LONGINT));
7  190 --        { Call this to set up an exception handler. This pushes your handler onto
7  191 --            a stack of exception handlers. }
7  192 --
7  193 --    PROCEDURE Failure(error: INTEGER; message: LONGINT);
7  194 --        { Call this to signal a failure.  Control will branch to the most recent
7  195 --            exception handler, which will be popped off the handler stack. }
7  196 --
7  197 --    PROCEDURE FailMemError;
```

```
7  198 --          { IF MemError <> noErr THEN Failure(MemError, 0);  If you are using
7  199 --              assembler, then you should just test the return code from the Memory
7  200 --              Manager in DO by calling FailOSErr. (See the discussion of MemError in
7  201 --              Inside Macintosh.) }
7  202 --
7  203 --      PROCEDURE FailResError;
7  204 --          { IF ResError <> noErr THEN Failure(ResError, 0); (See Inside Macintosh.) }
7  205 --
7  206 --      PROCEDURE FailNewMessage(error: INTEGER; oldMessage, newMessage: LONGINT);
7  207 --          { This does:
7  208 --              IF oldMessage = 0 THEN
7  209 --                  Failure(error, newMessage)
7  210 --              ELSE
7  211 --                  Failure(error, oldMessage);
7  212 --           }
7  213 --
7  214 --      PROCEDURE FailNIL(p: UNIV Ptr);
7  215 --          { Call this with a pointer/handler; this signals Failure(memFullErr, 0) iff
7  216 --              the pointer is NIL. }
7  217 --
7  218 --      PROCEDURE FailOSErr(error: INTEGER);
7  219 --          { Call this with an OSError; signals Failure(error, 0) iff error <> noErr. }
7  220 --
7  221 --      PROCEDURE SetInitHandler(handler: ProcPtr);
7  222 --          { Sets an initialization error handler; applications should not call
7  223 --              this, normally. }
7  224 --
7  225 --      PROCEDURE Success(VAR fi: FailInfo);
7  226 --          { Call this when you want to de-install your exception handler (pop 1
7  227 --              element off the handler stack). }
7  228 --
7  229 --      {$IFC qDebug}
7  230 --      PROCEDURE ProgramBreak(grievance: Str255);
7  231 --          { Causes a program break to the debug window, displaying grievance. }
7  232 --
7  233 --      PROCEDURE ProgramReport(grievance: Str255; break: BOOLEAN);
7  234 --          { Displays grievance in the debug window, and breaks if break is true. }
7  235 --      {$ENDC}
7  236 --
7  237 --
7  238 --      IMPLEMENTATION
7  239 --
7  240 --      {$I UFailure.inc1.p}
```

```
8    1 --     {$P}
8    2 --     {UFailure.incl.p}
8    3 --     {Copyright 1985-1988 Apple Computer, Inc.  All rights reserved.}
8    4 --
8    5 --     {-----------------------------------+
8    6 --     |   Local variables                 |
8    7 --     +-----------------------------------}
8    8 --     VAR
8    9 --         {$Z+}   {make gTopHandler accessable by assembly code}
8   10 --         gTopHandler:        PFailInfo;  {linked list of failure handlers}
8   11 --         {$Z-}
8   12 --         gInitHandler:       ProcPtr;
8   13 --
8   14 --
8   15 --     {-----------------------------------+
8   16 --     |   External Declarations           |
8   17 --     +-----------------------------------}
8   18 -- A   PROCEDURE CatchFailures (VAR fi: FailInfo;
8   19 --                              PROCEDURE Handler(e: INTEGER; m: LONGINT)); EXTERNAL;
8   20 --
8   21 -- A   PROCEDURE DoFailure (pf: PFailInfo); EXTERNAL;
8   22 --
8   23 --     {-----------------------------------+
8   24 --     |   CallInitHandler                 |
8   25 --     +-----------------------------------}
8   26 -- A   PROCEDURE CallInitHandler (error: INTEGER; message: LONGINT; p: ProcPtr);
8   27 --             INLINE  $205F,      {MOVE.L     (A7)+,A0    }
8   28 --                     $4E90;      {JMP        (A0)        }
8   29 --
8   30 --     {$S MAMain}
8   31 --     {-----------------------------------+
8   32 --     |   FailMemError                    |
8   33 --     +-----------------------------------}
8   34 -- A   PROCEDURE FailMemError;
8   35 --     VAR
8   36 --         e:  OSErr;
8   37 --     {$IFC qDebug}
8   38 --         s:      Str255;
8   39 --     {$ENDC}
8   40 0- A   BEGIN
8   41 --         e := MemError;
8   42 --
8   43 --     {$IFC qDebug}
8   44 --         IF gAskFailure AND (e = noErr) AND CanReadLn THEN
8   45 1-            BEGIN
8   46 --     {$%+}
8   47 --             GetMethodName(%_GetA6+4, s);
8   48 --     {$%-}
8   49 --             e := ReadInteger(CONCAT('FailMemError called by ', s, '.  Enter return error: '));
8   50 -1             END;
8   51 --     {$ENDC}
8   52 --
8   53 --         IF e <> noErr THEN
8   54 --             Failure(e, 0);
8   55 -0 A   END {FailMemError};
8   56 --
8   57 --
8   58 --     {$S MAMain}
8   59 --     {-----------------------------------+
8   60 --     |   FailNIL                         |
8   61 --     +-----------------------------------}
8   62 -- A   PROCEDURE FailNIL (p: UNIV Ptr);
8   63 0- A   BEGIN
8   64 --         { no check for gAskFailure here, since we do this when objects are created. }
8   65 --         IF p = NIL THEN
8   66 --             Failure(memFullErr, 0);
8   67 -0 A   END {FailNIL};
8   68 --
8   69 --
8   70 --     {$S MAMain}
8   71 --     {-----------------------------------+
8   72 --     |   FailNewMessage                  |
8   73 --     +-----------------------------------}
8   74 -- A   PROCEDURE FailNewMessage (error: INTEGER; oldMessage, newMessage: LONGINT);
8   75 0- A   BEGIN
8   76 --         IF oldMessage = 0 THEN
8   77 --             oldMessage := newMessage;
8   78 --         Failure(error, oldMessage);
8   79 -0 A   END {FailNewMessage};
8   80 --
8   81 --
8   82 --     {$S MAMain}
8   83 --     {-----------------------------------+
8   84 --     |   FailOSErr                       |
8   85 --     +-----------------------------------}
8   86 -- A   PROCEDURE FailOSErr (error: INTEGER);
8   87 --
8   88 --     {$IFC qDebug}
8   89 --     VAR
8   90 --         s:      Str255;
8   91 --     {$ENDC}
8   92 --
8   93 0- A   BEGIN
8   94 --     {$IFC qDebug}
8   95 --         IF gAskFailure AND (error = noErr) AND CanReadLn THEN
8   96 1-            BEGIN
8   97 --     {$%+}
8   98 --             GetMethodName(%_GetA6+4, s);
```

```
8   99 --      {$%-}
8  100 --              error := ReadInteger(CONCAT('FailOSErr called by ', s, '.  Enter return error: '));
8  101 -1              END;
8  102 --      {$ENDC}
8  103 --
8  104 --          IF error <> noErr THEN
8  105 --              Failure(error, 0);
8  106 -0 A    END {FailOSErr};
8  107 --
8  108 --
8  109 --      {$S MAMain}
8  110 --      {-----------------------------------+
8  111 --      |   FailResError                    |
8  112 --      +-----------------------------------}
8  113 -- A    PROCEDURE FailResError;
8  114 --      VAR
8  115 --          e:  OSErr;
8  116 --      {$IFC qDebug}
8  117 --          s:      Str255;
8  118 --      {$ENDC}
8  119 0- A    BEGIN
8  120 --          e := ResError;
8  121 --
8  122 --      {$IFC qDebug}
8  123 --          IF gAskFailure AND (e = noErr) AND CanReadLn THEN
8  124 1-              BEGIN
8  125 --      {$%+}
8  126 --              GetMethodName(%_GetA6+4, s);
8  127 --      {$%-}
8  128 --              e := ReadInteger(CONCAT('FailResError called by ', s, '.  Enter return error: '));
8  129 -1              END;
8  130 --      {$ENDC}
8  131 --
8  132 --          IF e <> noErr THEN
8  133 --              Failure(e, 0);
8  134 -0 A    END {FailResError};
8  135 --
8  136 --
8  137 --      {$S MAMain}
8  138 --      {-----------------------------------+
8  139 --      |   Failure                         |
8  140 --      +-----------------------------------}
8  141 -- A    PROCEDURE Failure (error: INTEGER; message: LONGINT);
8  142 --      VAR
8  143 --          pf:     PFailInfo;
8  144 --          ih:     ProcPtr;
8  145 --          pc:     LONGINT;
8  146 --      {$IFC qDebug}
8  147 --          cl:     String8;
8  148 --          me:     String8;
8  149 --          seg:    INTEGER;
8  150 --          who:    STRING[17];
8  151 --      {$ENDC}
8  152 0- A    BEGIN
8  153 --          pf := gTopHandler;
8  154 --
8  155 --          IF pf <> NIL THEN
8  156 1-              BEGIN
8  157 --      {$IFC qDebug}
8  158 --              pc := pf^.whoPC;
8  159 --              GetProcname(LONGINT(@pc), cl, me);
8  160 --              who := CONCAT(cl, '.', me);
8  161 --              IF cl = kSpace8 THEN
8  162 --                  who[9] := ' ';
8  163 --
8  164 --              Writeln('Failure caught by ', who);
8  165 --              Writeln('       error = ', error:1, ' message = ', message:1,
8  166 --                      ' (', BSR(message, 16):1, '/', BAND(message, $0000FFFF):1, ')');
8  167 --      {$ENDC}
8  168 --
8  169 --              {pop the stack first, because calling the handler is likely to
8  170 --                  result in a call to Failure}
8  171 --              gTopHandler := pf^.nextInfo;
8  172 --              pf^.error := error;
8  173 --              pf^.message := message;
8  174 --              DoFailure(pf);          {Go execute the failure handler}
8  175 -1              END
8  176 --          ELSE IF gInitHandler <> NIL THEN
8  177 1-              BEGIN
8  178 --              ih := gInitHandler;
8  179 --              gInitHandler := NIL;
8  180 --              CallInitHandler(error, message, ih);
8  181 --
8  182 --              ExitToShell;
8  183 -1              END
8  184 --          ELSE
8  185 1-              BEGIN
8  186 --      {$IFC qDebug}
8  187 --              ProgramBreak('Failure called, but no handler!');
8  188 --      {$ENDC}
8  189 -1              END;
8  190 -0 A    END {Failure};
8  191 --
8  192 --
8  193 --      {$S MAInit}
8  194 --      {$IFC qTrace}{$D+}{$ENDC}   {May be called before trace stuff is setup}
8  195 --      {-----------------------------------+
8  196 --      |   InitUFailure                    |
8  197 --      +-----------------------------------}
```

```
8  198 -- A   PROCEDURE InitUFailure;
8  199 --
8  200 0- A   BEGIN
8  201 --         gTopHandler := NIL;
8  202 --         gInitHandler := NIL;
8  203 --
8  204 --     {$IFC qDebug}
8  205 --         gAskAboutAlloc := FALSE;
8  206 --         gAskFailure := FALSE;
8  207 --     {$ENDC}
8  208 --
8  209 -0 A   END {InitUFailure};
8  210 --     {$IFC qTrace}{$D++}{$ENDC}
8  211 --
8  212 --
8  213 --     {$IFC qDebug}
8  214 --     {$IFC qTrace}{$D+}{$ENDC}
8  215 --     {$S MADebug}
8  216 --     {-----------------------------------+
8  217 --     |   ProgramBreak                    |
8  218 --     +-----------------------------------}
8  219 -- A   PROCEDURE ProgramBreak (grievance: Str255);
8  220 --         { ProgramBreak: Your app can call this when it comes to a situation that you do not expect
8  221 --             and cannot handle gracefully.  It beeps and displays a message.  If called before
8  222 --             there is a WriteLn window, it calls OBJFail, which goes into an infinite loop.
8  223 --             Otherwise, it enters our debugger. }
8  224 --     VAR
8  225 --         synthRec:   RECORD
8  226 --                     mode:       INTEGER;
8  227 --                     triplet:    Tone;
8  228 --                     endTriplet: Tone;
8  229 --                     END;
8  230 --
8  231 0- A   BEGIN
8  232 --     {$IFC FALSE}
8  233 --         WITH synthRec, triplet DO
8  234 1-             BEGIN
8  235 --             mode := swMode;
8  236 --
8  237 --             count := 445;
8  238 --             amplitude := 100;
8  239 --             duration := 25;
8  240 --
8  241 --             endTriplet.count := 0;
8  242 --             endTriplet.amplitude := 0;
8  243 --             endTriplet.duration := 0;
8  244 -1             END;
8  245 --
8  246 --         StartSound(@synthRec, SIZEOF(synthRec), Pointer(-1));
8  247 --     {$ENDC}
8  248 --         SysBeep(2);
8  249 --
8  250 --         WWForceOutput(forceOn, forceUnchanged);
8  251 --         WriteLn('ProgramBreak: ', grievance);
8  252 --         WWEndForce;
8  253 --
8  254 --     {$IFC qTrace}
8  255 --         TRCBreak;
8  256 --     {$ELSEC}
8  257 --         OBJFail(kFailNone);
8  258 --     {$ENDC}
8  259 -0 A   END {ProgramBreak};
8  260 --     {$IFC qTrace}{$D++}{$ENDC}
8  261 --
8  262 --
8  263 --     {$IFC qTrace}{$D+}{$ENDC}
8  264 --     {$S MADebug}
8  265 --     {-----------------------------------+
8  266 --     |   ProgramReport                   |
8  267 --     +-----------------------------------}
8  268 -- A   PROCEDURE ProgramReport (grievance: Str255; break: BOOLEAN);
8  269 --
8  270 0- A   BEGIN
8  271 --         Writeln(grievance);
8  272 --         IF break THEN
8  273 --             TRCBreak;
8  274 -0 A   END {ProgramReport};
8  275 --     {$IFC qTrace}{$D++}{$ENDC}
8  276 --     {$ENDC qDebug}
8  277 --
8  278 --
8  279 --     {$S MAInit}
8  280 --     {$IFC qTrace}{$D+}{$ENDC}
8  281 --     {-----------------------------------+
8  282 --     |   SetInitHandler                  |
8  283 --     +-----------------------------------}
8  284 -- A   PROCEDURE SetInitHandler (handler: ProcPtr);
8  285 0- A   BEGIN
8  286 --         gInitHandler := handler;
8  287 -0 A   END {SetInitHandler};
8  288 --     {$IFC qTrace}{$D++}{$ENDC}
8  289 --
8  290 --
8  291 --         {We assume that the programmer passes in the correct FailInfo record; ie. the one that is the
8  292 --             top of the stack.  If debugging is on, we check this fact.}
8  293 --     {$S MAMain}
8  294 --     {-----------------------------------+
8  295 --     |   Success                         |
8  296 --     +-----------------------------------}
```

```
8  297 -- A   PROCEDURE Success (VAR fi: FailInfo);
8  298 0- A   BEGIN
8  299 --         {$IFC qDebug}
8  300 --         IF gTopHandler <> @fi THEN
8  301 1-             BEGIN
8  302 --             Write('gTopHandler = ');
8  303 --             WritePtr(gTopHandler);
8  304 --             Write('parameter = ');
8  305 --             WritePtr(@fi);
8  306 --             WRITELN;
8  307 --             ProgramBreak('Problem with Success: too many or too few calls to Success');
8  308 -1             END;
8  309 --         {$ENDC}
8  310 --
8  311 --         gTopHandler := fi.nextInfo;
8  312 -0 A   END {Success};
8  313 --
7  241 --
7  242 --     END { UFailure }.
```

```
9    1 --    {$P}
9    2 --    {*****************************************************************************}
9    3 --    {*                                                                           *}
9    4 --    {*  File:           UGridView.p                                              *}
9    5 --    {*  Author:         Deb Orton                                                *}
9    6 --    {*  Date:           8 December 1987                                          *}
9    7 --    {*                                                                           *}
9    8 --    {*  Description:    Provide an object for viewing data in a list or grid.    *}
9    9 --    {*                                                                           *}
9   10 --    {*                                                                           *}
9   11 --    {*  Copyright © 1987, 1988 by Apple Computer, Inc.  All rights reserved.     *}
9   12 --    {*****************************************************************************}
9   13 --
9   14 --
9   15 --    {*****************************************************************************}
9   16 --    {*  HouseKeeping                                                             *}
9   17 --    {*****************************************************************************}
9   18 --
9   19 --    UNIT UGridView;
9   20 --
9   21 --    INTERFACE
9   22 --
9   23 --    USES
9   24 --        {$LOAD MacIntf.LOAD}
9   25 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
9   26 --        {$LOAD UMacApp.LOAD}
9   27 --            UMAUtil, UViewCoords, UFailure, UMemory, UMenuSetup, UObject,
9   28 --            UList, UAssociation,  UMacApp;
9   29 --
9   30 --    {*****************************************************************************}
9   31 --    {*  Global Constants                                                         *}
9   32 --    {*****************************************************************************}
9   33 --
9   34 --    CONST
9   35 --
9   36 --        {* Template identifiers for views defined in this unit *}
9   37 --        kGridView       = 'TGridView';
9   38 --        kTextGridView   = 'TTextGridView';
9   39 --        kTextListView   = 'TTextListView';
9   40 --
9   41 --        {* For setting column/row width/height in only one row/col *}
9   42 --        kOneRow = 1;
9   43 --        kOneCol = 1;
9   44 --
9   45 --        {* Select Command Identifiers *}
9   46 --        cCellSelect     =   1;
9   47 --        cRowSelect      =   2;
9   48 --        cColumnSelect   =   3;
9   49 --        cVertexSelect   =   4;
9   50 --
9   51 --        {* Booleans for Adornment *}
9   52 --        kAdorn          = TRUE;
9   53 --        kDontAdorn      = FALSE;
9   54 --
9   55 --        {* Booleans for SetSelection *}
9   56 --        kExtend         = TRUE;
9   57 --        kDontExtend     = FALSE;
9   58 --
9   59 --        kHighlight  = TRUE;
9   60 --        kDontHighlight  = FALSE;
9   61 --
9   62 --        kSelect         = TRUE;
9   63 --        kDeSelect       = FALSE;
9   64 --
9   65 --        {* Booleans for CreateHighlightRgn *}
9   66 --        kWholeRect      = TRUE;
9   67 --        kNotWholeRect   = FALSE;
9   68 --
9   69 --
9   70 --    {*****************************************************************************}
9   71 --    {*  Type and Object Definitions                                              *}
9   72 --    {*****************************************************************************}
9   73 --
9   74 --    TYPE
9   75 --        GridCell        = Point; {* A cell is a QuickDraw point *}
9   76 --        GridViewPart    = (badChoice, inCell, inRow, inColumn, inVertex);
9   77 --        RunArrayChunk   = RECORD
9   78 --                            count:  INTEGER;
9   79 --                            size:   INTEGER;
9   80 --                          END;
9   81 --
9   82 --        RunArray = ARRAY[0..100000] OF RunArrayChunk;
9   83 --        PRunArray = ^RunArray;
9   84 --        HRunArray = ^PRunArray;
9   85 --
9   86 --
9   87 --    {*****************************************************************************}
9   88 --    {*  View Template Types                                                      *}
9   89 --    {*****************************************************************************}
9   90 --
9   91 --
9   92 --        GridViewTemplate =  PACKED RECORD
9   93 --                            numOfRows:          INTEGER;
9   94 --                            numOfCols:          INTEGER;
9   95 --                            rowHeight:          INTEGER;
9   96 --                            colWidth:           INTEGER;
9   97 --                            rowInset:           INTEGER;
9   98 --                            colInset:           INTEGER;
```

```
9   99 --                          adornRows:          BOOLEAN;
9  100 --                          adornCols:          BOOLEAN;
9  101 --                          singleSelection:    BOOLEAN;
9  102 --                          filler:             0..8191;
9  103 --                      END;
9  104 --      GridViewTemplatePtr  =  ^GridViewTemplate;
9  105 --
9  106 --
9  107 --  {*************************************************************************}
9  108 --
9  109 --
9  110 --      TextGridViewTemplate =  PACKED RECORD
9  111 --                          itsFontFace:    Style;
9  112 --                          itsFontSize:    INTEGER;
9  113 --                          itsFontColor:   RGBColor;
9  114 --                          itsFontName:    Str255; {* a variable length P-String *}
9  115 --                      END;
9  116 --      TextGridViewTemplatePtr  =  ^TextGridViewTemplate;
9  117 --
9  118 --
9  119 --
9  120 --  {*************************************************************************}
9  121 --  {*  UGridView Objects                                                    *}
9  122 --  {*************************************************************************}
9  123 --
9  124 --  {*************************************************************************}
9  125 --  {*************************************************************************}
9  126 --  {*  TGridView Object                                                     *}
9  127 --  {*************************************************************************}
9  128 --  {*************************************************************************}
9  129 --
9  130 --      TGridView = OBJECT (TView)
9  131 --
9  132 --      {*************************************************************************}
9  133 --      {*  Instance Variables                                                   *}
9  134 --      {*************************************************************************}
9  135 --
9  136 --          fMaxHorizontal:     LONGINT;    {* horizontal pixels in view          *}
9  137 --          fMaxVertical:       LONGINT;    {* vertical pixels in view            *}
9  138 --          fNumOfCols:         INTEGER;    {* number of columns                  *}
9  139 --          fNumOfRows:         INTEGER;    {* number of rows                     *}
9  140 --          fColWidths:         HRunArray;  {* dynamic array containing col widths *}
9  141 --          fRowHeights:        HRunArray;  {* dynamic array containing row heights *}
9  142 --          fAdornRows:         BOOLEAN;    {* Draw adornment for rows?           *}
9  143 --          fAdornCols:         BOOLEAN;    {* Draw adornment for columns?        *}
9  144 --          fRowInset:          INTEGER;    {* Number of pixels between cell rows  *}
9  145 --          fColInset:          INTEGER;    {* Number of pixels between cell cols  *}
9  146 --          fSelections:        RgnHandle;  {* Region containing current selections *}
9  147 --          fWholeRect:         BOOLEAN;    {* Is the new selection a rectangle?   *}
9  148 --          fLastWholeRect:     BOOLEAN;    {* Is the old selection a rectangle?   *}
9  149 --          fSingleSelection:   BOOLEAN;    {* only one cell selected at a time?   *}
9  150 --          fAllowDimSelection: BOOLEAN;    {* allow the selection to be dimmed?   *}
9  151 --          fHLRegion:          RgnHandle;  {* Region of cells highlighted         *}
9  152 --          fTmpSelRgn:         RgnHandle;  {* Temporary region for selection      *}
9  153 --          fTmpRgn:            RgnHandle;  {* Temporary region for on-the-fly use  *}
9  154 --          fTmpHLRgn:          RgnHandle;  {* Temporary region for highlighting   *}
9  155 --          fNumOfRowChunks:    INTEGER;    {* # of Entries in the Row run array   *}
9  156 --          fNumOfColChunks:    INTEGER;    {* # of Entries in the Column run array *}
9  157 --          fLastRow:           INTEGER;    {* These 4 fields used for run array   *}
9  158 --          fLastRowChunkNum:   INTEGER;    {* cacheing for rows...               *}
9  159 --          fLastRowTotal:      LONGINT;
9  160 --          fLastRowIndex:      INTEGER;
9  161 --          fLastCol:           INTEGER;    {* These 4 fields used for run array   *}
9  162 --          fLastColChunkNum:   INTEGER;    {* cacheing for columns...            *}
9  163 --          fLastColTotal:      LONGINT;
9  164 --          fLastColIndex:      INTEGER;
9  165 --
9  166 --      {*************************************************************************}
9  167 --      {*  Initialization and Free Methods                                      *}
9  168 --      {*************************************************************************}
9  169 --
9  170 --  {$IFC qProceduralViews}
9  171 --          PROCEDURE TGridView.IGridView (
9  172 --              itsDocument:        TDocument;  {* Its document                 *}
9  173 --              itsSuperView:       TView;      {* Its parent view              *}
9  174 --              itsLocation:        VPoint;     {* Top, Left in parent's coords  *}
9  175 --              numOfRows:          INTEGER;    {* number of rows initially     *}
9  176 --              numOfCols:          INTEGER;    {* number of columns initially  *}
9  177 --              rowHeight:          INTEGER;    {* Height for initial rows      *}
9  178 --              colWidth:           INTEGER;    {* width for initial columns    *}
9  179 --              adornRows:          BOOLEAN;    {* Adornment for Rows?          *}
9  180 --              adornCols:          BOOLEAN;    {* Adornment for Columns?       *}
9  181 --              rowInset:           INTEGER;    {* horizontal space between cells  *}
9  182 --              colInset:           INTEGER;    {* vertical   space between cells  *}
9  183 --              singleSelection:    BOOLEAN);   {* single cell selection?       *}
9  184 --
9  185 --              { Initialize the gridview.  If numOfRows or numOfCols is non-zero
9  186 --                  then the gridview is initialized to the specified size using
9  187 --                  rowHeight and colWIdth.  If either adorn is kDontAdorn, then that
9  188 --                  adorn will not be called.  The "Insets" allow space between
9  189 --                  cells and may be used to draw row and column adornments. }
9  190 --  {$ENDC qProceduralViews}
9  191 --
9  192 --  {$IFC qTemplateViews}
9  193 --          PROCEDURE TGridView.IRes (itsDocument: TDocument; itsSuperView: TView;
9  194 --              VAR itsParams: Ptr); OVERRIDE;
9  195 --              { Initialize the view template. }
9  196 --  {$ENDC qTemplateViews}
9  197 --
```

```
9  198 --     {$IFC qWriteTemplates}
9  199 --             PROCEDURE TGridView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
9  200 --                 { Write this object out as a view resource. }
9  201 --
9  202 --             PROCEDURE TGridView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
9  203 --                 { Set up the type and signature of this object and call WRes. }
9  204 --     {$ENDC qWriteTemplates}
9  205 --
9  206 --             PROCEDURE TGridView.Free; OVERRIDE;
9  207 --                 { Free the TGridView object }
9  208 --
9  209 --         {*****************************************************************************}
9  210 --         {*  Methods to OVERRIDE                                                      *}
9  211 --         {*****************************************************************************}
9  212 --
9  213 --             {*****************************************************************************}
9  214 --             {*  Inherited Methods  (OVERRIDE)                                        *}
9  215 --             {*****************************************************************************}
9  216 --
9  217 --             PROCEDURE TGridView.CalcMinSize (VAR minSize: VPoint); OVERRIDE;
9  218 --                 { Sets the extent of the view. }
9  219 --
9  220 --             PROCEDURE TGridView.DoHighlightSelection (fromHL, toHL: HLState); OVERRIDE;
9  221 --                 { Do highlighting }
9  222 --
9  223 --             FUNCTION  TGridView.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
9  224 --                     VAR hysteresis: Point) : TCommand; OVERRIDE;
9  225 --                 { Handle mouse commands }
9  226 --
9  227 --             PROCEDURE TGridView.Draw (area: Rect); OVERRIDE;
9  228 --                 { Calls DrawRangeOfCells and AdornRow/Col as appropriate. }
9  229 --
9  230 --
9  231 --             {*****************************************************************************}
9  232 --             {*  UGridView Methods to OVERRIDE                                        *}
9  233 --             {*****************************************************************************}
9  234 --
9  235 --             PROCEDURE TGridView.AdornCol (aCol: INTEGER; area: Rect);
9  236 --             PROCEDURE TGridView.AdornRow (aRow: INTEGER; area: Rect);
9  237 --                 { If fAdornXxx is TRUE, these methods are called to draw the row and
9  238 --                   column adornments. }
9  239 --
9  240 --             FUNCTION  TGridView.CanSelectCell (aCell: GridCell) : BOOLEAN;
9  241 --                 { This method checks to see if a cell can be seleced.  By default,
9  242 --                   this method always returns TRUE. }
9  243 --
9  244 --             FUNCTION  TGridView.CellExists (aCell: GridCell) : BOOLEAN;
9  245 --                 { By default this method always returns TRUE.  Cells that do not
9  246 --                   exists will be skipped when drawing is done. }
9  247 --
9  248 --             PROCEDURE TGridView.DoHilite (hlRgn: RgnHandle; fromHL, toHL: HLState);
9  249 --                 { This guys actually does the highlighting (inverts the rect).
9  250 --                   Override it to get different highlighting behavior }
9  251 --
9  252 --             PROCEDURE TGridView.DrawRangeOfCells (startCell, stopCell: GridCell; aQDRect: Rect);
9  253 --                 { This method calls DrawCell for each cell in need of re-drawing. }
9  254 --
9  255 --             PROCEDURE TGridView.DrawCell (aCell: GridCell; aQDRect: Rect);
9  256 --                 { This method draws each individual cell.  IT MUST BE OVERRIDDEN! }
9  257 --
9  258 --             {*****************************************************************************}
9  259 --             {*  General UGridView Methods                                            *}
9  260 --             {*****************************************************************************}
9  261 --
9  262 --             PROCEDURE TGridVIew.AllCellsDo (PROCEDURE DoToCell (aCell: GridCell));
9  263 --                 { For every cell in the view perform "DoToCell". }
9  264 --
9  265 --             PROCEDURE TGridView.CellToVRect (aCell: GridCell; VAR aRect: VRect);
9  266 --                 { Get the rectangle bounding a given cell.  This includes the row
9  267 --                   and column insets. }
9  268 --
9  269 --             PROCEDURE TGridView.ColToVRect (aCol: INTEGER; numOfCols: INTEGER; VAR aRect: VRect);
9  270 --             PROCEDURE TGridView.RowToVRect (aRow: INTEGER; numOfRows: INTEGER; VAR aRect: VRect);
9  271 --                 { Get the rectangle bounding the given rows or columns.  This includes
9  272 --                   the cells in the row or column and the row or column insets. }
9  273 --
9  274 --             PROCEDURE TGridView.CreateHighlightRgn (oldRgn, newRgn: RgnHandle;
9  275 --                                                     wholeRect: BOOLEAN);
9  276 --                 { This guy goes through all the cells in the oldRgn and creates a
9  277 --                   newRgn to be used for highlighting. }
9  278 --
9  279 --             PROCEDURE TGridView.DelColAt (aCol: INTEGER; numOfCols: INTEGER);
9  280 --             PROCEDURE TGridView.DelRowAt (aRow: INTEGER; numOfRows: INTEGER);
9  281 --                 { Delete the specified columns.  This causes a redraw of only the
9  282 --                   necessary cells. }
9  283 --
9  284 --             PROCEDURE TGridView.DelColFirst (numOfCols: INTEGER);
9  285 --             PROCEDURE TGridView.DelRowFirst (numOfRows: INTEGER);
9  286 --                 { Delete the specified columns.  This causes a redraw of only the
9  287 --                   necessary cells. }
9  288 --
9  289 --             PROCEDURE TGridView.DelColLast (numOfCols: INTEGER);
9  290 --             PROCEDURE TGridView.DelRowLast (numOfRows: INTEGER);
9  291 --                 { Delete the specified columns.  This causes a redraw of only the
9  292 --                   necessary cells. }
9  293 --
9  294 --             PROCEDURE TGridView.EachCellDo (startCell, stopCell: GridCell;
9  295 --                             PROCEDURE DoToCell (aCell: GridCell));
9  296 --                 { For each cell in the rectangle defined by startCell and stopCell
```

```
9  297 --                    perform "DoToCell". }
9  298 --
9  299 --            PROCEDURE TGridView.EachSelectedCellDo (PROCEDURE DoToCell (aCell: GridCell));
9  300 --                { For each cell in the selected region perform "DoToCell". }
9  301 --
9  302 --            PROCEDURE TGridView.EachInRgn (aRgn: RgnHandle;
9  303 --                            PROCEDURE DoToCell (aCell: GridCell));
9  304 --                { For each cell in the specified region perform "DoToCell". }
9  305 --
9  306 --            FUNCTION  TGridView.FindColChunk (aCol: INTEGER; VAR aChunkNum: INTEGER;
9  307 --                    VAR theTotal: LONGINT; VAR indexInChunk: INTEGER): BOOLEAN;
9  308 --
9  309 --            FUNCTION  TGridView.FindRowChunk (aRow: INTEGER; VAR aChunkNum: INTEGER;
9  310 --                    VAR theTotal: LONGINT; VAR indexInChunk: INTEGER): BOOLEAN;
9  311 --
9  312 --            FUNCTION  TGridView.FirstSelectedCell: GridCell;
9  313 --                { Returns the top left cell in the selection range, if any }
9  314 --
9  315 --            FUNCTION  TGridView.GetColWidth (aCol: INTEGER) : INTEGER;
9  316 --            FUNCTION  TGridView.GetRowHeight (aRow: INTEGER) : INTEGER;
9  317 --                { Return the width or height for the specified row or column. }
9  318 --
9  319 --            FUNCTION TGridView.IdentifyPoint (theQDPoint: Point;
9  320 --                            VAR aRow, aCol: INTEGER): GridViewPart;
9  321 --                { Return the GridViewPart (inCell, inColumn, inRow, inVertex)
9  322 --                    in which the specified point lies.  The "vertex" is the area
9  323 --                    where a row and column meet. }
9  324 --
9  325 --            PROCEDURE TGridView.InsColBefore (aCol: INTEGER; numOfCols: INTEGER; aWidth: INTEGER);
9  326 --            PROCEDURE TGridView.InsRowBefore (aRow: INTEGER; numOfRows: INTEGER; aHeight: INTEGER);
9  327 --                { Insert the specified rows or columns before the row/column given.
9  328 --                    The heights/widths of the rows/columns are all set to the specified
9  329 --                    Height/Width. }
9  330 --
9  331 --            PROCEDURE TGridView.InsColFirst (numOfCols: INTEGER; aWidth: INTEGER);
9  332 --            PROCEDURE TGridView.InsRowFirst (numOfRows: INTEGER; aHeight: INTEGER);
9  333 --                { Insert the specified rows or columns before the row/column given.
9  334 --                    The heights/widths of the rows/columns are all set to the specified
9  335 --                    Height/Width. }
9  336 --
9  337 --            PROCEDURE TGridView.InsColLast (numOfCols: INTEGER; aWidth: INTEGER);
9  338 --            PROCEDURE TGridView.InsRowLast (numOfRows: INTEGER; aHeight: INTEGER);
9  339 --                { Insert the specified rows or columns before the row/column given.
9  340 --                    The heights/widths of the rows/columns are all set to the specified
9  341 --                    Height/Width. }
9  342 --
9  343 --            PROCEDURE TGridView.InvalidateCell (aCell: GridCell);
9  344 --                { Cause a cell to be marked invalid (in need of re-drawing). }
9  345 --
9  346 --            PROCEDURE TGridView.InvalidateSelection;
9  347 --                { Cause the rectangle bounding the selections to be marked
9  348 --                    invalid (in need of re-drawing). }
9  349 --
9  350 --            FUNCTION  TGridView.IsCellSelected (aCell: GridCell) : BOOLEAN;
9  351 --                { Check if the specified cell is currently selected. }
9  352 --
9  353 --            FUNCTION  TGridView.LastSelectedCell: GridCell;
9  354 --                { Returns the bottom right cell in the selection range, if any }
9  355 --
9  356 --            FUNCTION TGridView.SameCell (aCell, bCell: GridCell): BOOLEAN;
9  357 --                { Check if the cells are the same }
9  358 --
9  359 --            PROCEDURE TGridView.ScrollSelectionIntoView (redraw: BOOLEAN);
9  360 --                { Scroll the specified rectangle into view. }
9  361 --
9  362 --            PROCEDURE TGridView.SetColWidth (aCol: INTEGER; numOfCols: INTEGER; aWidth: INTEGER);
9  363 --            PROCEDURE TGridView.SetRowHeight (aRow: INTEGER; numOfRows: INTEGER; aHeight: INTEGER);
9  364 --                { Set the width/height of the specified rows/columns to that given. }
9  365 --
9  366 --            PROCEDURE TGridView.SelectCell (theCell: GridCell; extend, highlight, select: BOOLEAN);
9  367 --                { Set the current selection to the specified cell.  This guy sets up a
9  368 --                    region and then calls SetSelection. }
9  369 --
9  370 --            PROCEDURE TGridView.SetEmptySelection (highlight: BOOLEAN);
9  371 --                { Set the current selection to be empty or nothing.  If highlight
9  372 --                    is kHighlight then the appropriate highlighting is performed. }
9  373 --
9  374 --            PROCEDURE TGridView.SetSelection (theRegion: RgnHandle; extend, highlight, select: BOOLEAN);
9  375 --                { Set the current selections to the specified region.  If extend is kExtend
9  376 --                    then theRegion is "added" to that already selected, if highlight is
9  377 --                    kHighlight then theRegion is highlighted as well.  IF select is kSelect then
9  378 --                    the region is selected, if kDeSelect then de-selected}
9  379 --
9  380 --            PROCEDURE TGridView.SetSelRect (theTop, theLeft, theBottom, theRight: INTEGER;
9  381 --                            extend, highlight, select: BOOLEAN);
9  382 --                { Set the current selections to the specified rectangle.  and
9  383 --                    call SetSelection. }
9  384 --
9  385 --            PROCEDURE TGridView.SetSingleSelection (theSetting: BOOLEAN);
9  386 --                { If TRUE then only one item/cell can be selected at a time }
9  387 --
9  388 --            FUNCTION  TGridView.VPointToCell (aPoint: VPoint): GridCell;
9  389 --                { Determine the cell in which a given point lies. }
9  390 --
9  391 --        {*****************************************************************************}
9  392 --        {*  Debugging Methods                                                        *}
9  393 --        {*****************************************************************************}
9  394 --
9  395 --    {$IFC qDebug}
```

```
9  396 --              PROCEDURE TGridView.Fields (
9  397 --                      PROCEDURE DoToField (
9  398 --                          fieldName: Str255;
9  399 --                          fieldAddr: Ptr;
9  400 --                          fieldType: INTEGER)); OVERRIDE;
9  401 --      {$ENDC}
9  402 --
9  403 --          END;  {* TGridView *}
9  404 --
9  405 --
9  406 --      {*************************************************************************}
9  407 --      {*************************************************************************}
9  408 --      {*  TTextGridView Object Sub-Class                                       *}
9  409 --      {*************************************************************************}
9  410 --      {*************************************************************************}
9  411 --
9  412 --          TTextGridView = OBJECT (TGridView)
9  413 --
9  414 --          {*************************************************************************}
9  415 --          {*  Instance Variables                                                   *}
9  416 --          {*************************************************************************}
9  417 --
9  418 --              fTextStyle:     TextStyle;  {* The text style (color, size, etc. )      *}
9  419 --              fLineHeight:    INTEGER;    {* height of each item including leading    *}
9  420 --              fLineAscent:    INTEGER;    {* pos. of baseline relative to top of line *}
9  421 --
9  422 --          {*************************************************************************}
9  423 --          {*  Initialization and Free Methods                                      *}
9  424 --          {*************************************************************************}
9  425 --
9  426 --      {$IFC qProceduralViews}
9  427 --              PROCEDURE TTextGridView.ITextGridView (
9  428 --                  itsDocument:        TDocument;  {* Its document                    *}
9  429 --                  itsSuperView:       TView;      {* Its parent view                 *}
9  430 --                  itsLocation:        VPoint;     {* Top, Left in parent's coords    *}
9  431 --                  numOfRows:          INTEGER;    {* Number of rows initially        *}
9  432 --                  numOfCols:          INTEGER;    {* Number of columns initially     *}
9  433 --                  ColWidth:           INTEGER;    {* Width of items in the columns   *}
9  434 --                  adornRows:          BOOLEAN;    {* Adornment for Rows?             *}
9  435 --                  adornCols:          BOOLEAN;    {* Adornment for Columns?          *}
9  436 --                  rowInset:           INTEGER;    {* horizontal space between cells  *}
9  437 --                  colInset:           INTEGER;    {* vertical   space between cells  *}
9  438 --                  singleSelection:    BOOLEAN;    {* single cell selection?          *}
9  439 --                  itsTextStyle:       TextStyle); {* size, color, etc. font info     *}
9  440 --
9  441 --                  { Initialize the TextGridView.  If numOfRows or numOfCols is non-zero
9  442 --                      then the gridview is initialized to the specified size.  The row
9  443 --                      and height of the cells is determine by the font size, style etc.
9  444 --                      The default size may be changed with calls to the the SetRowHeight
9  445 --                      and SetColWidth methods.  If either adorn is kDontAdorn, then that
9  446 --                      adorn will not be called.  The "Insets" allow space between
9  447 --                      cells and may be used to draw row and column adornments. }
9  448 --      {$ENDC qProceduralViews}
9  449 --
9  450 --      {$IFC qTemplateViews}
9  451 --              PROCEDURE TTextGridView.IRes (itsDocument: TDocument; itsSuperView: TView;
9  452 --                  VAR itsParams: Ptr); OVERRIDE;
9  453 --                  { Initialize the view template. }
9  454 --      {$ENDC qTemplateViews}
9  455 --
9  456 --      {$IFC qWriteTemplates}
9  457 --              PROCEDURE TTextGridView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
9  458 --                  { Write this object out as a view resource. }
9  459 --
9  460 --              PROCEDURE TTextGridView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
9  461 --                  { Set up the type and signature of this object and call WRes. }
9  462 --      {$ENDC qWriteTemplates}
9  463 --
9  464 --
9  465 --          {*************************************************************************}
9  466 --          {*  Methods To OVERRIDE                                                  *}
9  467 --          {*************************************************************************}
9  468 --
9  469 --              PROCEDURE TTextGridView.GetText (aCell: GridCell; VAR aString: Str255);
9  470 --                  { This routine gets the text to be draw in a given cell.  IT MUST
9  471 --                      BE OVERRIDDEN! }
9  472 --
9  473 --
9  474 --          {*************************************************************************}
9  475 --          {*  General Methods                                                      *}
9  476 --          {*************************************************************************}
9  477 --
9  478 --
9  479 --              PROCEDURE TTextGridView.DrawCell (aCell: GridCell; aQDRect: Rect); OVERRIDE;
9  480 --                  { Calls GetText and then draws the text }
9  481 --
9  482 --              FUNCTION  TTextGridView.Focus : BOOLEAN; OVERRIDE;
9  483 --                  { Calls inherited focus and calls SetUpFont }
9  484 --
9  485 --              PROCEDURE TTextGridView.SetUpFont;
9  486 --                  {Sets up the font for this view. }
9  487 --
9  488 --              PROCEDURE TTextGridView.SetPen;
9  489 --                  {Sets the font characteristics of the current port.  Called at
9  490 --                      the beginning of the Draw method.}
9  491 --
9  492 --      {$IFC qDebug}
9  493 --              PROCEDURE TTextGridView.Fields (
9  494 --                      PROCEDURE DoToField (
```

```
9  495 --                          fieldName: Str255;
9  496 --                          fieldAddr: Ptr;
9  497 --                          fieldType: INTEGER)); OVERRIDE;
9  498 --      {$ENDC}
9  499 --
9  500 --          END; {* TTextGridView *}
9  501 --
9  502 --
9  503 --      {*****************************************************************************}
9  504 --      {*****************************************************************************}
9  505 --      {*  TTextListView Object Sub-Class                                           *}
9  506 --      {*****************************************************************************}
9  507 --      {*****************************************************************************}
9  508 --
9  509 --          TTextListView = OBJECT (TTextGridView)
9  510 --
9  511 --          {*****************************************************************************}
9  512 --          {*  Instance Variables                                                       *}
9  513 --          {*****************************************************************************}
9  514 --
9  515 --                  {* none *}
9  516 --
9  517 --          {*****************************************************************************}
9  518 --          {*  Initialization and Free Methods                                          *}
9  519 --          {*****************************************************************************}
9  520 --
9  521 --      {$IFC qProceduralViews}
9  522 --              PROCEDURE TTextListView.ITextListView (
9  523 --                  itsDocument:    TDocument;      {* Its document                  *}
9  524 --                  itsSuperView:   TView;          {* Its parent view               *}
9  525 --                  itsLocation:    VPoint;         {* Top, Left in parent's coords  *}
9  526 --                  numOfItems:     INTEGER;        {* Number of items initially     *}
9  527 --                  adornRows:      BOOLEAN;        {* Draw the row adornments?      *}
9  528 --                  adornCols:      BOOLEAN;        {* Draw the col adornment?       *}
9  529 --                  rowInset:       INTEGER;        {* Amount to inset the rows      *}
9  530 --                  colInset:       INTEGER;        {* Amount to inset the column    *}
9  531 --                  singleSelection:BOOLEAN;        {* single cell selection?        *}
9  532 --                  itsTextStyle:   TextStyle);     {* size, color, etc. font info   *}
9  533 --
9  534 --                  { Initialize the TextListView.  If numOfItems is non-zero
9  535 --                      then the listview is initialized to the specified size.  The width
9  536 --                      and height of the items are determined by the font size, style etc.
9  537 --                      The default size may be changed with calls to the the SetItemHeight
9  538 --                      and SetItemWidth methods.  The "Insets" allow space between items. }
9  539 --      {$ENDC qProceduralViews}
9  540 --
9  541 --      {$IFC qWriteTemplates}
9  542 --              PROCEDURE TTextListView.WriteRes (theResource: ViewRsrcHndl;
9  543 --                                                VAR itsParams: Ptr); OVERRIDE;
9  544 --                  { Set up the type and signature of this object and call WRes. }
9  545 --      {$ENDC qWriteTemplates}
9  546 --
9  547 --
9  548 --          {*****************************************************************************}
9  549 --          {*  Methods To OVERRIDE                                                      *}
9  550 --          {*****************************************************************************}
9  551 --
9  552 --              PROCEDURE TTextListView.GetItemText (anItem: INTEGER; VAR aString: Str255);
9  553 --                  { Get the text to be drawn for a given item.  THIS METHOD MUST BE
9  554 --                      OVERRIDDEN !! }
9  555 --
9  556 --              FUNCTION  TTextListView.CanSelectItem (anItem: INTEGER) : BOOLEAN;
9  557 --                  { Determine if the item is select-able }
9  558 --
9  559 --          {*****************************************************************************}
9  560 --          {*  General Methods                                                          *}
9  561 --          {*****************************************************************************}
9  562 --              PROCEDURE TTextListView.AllItemsDo (PROCEDURE DoToItem (anItem: INTEGER));
9  563 --                  { For all the items in the view perform "DoToItem". }
9  564 --
9  565 --              FUNCTION  TTextListView.CanSelectCell (aCell: GridCell) : BOOLEAN; OVERRIDE;
9  566 --                  { This method checks to see if a cell can be seleced.  This method
9  567 --                      simply calls CanSelectItem }
9  568 --
9  569 --              PROCEDURE TTextListView.DelItemAt (anItem: INTEGER; numOfItems: INTEGER);
9  570 --              PROCEDURE TTextListView.DelItemFirst (numOfItems: INTEGER);
9  571 --              PROCEDURE TTextListView.DelItemLast (numOfItems: INTEGER);
9  572 --                  { Delete the specified items.  This causes a redraw of only the
9  573 --                      necessary cells. }
9  574 --
9  575 --              PROCEDURE TTextListView.EachItemDo (start, stop: INTEGER;
9  576 --                                                  PROCEDURE DoToItem (anItem: INTEGER));
9  577 --                  { For each item in the specified range perform "DoToItem". }
9  578 --
9  579 --              PROCEDURE TTextListView.EachSelectedItemDo (PROCEDURE DoToItem (anItem: INTEGER));
9  580 --                  { For each item selected perform "DoToItem". }
9  581 --
9  582 --              FUNCTION  TTextListView.GetItemHeight (anItem: INTEGER) : INTEGER;
9  583 --                  { Return the height of the specified item. }
9  584 --
9  585 --              FUNCTION  TTextListView.GetItemWidth: INTEGER;
9  586 --                  { Return the width of the view. }
9  587 --
9  588 --              PROCEDURE TTextListView.GetText (aCell: GridCell; VAR aString: Str255); OVERRIDE;
9  589 --                  { Calls GetItemText with an item number. }
9  590 --
9  591 --              PROCEDURE TTextListView.InsItemBefore (anItem: INTEGER; numOfItems: INTEGER);
9  592 --              PROCEDURE TTextListView.InsItemFirst (numOfItems: INTEGER);
9  593 --              PROCEDURE TTextListView.InsItemLast (numOfItems: INTEGER);
```

```
9  594 --                { Insert the specified items.  This causes a redraw of only the
9  595 --                    necessary cells. }
9  596 --
9  597 --            FUNCTION  TTextListView.IsItemSelected (anItem: INTEGER) : BOOLEAN;
9  598 --                { Check if the specified item is currently selected. }
9  599 --
9  600 --            PROCEDURE TTextListView.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
9  601 --                { Resize the view. }
9  602 --
9  603 --            PROCEDURE TTextListView.SelectCell (theCell: GridCell; extend, highlight, select: BOOLEAN); OVERRIDE;
9  604 --                { Calls SelectItem with the appropriate item number }
9  605 --
9  606 --            PROCEDURE TTextListView.SelectItem (anItem: INTEGER; extend, highlight, select: BOOLEAN);
9  607 --                { Select the given item.  If extend is true then the item is added to the
9  608 --                    current selection.  Otherwise the current selection is deselected.
9  609 --                    If highlight is true then the selected item is highlighted.}
9  610 --
9  611 --            PROCEDURE TTextListView.SetItemHeight (anItem: INTEGER; numOfItems: INTEGER;
9  612 --                                aHeight: INTEGER);
9  613 --                { Set the height of an item to be different from the default.  This
9  614 --                    changes the height of only the specified items. }
9  615 --
9  616 --            PROCEDURE TTextListView.SetItemWidth (aWidth: INTEGER);
9  617 --                { Set the width of the view to aWidth.  (This changes the width of all
9  618 --                    the items. }
9  619 --
9  620 --        {$IFC qDebug}
9  621 --                PROCEDURE TTextListView.Fields (
9  622 --                        PROCEDURE DoToField (
9  623 --                            fieldName: Str255;
9  624 --                            fieldAddr: Ptr;
9  625 --                            fieldType: INTEGER)); OVERRIDE;
9  626 --        {$ENDC}
9  627 --
9  628 --            END;    {* TTextListView *}
9  629 --
9  630 --
9  631 --
9  632 --        {*****************************************************************************}
9  633 --        {*  Support Objects                                                          *}
9  634 --        {*****************************************************************************}
9  635 --
9  636 --        {*****************************************************************************}
9  637 --        {*  TGridSelectCommand                                                       *}
9  638 --        {*****************************************************************************}
9  639 --        TGridSelectCommand = OBJECT (TCommand)
9  640 --            fGridView:          TGridView;      {* The associated GridView  *}
9  641 --            fShiftKey:          BOOLEAN;        {* Shift Key down?          *}
9  642 --            fCmdKey:            BOOLEAN;        {* Command Key down?        *}
9  643 --            fOldRgn:            RgnHandle;      {* region of old selected cells *}
9  644 --            fNewRgn:            RgnHandle;      {* region of new selected cells *}
9  645 --
9  646 --            PROCEDURE TGridSelectCommand.IGridSelectCommand (itsCmdNumber: CmdNumber;
9  647 --                            itsView: TGridView; theShiftKey, theCmdKey: BOOLEAN);
9  648 --                { Initialize the command object. }
9  649 --
9  650 --            PROCEDURE TGridSelectCommand.Free; OVERRIDE;
9  651 --                { Free the command object. }
9  652 --
9  653 --            PROCEDURE TGridSelectCommand.TrackFeedback (anchorPoint, nextPoint: VPoint;
9  654 --                            turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
9  655 --                { Track the mouse while the button is down. }
9  656 --
9  657 --        {$IFC qDebug}
9  658 --            PROCEDURE TGridSelectCommand.Fields (
9  659 --                        PROCEDURE DoToField (
9  660 --                            fieldName: Str255;
9  661 --                            fieldAddr: Ptr;
9  662 --                            fieldType: INTEGER)); OVERRIDE;
9  663 --        {$ENDC}
9  664 --
9  665 --        END;    {* TGridSelectCommand *}
9  666 --
9  667 --
9  668 --        {*****************************************************************************}
9  669 --        {*  TCellSelectCommand                                                       *}
9  670 --        {*****************************************************************************}
9  671 --
9  672 --        TCellSelectCommand = OBJECT (TGridSelectCommand)
9  673 --            fOldCell:           GridCell;       {* The last cell selected   *}
9  674 --            fNewMouseDown:      BOOLEAN;        {* determine if this is a new mouse down *}
9  675 --            fKeepSelecting:     BOOLEAN;        {* select all cells? (Cmd Key) *}
9  676 --
9  677 --            PROCEDURE TCellSelectCommand.ICellSelectCommand (itsView: TGridView;
9  678 --                            aCell: GridCell; theShiftKey, theCmdKey: BOOLEAN);
9  679 --                { Initialize the command object. }
9  680 --
9  681 --            PROCEDURE TCellSelectCommand.HandleShiftKeyDown (aView: TGridView;
9  682 --                                    aCell, anchorCell: GridCell; mouseDidMove: BOOLEAN);
9  683 --                { Provides support for TrackFeedback and TrackMouse when the shift
9  684 --                    key is down with the mouse button. }
9  685 --
9  686 --            PROCEDURE TCellSelectCommand.TrackFeedback (anchorPoint, nextPoint: VPoint;
9  687 --                            turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
9  688 --                { Track the mouse while the button is down. }
9  689 --
9  690 --            FUNCTION TCellSelectCommand.TrackMouse (aTrackPhase: TrackPhase;
9  691 --                                    VAR anchorPoint, previousPoint, nextPoint: VPoint;
9  692 --                                    mouseDidMove: BOOLEAN): TCommand; OVERRIDE;
```

```
9  693 --              { Track the mouse when the button is up. }
9  694 --
9  695 --      {$IFC qDebug}
9  696 --          PROCEDURE TCellSelectCommand.Fields (
9  697 --                      PROCEDURE DoToField (
9  698 --                          fieldName: Str255;
9  699 --                          fieldAddr: Ptr;
9  700 --                          fieldType: INTEGER)); OVERRIDE;
9  701 --      {$ENDC}
9  702 --
9  703 --
9  704 --      END;    {* TCellSelectCommand *}
9  705 --
9  706 --
9  707 --      {*****************************************************************************}
9  708 --      {*  TRowSelectCommand                                                        *}
9  709 --      {*****************************************************************************}
9  710 --
9  711 --      TRowSelectCommand = OBJECT (TGridSelectCommand)
9  712 --          fTheRow:        INTEGER;    {* The row selected         *}
9  713 --
9  714 --          PROCEDURE TRowSelectCommand.IRowSelectCommand (itsView: TGridView;
9  715 --                      aRow: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
9  716 --              { Initialize the command object. }
9  717 --
9  718 --      {$IFC qDebug}
9  719 --          PROCEDURE TRowSelectCommand.Fields (
9  720 --                      PROCEDURE DoToField (
9  721 --                          fieldName: Str255;
9  722 --                          fieldAddr: Ptr;
9  723 --                          fieldType: INTEGER)); OVERRIDE;
9  724 --      {$ENDC}
9  725 --
9  726 --
9  727 --      END;    {* TRowSelectCommand *}
9  728 --
9  729 --
9  730 --      {*****************************************************************************}
9  731 --      {*  TColSelectCommand                                                        *}
9  732 --      {*****************************************************************************}
9  733 --
9  734 --      TColSelectCommand = OBJECT (TGridSelectCommand)
9  735 --          fTheColumn:     INTEGER;    {* The column selected      *}
9  736 --
9  737 --          PROCEDURE TColSelectCommand.IColSelectCommand (itsView: TGridView;
9  738 --                      aColumn: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
9  739 --              { Initialize the command object. }
9  740 --
9  741 --      {$IFC qDebug}
9  742 --          PROCEDURE TColSelectCommand.Fields (
9  743 --                      PROCEDURE DoToField (
9  744 --                          fieldName: Str255;
9  745 --                          fieldAddr: Ptr;
9  746 --                          fieldType: INTEGER)); OVERRIDE;
9  747 --      {$ENDC}
9  748 --
9  749 --      END; {* TColSelectCommand *}
9  750 --
9  751 --
9  752 --      {*****************************************************************************}
9  753 --      {*  TVertexSelectCommand                                                     *}
9  754 --      {*****************************************************************************}
9  755 --
9  756 --      TVertexSelectCommand = OBJECT (TGridSelectCommand)
9  757 --          fTheColumn:     INTEGER;    {* The column selected      *}
9  758 --          fTheRow:        INTEGER;    {* The column selected      *}
9  759 --
9  760 --          PROCEDURE TVertexSelectCommand.IVertexSelectCommand (itsView: TGridView;
9  761 --                      aRow, aColumn: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
9  762 --              { Initialize the command object. }
9  763 --
9  764 --      {$IFC qDebug}
9  765 --          PROCEDURE TVertexSelectCommand.Fields (
9  766 --                      PROCEDURE DoToField (
9  767 --                          fieldName: Str255;
9  768 --                          fieldAddr: Ptr;
9  769 --                          fieldType: INTEGER)); OVERRIDE;
9  770 --      {$ENDC}
9  771 --
9  772 --      END; {* TVertexSelectCommand *}
9  773 --
9  774 --
9  775 --      {*****************************************************************************}
9  776 --      {*  Implementation                                                           *}
9  777 --      {*****************************************************************************}
9  778 --
9  779 --      IMPLEMENTATION
9  780 --
9  781 --      {$I UGridView.incl.p}
```

```
10    1 --      {$P}
10    2 --      {*************************************************************************}
10    3 --      {*                                                                       *}
10    4 --      {*  File:           UGridView.incl.p                                     *}
10    5 --      {*  Author:         Deb Orton                                            *}
10    6 --      {*  Date:           8 December 1987                                      *}
10    7 --      {*                                                                       *}
10    8 --      {*                                                                       *}
10    9 --      {*  Copyright © 1987, 1988 by Apple Computer, Inc.  All rights reserved.  *}
10   10 --      {*************************************************************************}
10   11 --
10   12 --
10   13 --      {*************************************************************************}
10   14 --      {*************************************************************************}
10   15 --      {*  TGridView Object                                                     *}
10   16 --      {*************************************************************************}
10   17 --      {*************************************************************************}
10   18 --
10   19 --      {*************************************************************************}
10   20 --      {*  TGridView.IGridView                                                  *}
10   21 --      {*      Initialize the gridview.  If numOfRows or numOfCols is non-zero  *}
10   22 --      {*      then the gridview is initialized to the specified size using     *}
10   23 --      {*      RowHeight and ColWIdth.  If either adorn is false, then that     *}
10   24 --      {*      adorn method will not be called.  The "Insets" allow space between *}
10   25 --      {*      cells and may be used to draw row and column adornments.         *}
10   26 --      {*************************************************************************}
10   27 --      {$IFC qProceduralViews}
10   28 --      {$S GVOpen}
10   29 -- A    PROCEDURE TGridView.IGridView (
10   30 --          itsDocument:        TDocument;  {* Its document                  *}
10   31 --          itsSuperView:       TView;      {* Its parent view               *}
10   32 --          itsLocation:        VPoint;     {* Top, Left in parent's coords  *}
10   33 --          numOfRows:          INTEGER;    {* Number of rows initially      *}
10   34 --          numOfCols:          INTEGER;    {* Number of columns initially   *}
10   35 --          rowHeight:          INTEGER;    {* Height of initial rows        *}
10   36 --          colWidth:           INTEGER;    {* Height of initial columns     *}
10   37 --          adornRows:          BOOLEAN;    {* Adornment for Rows?           *}
10   38 --          adornCols:          BOOLEAN;    {* Adornment for Columns?        *}
10   39 --          rowInset:           INTEGER;    {* horizontal space between cells *}
10   40 --          colInset:           INTEGER;    {* vertical   space between cells *}
10   41 --          singleSelection:    BOOLEAN);   {* single cell selection?        *}
10   42 --
10   43 --      VAR
10   44 --          itsSize:            VPoint;
10   45 --          aHRunArray:         HRunArray;
10   46 --          tmpRgn:             RgnHandle;
10   47 --
10   48 0- A    BEGIN
10   49 --
10   50 --          fNumOfRows      := 0;
10   51 --          fNumOfCols      := 0;
10   52 --
10   53 --          fAdornRows := adornRows;
10   54 --          fAdornCols := adornCols;
10   55 --
10   56 --          {* Make sure the insets are evenly divided between top/bottom or left/right *}
10   57 --          IF ODD (rowInset) THEN
10   58 --              fRowInset := rowInset + 1
10   59 --          ELSE
10   60 --              fRowInset := rowInset;
10   61 --
10   62 --          IF ODD (colInset) THEN
10   63 --              fColInset := colInset + 1
10   64 --          ELSE
10   65 --              fColInset := colInset;
10   66 --
10   67 --
10   68 --          {* "Munger" is used to grow and shrink the array containing row heights and
10   69 --           * column widths.  The fRowHeights and fColWidths are initially handles
10   70 --           * to empty arrays.
10   71 --           *}
10   72 --
10   73 --          aHRunArray := HRunArray (NewPermHandle (0));
10   74 --          FailNil (aHRunArray);
10   75 --          fColWidths := aHRunArray;
10   76 --
10   77 --          aHRunArray := HRunArray (NewPermHandle (0));
10   78 --          FailNil (aHRunArray);
10   79 --          fRowHeights := aHRunArray;
10   80 --
10   81 --
10   82 --          {* for cacheing run array values *}
10   83 --          fLastRow := 0;
10   84 --          fLastRowChunkNum := 0;
10   85 --          fLastRowTotal := 0;
10   86 --          fLastRowIndex := 1;
10   87 --          fNumOfRowChunks := 0;
10   88 --
10   89 --          fLastCol := 0;
10   90 --          fLastColChunkNum := 0;
10   91 --          fLastColTotal := 0;
10   92 --          fLastColIndex := 1;
10   93 --          fNumOfColChunks := 0;
10   94 --
10   95 --          fMaxHorizontal  := 0;
10   96 --          fMaxVertical    := 0;
10   97 --
10   98 --          itsSize.v := 0;
```

```
10   99 --          itsSize.h := 0;
10  100 --
10  101 --          IView (itsDocument, itsSuperView, itsLocation, itsSize,
10  102 --              sizeVariable, sizeVariable);
10  103 --
10  104 --          tmpRgn  := MakeNewRgn;
10  105 --          fSelections := tmpRgn;      {* region to hold current selections *}
10  106 --
10  107 --          tmpRgn  := MakeNewRgn;
10  108 --          fHLRegion   := tmpRgn;      {* region to hold current highlighted cells *}
10  109 --
10  110 --          tmpRgn  := MakeNewRgn;
10  111 --          fTmpRgn    := tmpRgn;       {* temporary region for use on-the-fly *}
10  112 --
10  113 --          tmpRgn  := MakeNewRgn;
10  114 --          fTmpSelRgn    := tmpRgn;    {* temporary region for selection *}
10  115 --
10  116 --          tmpRgn  := MakeNewRgn;
10  117 --          fTmpHLRgn    := tmpRgn;     {* temporary region for highlighting *}
10  118 --
10  119 --          fSingleSelection := singleSelection;
10  120 --          fWholeRect := FALSE;        {* the new selected region is not a rectangle *}
10  121 --          fLastWholeRect := FALSE;    {* the old selected region is not a rectangle *}
10  122 --          fAllowDimSelection := FALSE; {* dimming the selection is possible *}
10  123 --
10  124 --          IF (numOfRows > 0) THEN
10  125 --              InsRowFirst (numOfRows, rowHeight);
10  126 --          IF (numOfCols > 0) THEN
10  127 --              InsColFirst  (numOfCols, colWidth);
10  128 --
10  129 -0 A    END; {* TGridView.IGridView *}
10  130 --     {$ENDC qTemplateViews}
10  131 --
10  132 --
10  133 --
10  134 --     {*****************************************************************************}
10  135 --     {*  TGridView.IRes                                                           *}
10  136 --     {*      Initialize the GridView for use with view templates.                 *}
10  137 --     {*****************************************************************************}
10  138 --     {$IFC qTemplateViews}
10  139 --     {$S GVOpen}
10  140 -- A   PROCEDURE TGridView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
10  141 --     VAR
10  142 --         aHRunArray:     HRunArray;
10  143 --         tmpRgn:         RgnHandle;
10  144 --
10  145 0- A   BEGIN
10  146 --         INHERITED IRes (itsDocument, itsSuperView, itsParams);
10  147 --
10  148 --         WITH GridViewTemplatePtr (itsParams)^ DO
10  149 1-         BEGIN
10  150 --             fNumOfRows      := 0;
10  151 --             fNumOfCols      := 0;
10  152 --
10  153 --             fAdornRows := adornRows;
10  154 --             fAdornCols := adornCols;
10  155 --
10  156 --             {* Make sure the insets are evenly divided between top/bottom or left/right *}
10  157 --             IF ODD (rowInset) THEN
10  158 --                 fRowInset := rowInset + 1
10  159 --             ELSE
10  160 --                 fRowInset := rowInset;
10  161 --
10  162 --             IF ODD (colInset) THEN
10  163 --                 fColInset := colInset + 1
10  164 --             ELSE
10  165 --                 fColInset := colInset;
10  166 --
10  167 --
10  168 --             fSingleSelection := singleSelection;
10  169 --             fWholeRect := FALSE;        {* the new selected region is not a rectangle *}
10  170 --             fLastWholeRect := FALSE;    {* the old selected region is not a rectangle *}
10  171 --             fAllowDimSelection := FALSE; {* dimming the selection is possible *}
10  172 --
10  173 --
10  174 --             {* "Munger" is used to grow and shrink the array containing row heights and
10  175 --              * column widths.  The fRowHeights and fColWidths are initially handles
10  176 --              * to empty arrays.
10  177 --              *}
10  178 --
10  179 --             aHRunArray := HRunArray (NewPermHandle (0));
10  180 --             FailNil (aHRunArray);
10  181 --             fColWidths := aHRunArray;
10  182 --
10  183 --             aHRunArray := HRunArray (NewPermHandle (0));
10  184 --             FailNil (aHRunArray);
10  185 --             fRowHeights := aHRunArray;
10  186 --
10  187 --
10  188 --             {* for cacheing run array values *}
10  189 --             fLastRow := 0;
10  190 --             fLastRowChunkNum := 0;
10  191 --             fLastRowTotal := 0;
10  192 --             fLastRowIndex := 1;
10  193 --             fNumOfRowChunks := 0;
10  194 --
10  195 --             fLastCol := 0;
10  196 --             fLastColChunkNum := 0;
10  197 --             fLastColTotal := 0;
```

```
10  198 --              fLastColIndex := 1;
10  199 --              fNumOfColChunks := 0;
10  200 --
10  201 --              fMaxHorizontal  := 0;
10  202 --              fMaxVertical    := 0;
10  203 --
10  204 --              tmpRgn  := MakeNewRgn;
10  205 --              fSelections := tmpRgn;      {* region to hold current selections *}
10  206 --
10  207 --              tmpRgn  := MakeNewRgn;
10  208 --              fHLRegion   := tmpRgn;      {* region to hold current highlighted cells *}
10  209 --
10  210 --              tmpRgn  := MakeNewRgn;
10  211 --              fTmpRgn    := tmpRgn;       {* temporary region for use on-the-fly *}
10  212 --
10  213 --              tmpRgn  := MakeNewRgn;
10  214 --              fTmpSelRgn    := tmpRgn;    {* temporary region for selection *}
10  215 --
10  216 --              tmpRgn  := MakeNewRgn;
10  217 --              fTmpHLRgn    := tmpRgn;     {* temporary region for highlighting *}
10  218 --
10  219 --              IF (itsSuperView <> NIL) & (fSizeDeterminer[h] <> sizeFixed) THEN
10  220 --                  fSize.h := itsSuperView.fSize.h;
10  221 --
10  222 --              IF (numOfRows > 0) THEN
10  223 --                  InsRowFirst (numOfRows, rowHeight);
10  224 --              IF (numOfCols > 0) THEN
10  225 --                  InsColFirst  (numOfCols, colWidth);
10  226 --
10  227 -1          END;
10  228 --
10  229 --          OffsetPtr (itsParams, SIZEOF (GridViewTemplate));
10  230 -0 A    END; {* TGridView.IRes *}
10  231 --      {$ENDC qTemplateViews}
10  232 --
10  233 --
10  234 --      {*****************************************************************************}
10  235 --      {*  TGridView.WRes                                                           *}
10  236 --      {*      Write the resource file.                                             *}
10  237 --      {*****************************************************************************}
10  238 --      {$IFC qWriteTemplates}
10  239 --      {$S GVNonRes}
10  240 -- A    PROCEDURE TGridView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
10  241 --
10  242 --      VAR
10  243 --          gvPtr:      GridViewTemplatePtr;
10  244 --
10  245 0- A    BEGIN
10  246 --          INHERITED WRes(theResource, itsParams);
10  247 --
10  248 --          gvPtr := GridViewTemplatePtr(ExpandPtr(theResource, itsParams,
10  249 --            SIZEOF(GridViewTemplate)));
10  250 --
10  251 --          WITH gvPtr^ DO
10  252 1-            BEGIN
10  253 --              numOfRows := fNumOfRows;
10  254 --              numOfCols := fNumOfCols;
10  255 --              IF fNumOfRows > 0 THEN
10  256 --                  rowHeight := GetRowHeight(1)
10  257 --              ELSE
10  258 --                  rowHeight := 0;
10  259 --              IF fNumOfCols > 0 THEN
10  260 --                  colWidth := GetColWidth(1)
10  261 --              ELSE
10  262 --                  colWidth := 0;
10  263 --              rowInset := fRowInset;
10  264 --              colInset := fColInset;
10  265 --              adornRows := fAdornRows;
10  266 --              adornCols := fAdornCols;
10  267 --              singleSelection := fSingleSelection;
10  268 -1              END;
10  269 -0 A    END; {* TGridView.WRes *}
10  270 --
10  271 --
10  272 --      {*****************************************************************************}
10  273 --      {*  TGridView.WriteRes                                                       *}
10  274 --      {*      Set up for writing the resource file                                 *}
10  275 --      {*****************************************************************************}
10  276 --      {$S GVNonRes}
10  277 -- A    PROCEDURE TGridView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
10  278 0- A    BEGIN
10  279 --          gWResSignature := 'grid';   gWResType := 'TGridView';
10  280 --          WRes(theResource, itsParams);
10  281 -0 A    END; {* TGridView.WriteRes *}
10  282 --      {$ENDC qWriteTemplates}
10  283 --
10  284 --
10  285 --      {*****************************************************************************}
10  286 --      {*  TGridView.Free                                                           *}
10  287 --      {*      Free the TGridView object and any associated objects                 *}
10  288 --      {*****************************************************************************}
10  289 --      {$S GVClose}
10  290 -- A    PROCEDURE TGridView.Free; OVERRIDE;
10  291 0- A    BEGIN
10  292 --          DisposIfHandle (fSelections);       {* Dispose regions *}
10  293 --          DisposIfHandle (fHLRegion);
10  294 --          DisposIfHandle (fTmpRgn);
10  295 --          DisposIfHandle (fTmpSelRgn);
10  296 --          DisposIfHandle (fTmpHLRgn);
```

```
10  297 --          DisposIfHandle (fColWidths);        {* Dispose run arrays *}
10  298 --          DisposIfHandle (fRowHeights);
10  299 --
10  300 --          INHERITED Free;
10  301 -0 A    END; {* TGridView.Free *}
10  302 --
10  303 --
10  304 --      {*****************************************************************************}
10  305 --      {*  TGridView.AllCellsDo                                                     *}
10  306 --      {*      For all cells in the view perform "DoToCell".                        *}
10  307 --      {*****************************************************************************}
10  308 --      {$S GVRes}
10  309 -- A    PROCEDURE TGridVIew.AllCellsDo (PROCEDURE DoToCell (aCell: GridCell));
10  310 --      VAR
10  311 --          startCell:  GridCell;
10  312 --          stopCell:   GridCell;
10  313 --
10  314 0- A    BEGIN
10  315 --          startCell.h := 1;
10  316 --          startCell.v := 1;
10  317 --          stopCell.h  := fNumOfCols;
10  318 --          stopCell.v  := fNumOfRows;
10  319 --
10  320 --          EachCellDo (startCell, stopCell, DoToCell);
10  321 -0 A    END; {* TGridView.AllCellsDo *}
10  322 --
10  323 --
10  324 --      {*****************************************************************************}
10  325 --      {*  TGridView.AdornCol                                                       *}
10  326 --      {*      Draw the column adornments for the whole column.                     *}
10  327 --      {*****************************************************************************}
10  328 --      {$S GVRes}
10  329 -- A    PROCEDURE TGridView.AdornCol (aCol: INTEGER; area: Rect);
10  330 0- A    BEGIN
10  331 -0 A    END; {* TGridView.AdornCol *}
10  332 --
10  333 --
10  334 --      {*****************************************************************************}
10  335 --      {*  TGridView.AdornRow                                                       *}
10  336 --      {*      Draw the row adornments for the whole row.                           *}
10  337 --      {*****************************************************************************}
10  338 --      {$S GVRes}
10  339 -- A    PROCEDURE TGridView.AdornRow (aRow: INTEGER; area: Rect);
10  340 0- A    BEGIN
10  341 -0 A    END; {* TGridView.AdornRow *}
10  342 --
10  343 --
10  344 --      {*****************************************************************************}
10  345 --      {*  TGridView.CalcMinSize                                                    *}
10  346 --      {*      This method adjusts the size of the extent by the amount of space    *}
10  347 --      {*      needed to display all the rows and columns.  It is called (by TView. *}
10  348 --      {*      ComputeExtent) whenever a row or column is added or deleted.         *}
10  349 --      {*****************************************************************************}
10  350 --      {$S GVRes}
10  351 -- A    PROCEDURE TGridView.CalcMinSize (VAR minSize: VPoint);  OVERRIDE;
10  352 0- A    BEGIN
10  353 --          INHERITED CalcMinSize (minSize);
10  354 --
10  355 --          {* Set the amount of room needed for that many items *}
10  356 --          minSize.v := fMaxVertical;
10  357 --          minSize.h := fMaxHorizontal;
10  358 -0 A    END; {* TGridView.CalcMinSize *}
10  359 --
10  360 --
10  361 --      {*****************************************************************************}
10  362 --      {*  TGridView.CanSelectCell                                                  *}
10  363 --      {*      Determine if a cell can be selected.  By default, this method returns*}
10  364 --      {*      TRUE for everything.                                                 *}
10  365 --      {*****************************************************************************}
10  366 --      {$S GVRes}
10  367 -- A    FUNCTION TGridView.CanSelectCell (aCell: GridCell): BOOLEAN;
10  368 --
10  369 0- A    BEGIN
10  370 --          {For now, allow any item to be selected.}
10  371 --          CanSelectCell := TRUE;
10  372 -0 A    END; {* TGridView.CanSelectCell *}
10  373 --
10  374 --
10  375 --      {*****************************************************************************}
10  376 --      {*  TGridView.CellExists                                                     *}
10  377 --      {*      Determine if a cell can be drawn.  By default, this method returns   *}
10  378 --      {*      TRUE for everything.                                                 *}
10  379 --      {*****************************************************************************}
10  380 --      {$S GVRes}
10  381 -- A    FUNCTION TGridView.CellExists (aCell: GridCell): BOOLEAN;
10  382 --
10  383 0- A    BEGIN
10  384 --          {allow any cell to be drawn.}
10  385 --          CellExists := TRUE;
10  386 -0 A    END; {* TGridView.CellExists *}
10  387 --
10  388 --
10  389 --      {*****************************************************************************}
10  390 --      {*  TGridView.CellToVRect                                                    *}
10  391 --      {*      Calculate the bounding rectangle for a given cell.  This includes    *}
10  392 --      {*      any row or column insets.                                            *}
10  393 --      {*****************************************************************************}
10  394 --      {$S GVRes}
10  395 -- A    PROCEDURE  TGridView.CellToVRect (aCell: GridCell; VAR aRect: VRect);
```

```
10  396 --      VAR
10  397 --          bRect:      VRect;
10  398 --
10  399 0- A    BEGIN
10  400 --          IF (aCell.h < 1) | (aCell.v < 1) |
10  401 --                  (aCell.h > fNumOfCols) | (aCell.v > fNumOfRows) THEN
10  402 1-          BEGIN
10  403 --      {$IFC qRangeCheck}
10  404 --      {$IFC qDebug}
10  405 --              Writeln('aCell.h = ', aCell.h:6, ' fNumOfCols = ', fNumOfCols:6);
10  406 --              Writeln('aCell.v = ', aCell.v:6, ' fNumOfRows = ', fNumOfRows:6);
10  407 --      {$ENDC}
10  408 --              ProgramBreak('Range Check in CellToVRect');
10  409 --      {$ENDC}
10  410 --              SetVRect (aRect, 0, 0, 0, 0);
10  411 -1          END
10  412 --          ELSE    {* all the params look OK *}
10  413 1-          BEGIN
10  414 --              RowToVRect (aCell.v, 1, aRect);
10  415 --              ColToVRect (aCell.h, 1, bRect);
10  416 --              WITH aRect DO
10  417 2-              BEGIN
10  418 --                  left    := bRect.left;
10  419 --                  right   := bRect.right;
10  420 -2              END;
10  421 -1          END;
10  422 -0 A   END; {* TGridView.CellToVRect *}
10  423 --
10  424 --
10  425 --      {*****************************************************************************}
10  426 --      {*  TGridView.ColToVRect                                                     *}
10  427 --      {*      Calculate the bounding rectangle for the given column(s).  The top and  *}
10  428 --      {*      bottom of the rectangle are the same as the view.                    *}
10  429 --      {*      The rectangle includes the cells in that column and the column inset.   *}
10  430 --      {*****************************************************************************}
10  431 --      {$S GVRes}
10  432 -- A   PROCEDURE TGridView.ColToVRect (aCol: INTEGER; numOfCols: INTEGER; VAR aRect: VRect);
10  433 --      VAR
10  434 --          width:      LONGINT;
10  435 --          Num:        INTEGER;
10  436 --          theTotal:   LONGINT;
10  437 --          Index:      INTEGER;
10  438 --          i:          INTEGER;
10  439 --
10  440 0- A   BEGIN
10  441 --          IF (aCol < 1) | (numOfCols < 1) | (aCol + numOfCols - 1 > fNumOfCols) THEN
10  442 1-          BEGIN
10  443 --      {$IFC qRangeCheck}
10  444 --      {$IFC qDebug}
10  445 --              Writeln('fNumOfCols = ', fNumOfCols:1, '  aCol = ', aCol:1);
10  446 --      {$ENDC}
10  447 --              ProgramBreak('Range Check in ColToVRect');
10  448 --      {$ENDC}
10  449 --              SetVRect (aRect, 0, 0, 0, 0);
10  450 -1          END
10  451 --          ELSE    {* all the params look OK *}
10  452 1-          BEGIN
10  453 --              width := 0;
10  454 --
10  455 --              IF (fNumOfColChunks = 1) THEN  {* only one column height *}
10  456 2-              BEGIN
10  457 --                  num := 0;
10  458 --                  theTotal := IntMultiply (fColWidths^^[num].size, aCol-1);
10  459 --                  width := IntMultiply (numOfCols, fColWidths^^[num].size);
10  460 -2              END
10  461 --              ELSE
10  462 2-              BEGIN
10  463 --                  IF FindColChunk (aCol, Num, theTotal, Index) THEN
10  464 3-                  BEGIN
10  465 --                      theTotal := theTotal + IntMultiply ((Index-1), fColWidths^^[Num].size);
10  466 --                      For i := aCol TO (aCol + numOfCols - 1) DO
10  467 --                          width := width + GetColWidth (i);
10  468 -3                  END;
10  469 -2              END;
10  470 --
10  471 --              WITH aRect DO
10  472 2-              BEGIN
10  473 --                  top     := 0;
10  474 --                  left    := theTotal;
10  475 --                  bottom  := fMaxVertical;
10  476 --                  right   := theTotal + width;
10  477 -2              END;
10  478 -1          END;
10  479 -0 A   END; {* TGridView.ColToVRect *}
10  480 --
10  481 --
10  482 --      {*****************************************************************************}
10  483 --      {*  TGridView.CreateHighlightRgn                                             *}
10  484 --      {*      This guy goes through all the cells in oldRgn and creates a          *}
10  485 --      {*      newRgn to be used for highlighting.                                  *}
10  486 --      {*****************************************************************************}
10  487 --      {$S GVRes}
10  488 -- A   PROCEDURE TGridView.CreateHighlightRgn (oldRgn, newRgn: RgnHandle; wholeRect: BOOLEAN);
10  489 --      VAR
10  490 --          aRect:      VRect;
10  491 --          bRect:      VRect;
10  492 --          aQDRect:    Rect;
10  493 --
10  494 -- B        PROCEDURE AddToHLRegion (aCell: GridCell);
```

```
10  495 0- B        BEGIN
10  496 --              IF ((aCell.h > 0) & (aCell.v > 0) &
10  497 --                  (aCell.h <= fNumOfCols) & (aCell.v <= fNumOfRows)) THEN
10  498 1-              BEGIN
10  499 --                  CellToVRect (aCell, aRect);
10  500 --                  ViewToQDRect (aRect, aQDRect);
10  501 --                  RectRgn (fTmpHLRgn, aQDRect);
10  502 --                  UnionRgn (fTmpHLRgn, newRgn, newRgn);
10  503 -1              END;
10  504 -0 B        END;
10  505 --
10  506 0- A    BEGIN
10  507 --          IF EmptyRgn (oldRgn) THEN
10  508 --              SetEmptyRgn (newRgn)
10  509 --
10  510 --          ELSE IF wholeRect THEN
10  511 1-          BEGIN
10  512 --              aQDRect := oldRgn^^.rgnBBox;
10  513 --              aQDRect.bottom := aQDRect.bottom - 1;
10  514 --              aQDRect.right := aQDRect.right - 1;
10  515 --
10  516 --              IF ((aQDRect.top > 0) & (aQDRect.left > 0) &
10  517 --                  (aQDRect.right <= fNumOfCols) &
10  518 --                  (aQDRect.bottom <= fNumOfRows) &
10  519 --                  (aQDRect.top <= aQDRect.bottom) &
10  520 --                  (aQDRect.left <= aQDRect.right)) THEN
10  521 2-              BEGIN
10  522 --                  CellToVRect (aQDRect.topleft, aRect);
10  523 --                  CellToVRect (aQDRect.botright, bRect);
10  524 --                  aRect.botright := bRect.botright;
10  525 --                  ViewToQDRect (aRect, aQDRect);
10  526 --                  RectRgn (newRgn, aQDRect);
10  527 -2              END;
10  528 -1          END
10  529 --          ELSE
10  530 1-          BEGIN
10  531 --              SetEmptyRgn (newRgn);
10  532 --              EachInRgn (oldRgn, AddToHLRegion);
10  533 -1          END;
10  534 -0 A    END; {* TGridView.CreateHighlightRgn *}
10  535 --
10  536 --
10  537 --      {*****************************************************************************}
10  538 --      {*  TGridView.DoHighlightSelection                                           *}
10  539 --      {*      Figure out what needs to be highlighted and call DoHilite.           *}
10  540 --      {*****************************************************************************}
10  541 --      {$S GVRes}
10  542 -- A    PROCEDURE TGridView.DoHighlightSelection (fromHL, toHL: HLState); OVERRIDE;
10  543 --      VAR
10  544 --          hlRect:     Rect;
10  545 --          aQDRect:    Rect;
10  546 --          aRect:      VRect;
10  547 --          bRect:      VRect;
10  548 --
10  549 0- A    BEGIN
10  550 --          IF NOT EmptyRgn (fHLRegion) THEN
10  551 1-          BEGIN
10  552 --              CreateHighlightRgn (fHLRegion, fTmpRgn, fWholeRect);
10  553 --              DoHilite (fTmpRgn, fromHL, toHL)
10  554 -1          END;
10  555 -0 A    END; {* TGridView.DoHighlightSelection *}
10  556 --
10  557 --
10  558 --      {*****************************************************************************}
10  559 --      {*  TGridView.DoHilite                                                       *}
10  560 --      {*      Invert the cell.  Override this guy to have different highlighting.  *}
10  561 --      {*****************************************************************************}
10  562 --      {$S GVRes}
10  563 -- A    PROCEDURE TGridView.DoHilite (hlRgn: RgnHandle; fromHL, toHL: HLState);
10  564 --      VAR
10  565 --          aRect:      VRect;
10  566 --          bRect:      VRect;
10  567 --          aQDRect:    Rect;
10  568 --          bQDRect:    Rect;
10  569 0- A    BEGIN
10  570 --          IF focus THEN
10  571 1-          BEGIN
10  572 --              IF NOT fAllowDimSelection THEN
10  573 2-              BEGIN
10  574 --                  IF fromHL = hlDim THEN
10  575 --                      fromHL := hlOFF;
10  576 --                  IF toHL = hlDim THEN
10  577 --                      toHL := hlOFF;
10  578 -2              END;
10  579 --
10  580 --              PenNormal;
10  581 --              IF (fromHL + toHL = hlOnOff) THEN
10  582 2-              BEGIN
10  583 --                  InvertRgn (hlRgn);  {* highlight the cells *}
10  584 -2              END
10  585 --              ELSE IF fromHL = hlDim THEN
10  586 2-              BEGIN
10  587 --                  IF toHL = hlON THEN
10  588 3-                  BEGIN
10  589 --                      InvertRgn (hlRgn);
10  590 --                      PenMode (patXor);
10  591 --                      FrameRgn (hlRgn);   {* erase the 'dim' *}
10  592 -3                  END
10  593 --                  ELSE {* toHL = hlOFF *}
```

```
10  594 3-                BEGIN
10  595 --                    PenMode (patXor);
10  596 --                    FrameRgn (hlRgn);   {* erase the 'dim' *}
10  597 -3                END;
10  598 -2            END
10  599 --            ELSE IF toHL = hlDim THEN
10  600 2-            BEGIN
10  601 --                IF fromHL = hlON THEN
10  602 --                    InvertRgn (hlRgn);
10  603 --                PenMode (patXor);
10  604 --                FrameRgn (hlRgn);   {* 'dim' the rect *}
10  605 -2            END;
10  606 -1        END;
10  607 -0 A   END; {* TGridView.DoHilite *}
10  608 --
10  609 --
10  610 --      {*****************************************************************************}
10  611 --      {*  TGridView.DoMouseCommand                                                 *}
10  612 --      {*      Determine if the mouse click was in a cell, a row, a column, or a    *}
10  613 --      {*      vertex.  (A vertex is the place where a row and column meet.)  Override *}
10  614 --      {*      IdentifyPoint to change the logic for how a point gets classified.   *}
10  615 --      {*      Based on the point classification, a command object is created and   *}
10  616 --      {*      control is passed on to that object.                                 *}
10  617 --      {*****************************************************************************}
10  618 --      {$S GVRes}
10  619 -- A    FUNCTION  TGridView.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
10  620 --                                          VAR hysteresis: Point) : TCommand; OVERRIDE;
10  621 --
10  622 --      VAR
10  623 --          whichPart:              GridViewPart;
10  624 --          aRow, aCol:             INTEGER;
10  625 --          aCellSelectCommand:     TCellSelectCommand;
10  626 --          aRowSelectCommand:      TRowSelectCommand;
10  627 --          aColumnSelectCommand:   TColSelectCommand;
10  628 --          aVertexSelectCommand:   TVertexSelectCommand;
10  629 --          aCell:                  GridCell;
10  630 --
10  631 0- A    BEGIN
10  632 --
10  633 --          whichPart := IdentifyPoint (themouse, aRow, aCol);
10  634 --          aCell.h := aCol;
10  635 --          aCell.v := aRow;
10  636 --
10  637 1-          CASE whichPart OF
10  638 --              inCell:
10  639 2-                  BEGIN
10  640 --                  NEW (aCellSelectCommand);
10  641 --                  FailNIL (aCellSelectCommand);
10  642 --                  aCellSelectCommand.ICellSelectCommand (SELF, aCell, info.theShiftKey,
10  643 --                      info.theCmdKey);
10  644 --                  DoMouseCommand := aCellSelectCommand;
10  645 -2                  END;
10  646 --
10  647 --              inRow:
10  648 2-                  BEGIN
10  649 --                  NEW (aRowSelectCommand);
10  650 --                  FailNIL (aRowSelectCommand);
10  651 --                  aRowSelectCommand.IRowSelectCommand (SELF, aRow, info.theShiftKey,
10  652 --                      info.theCmdKey);
10  653 --                  DoMouseCommand := aRowSelectCommand;
10  654 -2                  END;
10  655 --
10  656 --              inColumn:
10  657 2-                  BEGIN
10  658 --                  NEW (aColumnSelectCommand);
10  659 --                  FailNIL (aColumnSelectCommand);
10  660 --                  aColumnSelectCommand.IColSelectCommand (SELF, aCol, info.theShiftKey,
10  661 --                      info.theCmdKey);
10  662 --                  DoMouseCommand := aColumnSelectCommand;
10  663 -2                  END;
10  664 --
10  665 --              inVertex:
10  666 2-                  BEGIN
10  667 --                  NEW (aVertexSelectCommand);
10  668 --                  FailNIL (aVertexSelectCommand);
10  669 --                  aVertexSelectCommand.IVertexSelectCommand (SELF, aRow, aCol,
10  670 --                      info.theShiftKey, info.theCmdKey);
10  671 --                  DoMouseCommand := aVertexSelectCommand;
10  672 -2                  END;
10  673 --
10  674 --              OTHERWISE
10  675 --                  DoMouseCommand := gNoChanges;
10  676 -1              END;
10  677 -0 A   END; {* TGridView.DoMouseCommand *}
10  678 --
10  679 --
10  680 --      {*****************************************************************************}
10  681 --      {*  TGridView.Draw                                                           *}
10  682 --      {*      Draw each cell in "area".  If either adorn is TRUE then call that    *}
10  683 --      {*      Adorn.  ONLY the cells that need to be re-drawn are drawn.           *}
10  684 --      {*****************************************************************************}
10  685 --      {$S GVRes}
10  686 -- A    PROCEDURE TGridView.Draw (area: Rect); OVERRIDE;
10  687 --
10  688 --      VAR
10  689 --          aRect:          VRect;
10  690 --          bRect:          VRect;
10  691 --          aQDRect:        Rect;
10  692 --          i:              INTEGER;
```

```
10  693 --          startCell:      GridCell;
10  694 --          stopCell:       GridCell;
10  695 --          viewArea:       VRect;
10  696 --          colWidth:       INTEGER;
10  697 --          rowHeight:      INTEGER;
10  698 --
10  699 0- A    BEGIN
10  700 --          IF (fNumOfRows > 0) & (fNumOfCols > 0) THEN
10  701 1-          BEGIN   {* make sure we have something to draw *}
10  702 --              QDtoViewRect (area, viewArea);
10  703 --              viewarea.right  := viewarea.right - 1;
10  704 --              viewarea.bottom := viewarea.bottom - 1;
10  705 --
10  706 --
10  707 --              WITH viewArea DO
10  708 2-              BEGIN
10  709 --                  startCell := VPointToCell (topleft);
10  710 --                  stopCell := VPointToCell (botright);
10  711 -2              END;
10  712 --
10  713 --
10  714 --              CellToVRect (startCell, aRect);
10  715 --              CellToVRect (stopCell, bRect);
10  716 --              aRect.botright := bRect.botright;
10  717 --              ViewToQDRect (aRect, area);
10  718 --
10  719 --              DrawRangeOfCells (startCell, stopCell, area);
10  720 --
10  721 --              IF fAdornCols THEN
10  722 2-              BEGIN
10  723 --                  aQDRect := area;
10  724 --
10  725 --                  IF fNumOfColChunks = 1 THEN     {* only one width *}
10  726 --                      colWidth := fColWidths^^[0].size;
10  727 --
10  728 --                  For i := startCell.h to stopCell.h DO
10  729 3-                  BEGIN
10  730 --                      IF fNumOfColChunks = 1 THEN  {* only one height *}
10  731 --                          aQDRect.right := aQDRect.left + colWidth
10  732 --                      ELSE
10  733 --                          aQDRect.right := aQDRect.left + GetColWidth (i);
10  734 --
10  735 --                      AdornCol (i, aQDRect);
10  736 --                      aQDRect.left := aQDRect.right;
10  737 -3                  END;
10  738 -2              END;
10  739 --
10  740 --              IF fAdornRows THEN
10  741 2-              BEGIN
10  742 --                  aQDRect := area;
10  743 --
10  744 --                  IF fNumOfRowChunks = 1 THEN     {* only one height *}
10  745 --                      rowHeight := fRowHeights^^[0].size;
10  746 --
10  747 --                  For i := startCell.v to stopCell.v DO
10  748 3-                  BEGIN
10  749 --                      IF fNumOfRowChunks = 1 THEN  {* only one height *}
10  750 --                          aQDRect.bottom := aQDRect.top + rowheight
10  751 --                      ELSE
10  752 --                          aQDRect.bottom := aQDRect.top + GetRowHeight (i);
10  753 --
10  754 --                      AdornRow (i, aQDRect);
10  755 --                      aQDRect.top := aQDRect.bottom;
10  756 -3                  END;
10  757 -2              END;
10  758 -1          END;
10  759 -0 A    END; {* TGridView.Draw *}
10  760 --
10  761 --
10  762 --      {*****************************************************************************}
10  763 --      {*  TGridView.DrawRangeOfCells                                               *}
10  764 --      {*      This method calls DrawCell for each cell that needs to be drawn.     *}
10  765 --      {*****************************************************************************}
10  766 --      {$S GVRes}
10  767 -- A    PROCEDURE TGridView.DrawRangeOfCells (startCell, stopCell: GridCell; aQDRect: Rect);
10  768 --      VAR
10  769 --          colWidth:   INTEGER;
10  770 --          rowHeight:  INTEGER;
10  771 --          i, j:       INTEGER;
10  772 --          aCell:      GridCell;
10  773 --          left:       INTEGER;
10  774 --
10  775 0- A    BEGIN
10  776 --
10  777 --          aQDRect.left := aQDRect.left + BSR (fColInset, 1);  {* fColInset DIV 2 *}
10  778 --          aQDRect.top  := aQDRect.top  + BSR (fRowInset, 1);  {* fRowInset DIV 2 *}
10  779 --          left := aQDRect.left;
10  780 --
10  781 --          IF fNumOfColChunks = 1 THEN     {* only one width *}
10  782 --              colWidth := GetColWidth (1);
10  783 --          IF fNumOfRowChunks = 1 THEN     {* only one height *}
10  784 --              rowHeight := GetRowHeight (1);
10  785 --
10  786 --
10  787 --          For j := startCell.v to stopCell.v DO
10  788 1-          BEGIN
10  789 --              IF fNumOfRowChunks = 1 THEN  {* only one height *}
10  790 --                  aQDRect.bottom := aQDRect.top + rowheight - fRowInset
10  791 --              ELSE
```

```
10  792 --                aQDRect.bottom := aQDRect.top + GetRowHeight (j) - fRowInset;
10  793 --
10  794 --            aQDRect.left := left;  {* start back at the left for the next row *}
10  795 --
10  796 --            For i := startCell.h to stopCell.h DO
10  797 2-            BEGIN
10  798 --                IF fNumOfColChunks = 1 THEN  {* only one height *}
10  799 --                    aQDRect.right := aQDRect.left + colWidth - fColInset
10  800 --                ELSE
10  801 --                    aQDRect.right := aQDRect.left + GetColWidth (i) - fColInset;
10  802 --
10  803 --
10  804 --                aCell.h := i;
10  805 --                aCell.v := j;
10  806 --                IF CellExists (aCell) THEN
10  807 --                    DrawCell (aCell, aQDRect);
10  808 --
10  809 --                aQDRect.left := aQDRect.right + fColInset;
10  810 -2            END;
10  811 --            aQDRect.top := aQDRect.bottom + fRowInset;
10  812 -1        END;
10  813 -0 A   END; {* TGridView.DrawRangeOfCells *}
10  814 --
10  815 --
10  816 --    {*****************************************************************************}
10  817 --    {*  TGridView.DrawCell                                                       *}
10  818 --    {*      This method draws the given cell inside the specified rectangle.  The   *}
10  819 --    {*      rectangle is the cell size without any "insets".                     *}
10  820 --    {*      It MUST be overridden!!                                              *}
10  821 --    {*****************************************************************************}
10  822 --    {$S GVRes}
10  823 -- A   PROCEDURE TGridView.DrawCell (aCell: GridCell; aQDRect: Rect);
10  824 0- A   BEGIN
10  825 --        { Should always be overridden.}
10  826 --
10  827 --    {$IFC qDebug}
10  828 --        WRITELN ('TGridView: DrawCell MUST be overridden!');
10  829 --    {$ENDC qDebug}
10  830 -0 A   END; {* TGridView.DrawCell *}
10  831 --
10  832 --
10  833 --    {*****************************************************************************}
10  834 --    {*  TGridView.DelColAt                                                       *}
10  835 --    {*      Delete numOfCols starting at aCol.  This method causes a re-calculation *}
10  836 --    {*      of the extent and redraws only the necessary cells (and adornments).    *}
10  837 --    {*****************************************************************************}
10  838 --    {$S GVNonRes}
10  839 -- A   PROCEDURE TGridView.DelColAt (aCol: INTEGER; numOfCols: INTEGER);
10  840 --    VAR
10  841 --        offset:     LONGINT;
10  842 --        dummy:      Ptr;
10  843 --        result:     INTEGER;
10  844 --        i:          INTEGER;
10  845 --        aRect:      VRect;
10  846 --        Num:        INTEGER;
10  847 --        theTotal:   LONGINT;
10  848 --        Index:      INTEGER;
10  849 --        chunkFound: BOOLEAN;
10  850 --        selRect:    Rect;
10  851 --
10  852 0- A   BEGIN
10  853 --        IF (aCol < 1) | (numOfCols < 1) | (aCol + numOfCols - 1 > fNumOfCols) THEN
10  854 1-        BEGIN
10  855 --            IF numOFCols <> 0 THEN
10  856 2-            BEGIN
10  857 --    {$IFC qRangeCheck}
10  858 --    {$IFC qDebug}
10  859 --                Writeln('fNumOfCols = ', fNumOfCols:1, '  aCol = ', aCol:1);
10  860 --    {$ENDC}
10  861 --                ProgramBreak('Range Check in DelColAt');
10  862 --    {$ENDC}
10  863 -2            END;
10  864 -1        END
10  865 --        ELSE
10  866 1-        BEGIN
10  867 --            {* make sure we take any deleted items out of the selection *}
10  868 --            selRect.top := 1;
10  869 --            selRect.left := aCol;
10  870 --            selRect.bottom := fNumOfRows + 1;
10  871 --            selRect.right := aCol + numOfCols;
10  872 --            RectRgn (fTmpRgn, selRect);
10  873 --            DiffRgn (fSelections, fTmpRgn, fSelections);
10  874 --            DiffRgn (fHLRegion, fTmpRgn, fHLRegion);
10  875 --
10  876 --            ColToVRect (Max (1, aCol), Max (1, fNumOfCols - aCol), aRect);
10  877 --            chunkFound := FindColChunk (aCol, Num, theTotal, Index);
10  878 --
10  879 --            FOR i := 1 TO numOfCols DO
10  880 2-            BEGIN
10  881 --                fMaxHorizontal := fMaxHorizontal - fColWidths^^[Num].size;
10  882 --
10  883 --                fColWidths^^[Num].count := fColWidths^^[Num].count - 1;
10  884 --
10  885 --                IF (fColWidths^^[Num].count < Index) THEN
10  886 3-                BEGIN
10  887 --                    IF (fColWidths^^[Num].count = 0) THEN
10  888 4-                    BEGIN
10  889 --                        {* need to delete that chunk *}
10  890 --                        offset := LONGINT (IntMultiply (4, Num));
```

```
10  891 --                          result := Munger (Handle (fColWidths), offset, Ptr (Ord (fColWidths^) + offset),
10  892 --                              sizeof (LONGINT), dummy, 0);
10  893 --                          fNumOfColChunks := fNumOfColChunks - 1;
10  894 --
10  895 --                          {* see if we can consolidate chunks *}
10  896 --                          IF (Num > 0) & (fColWidths^^[Num - 1].size = fColWidths^^[Num].size) THEN
10  897 5-                        BEGIN
10  898 --                              fColWidths^^[Num - 1].count := fColWidths^^[Num - 1].count +
10  899 --                                                              fColWidths^^[Num].count;
10  900 --                              {* need to delete that chunk *}
10  901 --                              offset := LONGINT (IntMultiply (4, Num));
10  902 --                              result := Munger (Handle (fColWidths), offset, Ptr (Ord (fColWidths^) + offset),
10  903 --                                  sizeof (LONGINT), dummy, 0);
10  904 --                              fNumOfColChunks := fNumOfColChunks - 1;
10  905 -5                          END;
10  906 -4                      END
10  907 --                      ELSE
10  908 4-                      BEGIN
10  909 --                          Num := Num + 1;
10  910 -4                      END;
10  911 --                      Index := 1;
10  912 -3                  END;
10  913 -2              END;
10  914 --
10  915 --              fNumOfCols := fNumOfCols - numOfCols;
10  916 --
10  917 --              {* reset the run array info *}
10  918 --              fLastCol := 0;
10  919 --              fLastColChunkNum := 0;
10  920 --              fLastColTotal := 0;
10  921 --              fLastColIndex := 1;
10  922 --
10  923 --              AdjustSize;
10  924 --
10  925 --              InvalidVRect (aRect);
10  926 -1          END;
10  927 -0 A    END; {* TGridView.DelColAt *}
10  928 --
10  929 --
10  930 --      {*****************************************************************************}
10  931 --      {*  TGridView.DelRowAt                                                       *}
10  932 --      {*      Delete numOfRows starting at aRow.  This method causes a re-calculation *}
10  933 --      {*      of the extent and redraws only the necessary cells (and adornments).    *}
10  934 --      {*****************************************************************************}
10  935 --      {$S GVNonRes}
10  936 -- A    PROCEDURE TGridView.DelRowAt (aRow: INTEGER; numOfRows: INTEGER);
10  937 --      VAR
10  938 --          offset:     LONGINT;
10  939 --          dummy:      Ptr;
10  940 --          result:     INTEGER;
10  941 --          i:          INTEGER;
10  942 --          aRect:      VRect;
10  943 --          Num:        INTEGER;
10  944 --          theTotal:   LONGINT;
10  945 --          Index:      INTEGER;
10  946 --          chunkFound: BOOLEAN;
10  947 --          selRect:    Rect;
10  948 --
10  949 0- A    BEGIN
10  950 --          IF (aRow < 1) | (numOfRows < 1) | (aRow + numOfRows - 1 > fNumOfRows) THEN
10  951 1-          BEGIN
10  952 --              IF numOfRows <> 0 THEN
10  953 2-              BEGIN
10  954 --      {$IFC qRangeCheck}
10  955 --      {$IFC qDebug}
10  956 --                  Writeln('fNumOfRows = ', fNumOfRows:1, '  aRow = ', aRow:1);
10  957 --      {$ENDC}
10  958 --                  ProgramBreak('Range Check in DelRowAt');
10  959 --      {$ENDC}
10  960 -2              END;
10  961 -1          END
10  962 --          ELSE
10  963 1-          BEGIN
10  964 --              {* make sure we take any deleted items out of the selection *}
10  965 --              selRect.top := aRow;
10  966 --              selRect.left := 1;
10  967 --              selRect.bottom := aRow + numOfRows;
10  968 --              selRect.right := fNumOfCols + 1;
10  969 --              RectRgn (fTmpRgn, selRect);
10  970 --              DiffRgn (fSelections, fTmpRgn, fSelections);
10  971 --              DiffRgn (fHLRegion, fTmpRgn, fHLRegion);
10  972 --
10  973 --              RowToVRect (Max (1, aRow), Max (1, fNumOfRows - aRow), aRect);
10  974 --              chunkFound := FindRowChunk (aRow, Num, theTotal, Index);
10  975 --
10  976 --              FOR i := 1 TO numOfRows DO
10  977 2-              BEGIN
10  978 --                  fMaxVertical := fMaxVertical - fRowHeights^^[Num].size;
10  979 --
10  980 --                  fRowHeights^^[Num].count := fRowHeights^^[Num].count - 1;
10  981 --
10  982 --                  IF (fRowHeights^^[Num].count < Index) THEN
10  983 3-                  BEGIN
10  984 --                      IF (fRowHeights^^[Num].count = 0) THEN
10  985 4-                      BEGIN
10  986 --                          {* need to delete that chunk *}
10  987 --                          offset := LONGINT (IntMultiply (4, Num));
10  988 --                          result := Munger (Handle (fRowHeights), offset, Ptr (Ord (fRowHeights^) + offset),
10  989 --                              sizeof (LONGINT), dummy, 0);
```

```
10  990 --                          fNumOfRowChunks := fNumOfRowChunks - 1;
10  991 --
10  992 --                          {* see if we can consolidate chunks *}
10  993 --                          IF (Num > 0) & (fRowHeights^^[Num - 1].size = fRowHeights^^[Num].size) THEN
10  994 5-                          BEGIN
10  995 --                              fRowHeights^^[Num - 1].count := fRowHeights^^[Num - 1].count +
10  996 --                                                              fRowHeights^^[Num].count;
10  997 --                              {* need to delete that chunk *}
10  998 --                              offset := LONGINT (IntMultiply (4, Num));
10  999 --                              result := Munger (Handle (fRowHeights), offset, Ptr (Ord (fRowHeights^) + offset),
10 1000 --                                  sizeof (LONGINT), dummy, 0);
10 1001 --                              fNumOfRowChunks := fNumOfRowChunks - 1;
10 1002 -5                          END;
10 1003 -4                      END
10 1004 --                      ELSE
10 1005 4-                      BEGIN
10 1006 --                          Num := Num + 1;
10 1007 -4                      END;
10 1008 --                      Index := 1;
10 1009 -3                  END;
10 1010 -2              END;
10 1011 --
10 1012 --              fNumOfRows := fNumOfRows - numOfRows;
10 1013 --
10 1014 --              {* reset run array info *}
10 1015 --              fLastRow := 0;
10 1016 --              fLastRowChunkNum := 0;
10 1017 --              fLastRowTotal := 0;
10 1018 --              fLastRowIndex := 1;
10 1019 --
10 1020 --              AdjustSize;
10 1021 --
10 1022 --              InvalidVRect (aRect);
10 1023 -1          END;
10 1024 -0 A   END; {* TGridView.DelRowAt *}
10 1025 --
10 1026 --
10 1027 --      {*****************************************************************************}
10 1028 --      {*  TGridView.DelColFirst                                                    *}
10 1029 --      {*      Delete the first numOfCols.  This method causes a re-calculation     *}
10 1030 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1031 --      {*****************************************************************************}
10 1032 --      {$S GVNonRes}
10 1033 -- A   PROCEDURE TGridView.DelColFirst (numOfCols: INTEGER);
10 1034 0- A   BEGIN
10 1035 --          DelColAt (1, numOfCols);
10 1036 -0 A   END; {* TGridView.DelColFirst *}
10 1037 --
10 1038 --
10 1039 --      {*****************************************************************************}
10 1040 --      {*  TGridView.DelRowFirst                                                    *}
10 1041 --      {*      Delete the first numOfRows.  This method causes a re-calculation     *}
10 1042 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1043 --      {*****************************************************************************}
10 1044 --      {$S GVNonRes}
10 1045 -- A   PROCEDURE TGridView.DelRowFirst (numOfRows: INTEGER);
10 1046 0- A   BEGIN
10 1047 --          DelRowAt (1, numOfRows);
10 1048 -0 A   END; {* TGridView.DelRowFirst *}
10 1049 --
10 1050 --
10 1051 --      {*****************************************************************************}
10 1052 --      {*  TGridView.DelColLast                                                     *}
10 1053 --      {*      Delete the last numOfCols.  This method causes a re-calculation      *}
10 1054 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1055 --      {*****************************************************************************}
10 1056 --      {$S GVNonRes}
10 1057 -- A   PROCEDURE TGridView.DelColLast (numOfCols: INTEGER);
10 1058 0- A   BEGIN
10 1059 --          DelColAt (fNumOfCols - numOfCols + 1, numOfCols);
10 1060 -0 A   END; {* TGridView.DelColLast *}
10 1061 --
10 1062 --      {*****************************************************************************}
10 1063 --      {*  TGridView.DelRowLast                                                     *}
10 1064 --      {*      Delete the last numOfRows.  This method causes a re-calculation      *}
10 1065 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1066 --      {*****************************************************************************}
10 1067 --      {$S GVNonRes}
10 1068 -- A   PROCEDURE TGridView.DelRowLast (numOfRows: INTEGER);
10 1069 0- A   BEGIN
10 1070 --          DelRowAt (fNumOfRows - numOfRows + 1, numOfRows);
10 1071 -0 A   END; {* TGridView.DelRowLast *}
10 1072 --
10 1073 --
10 1074 --      {*****************************************************************************}
10 1075 --      {*  TGridView.EachCellDo                                                     *}
10 1076 --      {*      For each cell in the rectangle defined by startCell and stopCell,    *}
10 1077 --      {*      perform the procedure: DoToCell.  startCell must be <= stopCell or the *}
10 1078 --      {*      result is undefined.                                                 *}
10 1079 --      {*****************************************************************************}
10 1080 --      {$S GVRes}
10 1081 -- A   PROCEDURE TGridVIew.EachCellDo (startCell, stopCell: GridCell;
10 1082 --                                      PROCEDURE DoToCell (aCell: GridCell));
10 1083 --      VAR
10 1084 --          i,j:            INTEGER;
10 1085 --          firstRow:       INTEGER;
10 1086 --          lastRow:        INTEGER;
10 1087 --          firstCol:       INTEGER;
10 1088 --          lastCol:        INTEGER;
```

```
10 1089 --          aCell:          GridCell;
10 1090 --
10 1091 0- A    BEGIN
10 1092 --          firstRow := Max (1, startCell.v);
10 1093 --          firstCol := Max (1, startCell.h);
10 1094 --          lastRow  := Min (fNumOfRows, stopCell.v);
10 1095 --          lastCol  := Min (fNumOfCols, stopCell.h);
10 1096 --
10 1097 --      {$IFC qDebug}
10 1098 --          IF FALSE THEN
10 1099 1-          BEGIN
10 1100 --              WRITELN ('TGridView.EachCellDo');
10 1101 --              WRITELN ('firstCol: ', firstCol:2, ' lastCol: ', lastCol:2);
10 1102 --              WRITELN ('firstRow: ', firstRow:2, ' lastRow: ', lastRow:2);
10 1103 --              WRITELN;
10 1104 -1          END;
10 1105 --      {$ENDC}
10 1106 --
10 1107 --
10 1108 --          FOR j := firstRow TO lastRow DO
10 1109 1-          BEGIN
10 1110 --              FOR i := firstCol TO lastCol DO
10 1111 2-              BEGIN
10 1112 --                  aCell.v := j;
10 1113 --                  aCell.h := i;
10 1114 --                  DoToCell (aCell);
10 1115 -2              END;
10 1116 -1          END;
10 1117 -0 A    END; {* TGridView.EachCellDo *}
10 1118 --
10 1119 --
10 1120 --      {*****************************************************************************}
10 1121 --      {*  TGridView.EachSelectedCellDo                                             *}
10 1122 --      {*      For each cell in the selection region, perform the procedure: DoToCell. *}
10 1123 --      {*****************************************************************************}
10 1124 --      {$S GVRes}
10 1125 -- A    PROCEDURE TGridVIew.EachSelectedCellDo (PROCEDURE DoToCell (aCell: GridCell));
10 1126 0- A    BEGIN
10 1127 --          EachInRgn (fSelections, DoToCell);
10 1128 -0 A    END; {* TGridView.EachSelectedCellDo *}
10 1129 --
10 1130 --
10 1131 --      {*****************************************************************************}
10 1132 --      {*  TGridView.EachInRgn                                                      *}
10 1133 --      {*      For each cell in the specified region perform the procedure DoToCell.   *}
10 1134 --      {*      This is the method to use to step through the region containing the     *}
10 1135 --      {*      current selections.                                                  *}
10 1136 --      {*****************************************************************************}
10 1137 --      {$S GVRes}
10 1138 -- A    PROCEDURE TGridVIew.EachInRgn (aRgn: RgnHandle;
10 1139 --                                     PROCEDURE DoToCell (aCell: GridCell));
10 1140 --      VAR
10 1141 --          boundRect:      Rect;
10 1142 --
10 1143 -- B        PROCEDURE IfInRegion (aCell: GridCell);
10 1144 0- B        BEGIN
10 1145 --              IF PtInRgn (aCell, aRgn) THEN
10 1146 1-              BEGIN
10 1147 --                  DoToCell (aCell);
10 1148 -1              END;
10 1149 -0 B        END;
10 1150 --
10 1151 0- A    BEGIN
10 1152 --          boundRect := aRgn^^.rgnBBox;
10 1153 --          EachCellDo (boundRect.TopLeft, boundRect.BotRight, IfInRegion);
10 1154 -0 A    END; {* TGridView.EachInRgn *}
10 1155 --
10 1156 --
10 1157 --      {*****************************************************************************}
10 1158 --      {*  TGridView.FindColChunk                                                   *}
10 1159 --      {*      Chunks start at zero; Items start at one; that is, the               *}
10 1160 --      {*      third width in the second chunk is in aChunkNum: 1 with Index: 3.    *}
10 1161 --      {*****************************************************************************}
10 1162 --      {$S GVRes}
10 1163 -- A    FUNCTION  TGridView.FindColChunk (aCol: INTEGER; VAR aChunkNum: INTEGER;
10 1164 --                      VAR theTotal: LONGINT; VAR indexInChunk: INTEGER): BOOLEAN;
10 1165 --      VAR
10 1166 --          thisCol:    INTEGER;
10 1167 --          count:      INTEGER;
10 1168 --          deltaCols:  INTEGER;
10 1169 --
10 1170 0- A    BEGIN
10 1171 --          IF (fNumOfColChunks <= 0) | (aCol > fNumOfCols) | (aCol <= 0) THEN
10 1172 1-          BEGIN
10 1173 --              aChunkNum := 0;
10 1174 --              theTotal  := 0;
10 1175 --              indexInChunk := 0;
10 1176 --              FindColChunk := FALSE;
10 1177 --              aCol := 0;
10 1178 -1          END
10 1179 --          ELSE IF (aCol = fLastCol) THEN
10 1180 1-          BEGIN   {* check for the very easy case *}
10 1181 --              aChunkNum := fLastColChunkNum;
10 1182 --              theTotal  := fLastColTotal;
10 1183 --              indexInChunk := fLastColIndex;
10 1184 --              FindColChunk := TRUE;
10 1185 -1          END
10 1186 --          ELSE
10 1187 1-          BEGIN
```

```
10 1188 --              deltaCols := ABS (aCol - fLastCol);
10 1189 --
10 1190 --              IF (deltaCols >= aCol) | (aCol <= fColWidths^^[0].count) THEN
10 1191 2-              BEGIN   {* start from the first chunk *}
10 1192 --                  aChunkNum := 0;
10 1193 --                  theTotal  := 0;
10 1194 --                  thisCol   := 0;
10 1195 -2              END
10 1196 --              ELSE IF (deltaCols > (fNumOfCols - aCol + 1)) THEN
10 1197 2-              BEGIN   {* start from the end chunk *}
10 1198 --                  aChunkNum := fNumOfColChunks - 1;
10 1199 --                  count     := fColWidths^^[aChunkNum].count;
10 1200 --                  theTotal  := fMaxHorizontal -
10 1201 --                               IntMultiply (count, fColWidths^^[aChunkNum].size);
10 1202 --                  thisCol   := fNumOfCols - count;
10 1203 -2              END
10 1204 --              ELSE
10 1205 2-              BEGIN   {* start from the previous values *}
10 1206 --                  aChunkNum := fLastColChunkNum;
10 1207 --                  theTotal  := fLastColTotal;
10 1208 --                  thisCol   := fLastCol - fLastColIndex;
10 1209 -2              END;
10 1210 --
10 1211 --              IF aCol > thisCol THEN
10 1212 2-              BEGIN
10 1213 --                  WHILE ((thisCol + fColWidths^^[aChunkNum].count) < aCol) DO
10 1214 3-                  BEGIN
10 1215 --                      count     := fColWidths^^[aChunkNum].count;
10 1216 --                      theTotal  := theTotal + IntMultiply (count, fColWidths^^[aChunkNum].size);
10 1217 --                      thisCol   := thisCol + count;
10 1218 --                      aChunkNum := aChunkNum + 1;
10 1219 -3                  END;
10 1220 -2              END
10 1221 --              ELSE
10 1222 2-              BEGIN
10 1223 3-                  REPEAT
10 1224 --                      aChunkNum := aChunkNum - 1;
10 1225 --                      count     := fColWidths^^[aChunkNum].count;
10 1226 --                      theTotal  := theTotal - IntMultiply (count, fColWidths^^[aChunkNum].size);
10 1227 --                      thisCol   := thisCol - count;
10 1228 -3                  UNTIL (thisCol <= aCol);
10 1229 -2              END;
10 1230 --              indexInChunk := aCol - thisCol;
10 1231 --              FindColChunk := TRUE;
10 1232 -1          END;
10 1233 --          {* cache the last values *}
10 1234 --          fLastCol := aCol;
10 1235 --          fLastColChunkNum := aChunkNum;
10 1236 --          fLastColTotal := theTotal;
10 1237 --          fLastColIndex := indexInChunk;
10 1238 -0 A    END; {* TGridView.FindColChunk *}
10 1239 --
10 1240 --
10 1241 --      {*****************************************************************************}
10 1242 --      {*  TGridView.FindRowChunk                                                   *}
10 1243 --      {*      Chunks start at zero; Items start at one; that is, the               *}
10 1244 --      {*      third height in the second chunk is in aChunkNum: 1 with Index: 3.   *}
10 1245 --      {*****************************************************************************}
10 1246 --      {$S GVRes}
10 1247 -- A    FUNCTION  TGridView.FindRowChunk (aRow: INTEGER; VAR aChunkNum: INTEGER;
10 1248 --                  VAR theTotal: LONGINT; VAR indexInChunk: INTEGER): BOOLEAN;
10 1249 --      VAR
10 1250 --          thisRow:    INTEGER;
10 1251 --          count:      INTEGER;
10 1252 --          deltaRows:  INTEGER;
10 1253 --
10 1254 0- A    BEGIN
10 1255 --          IF (fNumOfRowChunks <= 0) | (aRow > fNumOfRows) | (aRow <= 0) THEN
10 1256 1-          BEGIN
10 1257 --              aChunkNum := 0;
10 1258 --              theTotal  := 0;
10 1259 --              indexInChunk := 0;
10 1260 --              FindRowChunk := FALSE;
10 1261 --              aRow := 0;
10 1262 -1          END
10 1263 --          ELSE IF (aRow = fLastRow) THEN
10 1264 1-          BEGIN   {* check for the very easy case *}
10 1265 --              aChunkNum := fLastRowChunkNum;
10 1266 --              theTotal  := fLastRowTotal;
10 1267 --              indexInChunk := fLastRowIndex;
10 1268 --              FindRowChunk := TRUE;
10 1269 -1          END
10 1270 --          ELSE
10 1271 1-          BEGIN
10 1272 --              deltaRows := ABS (aRow - fLastRow);
10 1273 --
10 1274 --              IF (deltaRows >= aRow) | (aRow <= fRowHeights^^[0].count) THEN
10 1275 2-              BEGIN   {* start from the first chunk *}
10 1276 --                  aChunkNum := 0;
10 1277 --                  theTotal  := 0;
10 1278 --                  thisRow   := 0;
10 1279 -2              END
10 1280 --              ELSE IF (deltaRows > (fNumOfRows - aRow + 1)) THEN
10 1281 2-              BEGIN   {* start from the end chunk *}
10 1282 --                  aChunkNum := fNumOfRowChunks - 1;
10 1283 --                  count     := fRowHeights^^[aChunkNum].count;
10 1284 --                  theTotal  := fMaxVertical -
10 1285 --                               IntMultiply (count, fRowHeights^^[aChunkNum].size);
10 1286 --                  thisRow   := fNumOfRows - count;
```

```
10 1287 -2              END
10 1288 --              ELSE
10 1289 2-              BEGIN   {* start from the previous values *}
10 1290 --                  aChunkNum := fLastRowChunkNum;
10 1291 --                  theTotal  := fLastRowTotal;
10 1292 --                  thisRow   := fLastRow - fLastRowIndex;
10 1293 -2              END;
10 1294 --
10 1295 --              IF aRow > thisRow THEN
10 1296 2-              BEGIN
10 1297 --                  WHILE ((thisRow + fRowHeights^^[aChunkNum].count) < aRow) DO
10 1298 3-                  BEGIN
10 1299 --                      count     := fRowHeights^^[aChunkNum].count;
10 1300 --                      theTotal  := theTotal + IntMultiply (count, fRowHeights^^[aChunkNum].size);
10 1301 --                      thisRow   := thisRow + count;
10 1302 --                      aChunkNum := aChunkNum + 1;
10 1303 -3                  END;
10 1304 -2              END
10 1305 --              ELSE
10 1306 2-              BEGIN
10 1307 3-                  REPEAT
10 1308 --                      aChunkNum := aChunkNum - 1;
10 1309 --                      count     := fRowHeights^^[aChunkNum].count;
10 1310 --                      theTotal  := theTotal - IntMultiply (count, fRowHeights^^[aChunkNum].size);
10 1311 --                      thisRow   := thisRow - count;
10 1312 -3                  UNTIL (thisRow <= aRow);
10 1313 -2              END;
10 1314 --              indexInChunk := aRow - thisRow;
10 1315 --              FindRowChunk := TRUE;
10 1316 -1          END;
10 1317 --          {* cache the last values *}
10 1318 --          fLastRow := aRow;
10 1319 --          fLastRowChunkNum := aChunkNum;
10 1320 --          fLastRowTotal := theTotal;
10 1321 --          fLastRowIndex := indexInChunk;
10 1322 -0 A   END; {* TGridView.FindRowChunk *}
10 1323 --
10 1324 --
10 1325 --      {*****************************************************************************}
10 1326 --      {*  TGridView.FirstSelectedCell                                              *}
10 1327 --      {*      Returns the top left cell in the selected range, if any              *}
10 1328 --      {*****************************************************************************}
10 1329 --      {$S GVRes}
10 1330 -- A   FUNCTION  TGridView.FirstSelectedCell: GridCell;
10 1331 --
10 1332 0- A   BEGIN
10 1333 --          FirstSelectedCell := fSelections^^.rgnBBox.topLeft;
10 1334 -0 A   END; {* TGridView.FirstSelectedCell *}
10 1335 --
10 1336 --
10 1337 --      {*****************************************************************************}
10 1338 --      {*  TGridView.GetColWidth                                                    *}
10 1339 --      {*      Get the column width for the given column. This includes insets.     *}
10 1340 --      {*****************************************************************************}
10 1341 --      {$S GVRes}
10 1342 -- A   FUNCTION  TGridView.GetColWidth (aCol: INTEGER): INTEGER;
10 1343 --      VAR
10 1344 --          Num:        INTEGER;
10 1345 --          theTotal:   LONGINT;
10 1346 --          Index:      INTEGER;
10 1347 --
10 1348 0- A   BEGIN
10 1349 --      {$IFC qRangeCheck}
10 1350 --          IF (aCol < 1) OR (aCol > fNumOfCols) THEN
10 1351 1-          BEGIN
10 1352 --      {$IFC qDebug}
10 1353 --              Writeln('fNumOfCols = ', fNumOfCols:1, '  aCol = ', aCol:1);
10 1354 --      {$ENDC}
10 1355 --              ProgramBreak('Range Check in GetColWidth');
10 1356 -1          END;
10 1357 --      {$ENDC}
10 1358 --
10 1359 --          IF fNumOfColChunks = 1 THEN
10 1360 --              GetColWidth := fColWidths^^[0].size
10 1361 --          ELSE IF FindColChunk (aCol, Num, theTotal, Index) THEN
10 1362 --              GetColWidth := fColWidths^^[Num].size
10 1363 --          ELSE
10 1364 --              GetColWidth := 0;
10 1365 -0 A   END; {* TGridView.GetColWidth *}
10 1366 --
10 1367 --
10 1368 --      {*****************************************************************************}
10 1369 --      {*  TGridView.GetRowHeight                                                   *}
10 1370 --      {*      Get the row height for the given row.  This includes insets.         *}
10 1371 --      {*****************************************************************************}
10 1372 --      {$S GVRes}
10 1373 -- A   FUNCTION TGridView.GetRowHeight (aRow: INTEGER) : INTEGER;
10 1374 --      VAR
10 1375 --          Num:        INTEGER;
10 1376 --          theTotal:   LONGINT;
10 1377 --          Index:      INTEGER;
10 1378 --
10 1379 0- A   BEGIN
10 1380 --      {$IFC qRangeCheck}
10 1381 --          IF (aRow < 1) | (aRow > fNumOfRows) THEN
10 1382 1-          BEGIN
10 1383 --      {$IFC qDebug}
10 1384 --              Writeln('fNumOfRows = ', fNumOfRows:1, '  aRow = ', aRow:1);
10 1385 --      {$ENDC}
```

```
10 1386 --              ProgramBreak('Range Check in GetRowHeight');
10 1387 -1          END;
10 1388 --      {$ENDC}
10 1389 --
10 1390 --          IF fNumOfRowChunks = 1 THEN
10 1391 --              GetRowHeight := fRowHeights^^[0].size
10 1392 --          ELSE IF FindRowChunk (aRow, Num, theTotal, Index) THEN
10 1393 --              GetRowHeight := fRowHeights^^[Num].size
10 1394 --          ELSE
10 1395 --              GetRowHeight := 0;
10 1396 -0 A    END; {* TGridView.GetRowHeight *}
10 1397 --
10 1398 --
10 1399 --      {*****************************************************************************}
10 1400 --      {*  TGridView.InvalidateCell                                                 *}
10 1401 --      {*      Mark a cell invlaid.  This forces a re-draw of the specified cell.   *}
10 1402 --      {*****************************************************************************}
10 1403 --      {$S GVRes}
10 1404 -- A    PROCEDURE TGridView.InvalidateCell (aCell: GridCell);
10 1405 --      VAR
10 1406 --          aRect: VRect;
10 1407 --
10 1408 0- A    BEGIN
10 1409 --          CellToVRect (aCell, aRect);
10 1410 --          InvalidVRect (aRect);
10 1411 -0 A    END; {* TGridView.InvalidateCell *}
10 1412 --
10 1413 --
10 1414 --      {*****************************************************************************}
10 1415 --      {*  TGridView.InvalidateSelection                                            *}
10 1416 --      {*      Mark the rectangle bounding all the selections invalid.              *}
10 1417 --      {*      This forces a re-draw of the enclosed cells.                         *}
10 1418 --      {*****************************************************************************}
10 1419 --      {$S GVRes}
10 1420 -- A    PROCEDURE TGridView.InvalidateSelection;
10 1421 --      VAR
10 1422 --          boundRect:  VRect;
10 1423 --          aRect:      VRect;
10 1424 --          BBox:       Rect;
10 1425 --
10 1426 0- A    BEGIN
10 1427 --          BBox := fSelections^^.rgnBBox;
10 1428 --
10 1429 --          BBox.bottom := BBox.bottom - 1;
10 1430 --          BBox.right  := BBox.right - 1;
10 1431 --          CellToVRect (BBox.topleft, boundRect);
10 1432 --          CellToVRect (BBox.botright, aRect);
10 1433 --
10 1434 --          boundRect.bottom := aRect.bottom;
10 1435 --          boundRect.right  := aRect.right;
10 1436 --
10 1437 --          InvalidVRect (boundRect);
10 1438 -0 A    END; {* TGridView.InvalidateSelection *}
10 1439 --
10 1440 --
10 1441 --      {*****************************************************************************}
10 1442 --      {*  TGridView.IdentifyPoint                                                  *}
10 1443 --      {*      Given a point in view coordinates, return the row and column and the *}
10 1444 --      {*      part of the view in which the point lies (inCell, inRow, inColumn,   *}
10 1445 --      {*      inVertex).                                                           *}
10 1446 --      {*****************************************************************************}
10 1447 --      {$S GVRes}
10 1448 -- A    FUNCTION TGridView.IdentifyPoint (theQDPoint: Point;
10 1449 --                                  VAR aRow, aCol: INTEGER): GridViewPart;
10 1450 --      VAR
10 1451 --          aRect:          VRect;
10 1452 --          aPoint:         VPoint;
10 1453 --          aCell:          GridCell;
10 1454 --
10 1455 0- A    BEGIN
10 1456 --          QDToViewPt (theQDPoint, aPoint);
10 1457 --          aCell := VPointToCell (aPoint);
10 1458 --          CellToVRect (aCell, aRect);
10 1459 --          InsetVRect (aRect, fColInset DIV 2, fRowInset DIV 2);
10 1460 --          aRow := aCell.v;
10 1461 --          aCol := aCell.h;
10 1462 --
10 1463 --          IF (aCell.v > fMaxVertical) | (aCell.h > fMaxHorizontal) |
10 1464 --              (aCell.v < 0) | (aCell.h < 0) THEN
10 1465 1-          BEGIN
10 1466 --              IdentifyPoint := badChoice;
10 1467 -1          END
10 1468 --          ELSE
10 1469 1-          BEGIN
10 1470 --              IdentifyPoint := inCell;
10 1471 --
10 1472 --              IF fColInset > 0 THEN
10 1473 2-              BEGIN
10 1474 --                  IF (aPoint.h < aRect.left) THEN
10 1475 3-                  BEGIN
10 1476 --                      IdentifyPoint := InColumn; { To the left of the cell }
10 1477 -3                  END
10 1478 --
10 1479 --                  ELSE IF (aPoint.h > aRect.right) THEN
10 1480 3-                  BEGIN
10 1481 --                      IdentifyPoint := InColumn; { To the right of the cell }
10 1482 --                      aCol := aCol + 1;
10 1483 -3                  END;
10 1484 -2              END;
```

```
10 1485 --
10 1486 --              IF fRowInset > 0 THEN
10 1487 2-              BEGIN
10 1488 --                  IF (aPoint.v < aRect.top) THEN
10 1489 3-                  BEGIN
10 1490 --                      IF (IdentifyPoint = InColumn) THEN
10 1491 --                          IdentifyPoint := InVertex { Click on both }
10 1492 --                      ELSE
10 1493 --                          IdentifyPoint := InRow; { Above the cell }
10 1494 -3                  END
10 1495 --
10 1496 --                  ELSE IF (aPoint.v > aRect.bottom) THEN
10 1497 3-                  BEGIN
10 1498 --                      IF (IdentifyPoint = InColumn) THEN
10 1499 --                          IdentifyPoint := InVertex { Click on both }
10 1500 --                      ELSE
10 1501 --                          IdentifyPoint := InRow; { Above the cell }
10 1502 --                      aRow := aRow + 1;
10 1503 -3                  END;
10 1504 -2              END;
10 1505 -1          END;
10 1506 -0 A    END; {* TGridView.IdentifyPoint *}
10 1507 --
10 1508 --
10 1509 --      {*****************************************************************************}
10 1510 --      {*  TGridView.InsColBefore                                                   *}
10 1511 --      {*      Insert numOfCols starting at aCol.  This method causes a re-calculation *}
10 1512 --      {*      of the extent and redraws only the necessary cells (and adornments).    *}
10 1513 --      {*****************************************************************************}
10 1514 --      {$S GVRes}
10 1515 -- A    PROCEDURE TGridView.InsColBefore (aCol :INTEGER; numOfCols: INTEGER; aWidth: INTEGER);
10 1516 --      VAR
10 1517 --          oldSize:    Size;
10 1518 --          offset:     LONGINT;
10 1519 --          result:     LONGINT;
10 1520 --          aNumber:    LONGINT;
10 1521 --          aRect:      VRect;
10 1522 --          Num:        INTEGER;
10 1523 --          theTotal:   LONGINT;
10 1524 --          Index:      INTEGER;
10 1525 --
10 1526 0- A    BEGIN
10 1527 --          IF (aCol < 1) | (numOfCols < 1) THEN
10 1528 1-          BEGIN
10 1529 --              IF numOfCols <> 0 THEN
10 1530 2-              BEGIN
10 1531 --      {$IFC qRangeCheck}
10 1532 --      {$IFC qDebug}
10 1533 --                  Writeln('fNumOfCols = ', fNumOfCols:1, '  aCol = ', aCol:1);
10 1534 --      {$ENDC}
10 1535 --                  ProgramBreak('Range Check in InsColBefore');
10 1536 --      {$ENDC}
10 1537 -2              END;
10 1538 -1          END
10 1539 --          ELSE
10 1540 1-          BEGIN
10 1541 --              {* Check if we can just increment the last size count *}
10 1542 --              IF (aCol > fNumOfCols) & (fNumOfColChunks > 0) &
10 1543 --                  (fColWidths^^[fNumOfColChunks-1].size = aWidth) THEN
10 1544 2-              BEGIN {* Just need to increment the ctr *}
10 1545 --                  fColWidths^^[fNumOfColChunks-1].count :=
10 1546 --                          fColWidths^^[fNumOfColChunks-1].count + numOfCols;
10 1547 -2              END
10 1548 --
10 1549 --              {* check if we can increment any size count *}
10 1550 --              ELSE IF FindColChunk (aCol, Num, theTotal, Index) &
10 1551 --                  (fColWidths^^[Num].size = aWidth) THEN
10 1552 2-              BEGIN
10 1553 --                  fColWidths^^[Num].count := fColWidths^^[Num].count + numOfCols;
10 1554 -2              END
10 1555 --
10 1556 --              {* We need to create a new chunk, possibly two *}
10 1557 --              ELSE
10 1558 2-              BEGIN
10 1559 --                  oldSize := GetHandleSize (Handle (fColWidths));
10 1560 --                  aNumber := 0;
10 1561 --
10 1562 --                  IF (Index <= 1) | (aCol > fNumOfCols) THEN
10 1563 3-                  BEGIN   {* need to add one chunk *}
10 1564 --                      IF (aCol > fNumOfCols) THEN
10 1565 4-                      BEGIN   {* add a row on the end *}
10 1566 --                          Num := fNumOfColChunks;
10 1567 -4                      END;
10 1568 --
10 1569 --                      offset := LONGINT (IntMultiply (Num, 4));
10 1570 --                      result := Munger (Handle (fColWidths) , offset, NIL, 0, Ptr(@aNumber),
10 1571 --                              Sizeof (LONGINT));
10 1572 --
10 1573 --                      IF GetHandleSize (Handle (fColWidths)) <= oldSize THEN
10 1574 --                          Failure (memFullErr, 0);
10 1575 --
10 1576 --                      fColWidths^^[Num].size := aWidth;
10 1577 --                      fColWidths^^[Num].count := numOfCols;
10 1578 --                      fNumOfColChunks := fNumOfColChunks + 1;
10 1579 -3                  END
10 1580 --                  ELSE
10 1581 3-                  BEGIN
10 1582 --                      {* need to add two *}
10 1583 --                      offset := LONGINT (IntMultiply (Num+1, 4));
```

```
10 1584 --                      result := Munger (Handle (fColWidths) , offset, NIL, 0, Ptr(@aNumber),
10 1585 --                              2 * Sizeof (LONGINT));
10 1586 --
10 1587 --                      IF GetHandleSize (Handle (fColWidths)) <= oldSize THEN
10 1588 --                          Failure (memFullErr, 0);
10 1589 --
10 1590 --                      fColWidths^^[Num+2].count  := fColWidths^^[Num].count - Index + 1;
10 1591 --                      fColWidths^^[Num+2].size := fColWidths^^[Num].size;
10 1592 --                      fColWidths^^[Num].count := Index - 1;
10 1593 --                      fColWidths^^[Num+1].size := aWidth;
10 1594 --                      fColWidths^^[Num+1].count := numOfCols;
10 1595 --                      fNumOfColChunks := fNumOfColChunks + 2;
10 1596 -3                  END;
10 1597 -2              END;
10 1598 --
10 1599 --              fNumOfCols := fNumOfCols + numOfCols;
10 1600 --              fMaxHorizontal := fMaxHorizontal + IntMultiply (numOfCols, aWidth);
10 1601 --              AdjustSize;
10 1602 --
10 1603 --              ColToVRect (Max (1, aCol), Max (1, fNumOfCols - aCol + 1), aRect);
10 1604 --              InvalidVRect (aRect);
10 1605 -1          END;
10 1606 -0 A    END; {* TGridView.InsColBefore *}
10 1607 --
10 1608 --
10 1609 --      {*****************************************************************************}
10 1610 --      {*  TGridView.InsRowBefore                                                   *}
10 1611 --      {*      Insert numOfRows starting at aRow.  This method causes a re-calculation *}
10 1612 --      {*      of the extent and redraws only the necessary cells (and adornments).    *}
10 1613 --      {*****************************************************************************}
10 1614 --      {$S GVRes}
10 1615 -- A    PROCEDURE TGridView.InsRowBefore (aRow: INTEGER; numOfRows: INTEGER; aHeight: INTEGER);
10 1616 --      VAR
10 1617 --          oldSize:    Size;
10 1618 --          offset:     LONGINT;
10 1619 --          result:     LONGINT;
10 1620 --          aNumber:    LONGINT;
10 1621 --          aRect:      VRect;
10 1622 --          Num:        INTEGER;
10 1623 --          theTotal:   LONGINT;
10 1624 --          Index:      INTEGER;
10 1625 --
10 1626 0- A    BEGIN
10 1627 --          IF (aRow < 1) | (numOfRows < 1) THEN
10 1628 1-          BEGIN
10 1629 --              IF numOfRows <> 0 THEN
10 1630 2-              BEGIN
10 1631 --      {$IFC qRangeCheck}
10 1632 --      {$IFC qDebug}
10 1633 --                  Writeln('fNumOfRows = ', fNumOfRows:1, '  aRow = ', aRow:1);
10 1634 --      {$ENDC}
10 1635 --                  ProgramBreak('Range Check in InsRowBefore');
10 1636 --      {$ENDC}
10 1637 -2              END;
10 1638 -1          END
10 1639 --          ELSE
10 1640 1-          BEGIN
10 1641 --              {* Check if we can just increment the last size count *}
10 1642 --              IF (aRow > fNumOfRows) & (fNumOfRowChunks > 0) &
10 1643 --                  (fRowHeights^^[fNumOfRowChunks-1].size = aHeight) THEN
10 1644 2-              BEGIN {* Just need to increment the ctr *}
10 1645 --                  fRowHeights^^[fNumOfRowChunks-1].count :=
10 1646 --                      fRowHeights^^[fNumOfRowChunks-1].count + numOfRows;
10 1647 -2              END
10 1648 --
10 1649 --              {* figure out if we can increment any chunk count... *}
10 1650 --              ELSE IF FindRowChunk (aRow, Num, theTotal, Index) &
10 1651 --                  (fRowHeights^^[Num].size = aHeight) THEN
10 1652 2-              BEGIN
10 1653 --                  fRowHeights^^[Num].count := fRowHeights^^[Num].count + numOfRows;
10 1654 -2              END
10 1655 --
10 1656 --              {* We need to create a new chunk, possibly two *}
10 1657 --              ELSE
10 1658 2-              BEGIN
10 1659 --                  oldSize := GetHandleSize (Handle (fRowHeights));
10 1660 --                  aNumber := 0;
10 1661 --
10 1662 --                  IF (Index <= 1) | (aRow > fNumOfRows) THEN
10 1663 3-                  BEGIN   {* need to add one chunk *}
10 1664 --                      IF (aRow > fNumOfRows) THEN
10 1665 4-                      BEGIN   {* add a row on the end *}
10 1666 --                          Num := fNumOfRowChunks;
10 1667 -4                      END;
10 1668 --
10 1669 --                      offset := LONGINT (IntMultiply (Num, 4));
10 1670 --                      result := Munger (Handle (fRowHeights) , offset, NIL, 0, Ptr(@aNumber),
10 1671 --                              Sizeof (LONGINT));
10 1672 --
10 1673 --                      IF GetHandleSize (Handle (fRowHeights)) <= oldSize THEN
10 1674 --                          Failure (memFullErr, 0);
10 1675 --
10 1676 --                      fRowHeights^^[Num].size := aHeight;
10 1677 --                      fRowHeights^^[Num].count := numOfRows;
10 1678 --                      fNumOfRowChunks := fNumOfRowChunks + 1;
10 1679 -3                  END
10 1680 --                  ELSE
10 1681 3-                  BEGIN
10 1682 --                      {* need to add two *}
```

```
10 1683 --                      offset := LONGINT (IntMultiply (Num+1, 4));
10 1684 --                      result := Munger (Handle (fRowHeights) , offset, NIL, 0, Ptr(@aNumber),
10 1685 --                              2 * Sizeof (LONGINT));
10 1686 --
10 1687 --                      IF GetHandleSize (Handle (fRowHeights)) <= oldSize THEN
10 1688 --                          Failure (memFullErr, 0);
10 1689 --
10 1690 --                      fRowHeights^^[Num+2].count  := fRowHeights^^[Num].count - Index + 1;
10 1691 --                      fRowHeights^^[Num+2].size := fRowHeights^^[Num].size;
10 1692 --                      fRowHeights^^[Num].count := Index - 1;
10 1693 --                      fRowHeights^^[Num+1].size := aHeight;
10 1694 --                      fRowHeights^^[Num+1].count := numOfRows;
10 1695 --                      fNumOfRowChunks := fNumOfRowChunks + 2;
10 1696 -3                  END;
10 1697 -2              END;
10 1698 --
10 1699 --              fNumOfRows := fNumOfRows + numOfRows;
10 1700 --              fMaxVertical := fMaxVertical + IntMultiply (numOfRows, aHeight);
10 1701 --              AdjustSize;
10 1702 --
10 1703 --              RowToVRect (Max (1, aRow), Max (1, fNumOfRows - aRow + 1), aRect);
10 1704 --              InvalidVRect (aRect);
10 1705 -1          END;
10 1706 -0 A    END; {* TGridView.InsRowBefore *}
10 1707 --
10 1708 --
10 1709 --      {*****************************************************************************}
10 1710 --      {*  TGridView.InsColLast                                                     *}
10 1711 --      {*      Insert numOfCols at the end.  This method causes a re-calculation    *}
10 1712 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1713 --      {*****************************************************************************}
10 1714 --      {$S GVRes}
10 1715 -- A    PROCEDURE TGridView.InsColLast (numOfCols: INTEGER; aWidth: INTEGER);
10 1716 0- A    BEGIN
10 1717 --          InsColBefore (fNumOfCols + 1, numOfCols, aWidth);
10 1718 -0 A    END; {* TGridView.InsColLast *}
10 1719 --
10 1720 --
10 1721 --      {*****************************************************************************}
10 1722 --      {*  TGridView.InsRowLast                                                     *}
10 1723 --      {*      Insert numOfRows at the end.  This method causes a re-calculation    *}
10 1724 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1725 --      {*****************************************************************************}
10 1726 --      {$S GVRes}
10 1727 -- A    PROCEDURE TGridView.InsRowLast (numOfRows: INTEGER; aHeight: INTEGER);
10 1728 0- A    BEGIN
10 1729 --          InsRowBefore (fNumOfRows + 1, numOfRows, aHeight);
10 1730 -0 A    END; {* TGridView.InsRowLast *}
10 1731 --
10 1732 --
10 1733 --      {*****************************************************************************}
10 1734 --      {*  TGridView.InsColFirst                                                    *}
10 1735 --      {*      Insert numOfCols at the start.  This method causes a re-calculation  *}
10 1736 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1737 --      {*****************************************************************************}
10 1738 --      {$S GVRes}
10 1739 -- A    PROCEDURE TGridView.InsColFirst (numOfCols: INTEGER; aWidth: INTEGER);
10 1740 0- A    BEGIN
10 1741 --          InsColBefore (kOneCol, numOfCols, aWidth);
10 1742 -0 A    END; {* TGridView.InsColFirst *}
10 1743 --
10 1744 --
10 1745 --      {*****************************************************************************}
10 1746 --      {*  TGridView.InsRowFirst                                                    *}
10 1747 --      {*      Insert numOfRows at the start.  This method causes a re-calculation  *}
10 1748 --      {*      of the extent and redraws only the necessary cells (and adornments). *}
10 1749 --      {*****************************************************************************}
10 1750 --      {$S GVRes}
10 1751 -- A    PROCEDURE TGridView.InsRowFirst (numOfRows: INTEGER; aHeight: INTEGER);
10 1752 0- A    BEGIN
10 1753 --          InsRowBefore (kOneRow, numOfRows, aHeight);
10 1754 -0 A    END; {* TGridView.InsRowFirst *}
10 1755 --
10 1756 --
10 1757 --      {*****************************************************************************}
10 1758 --      {*  TGridView.IsCellSelected                                                 *}
10 1759 --      {*      Check if the specified cell is currently selected.                   *}
10 1760 --      {*****************************************************************************}
10 1761 --      {$S GVRes}
10 1762 -- A    FUNCTION  TGridView.IsCellSelected (aCell: GridCell) : BOOLEAN;
10 1763 0- A    BEGIN
10 1764 --          IsCellSelected := PtInRgn (aCell, fSelections);
10 1765 -0 A    END; {* TGridView.IsCellSelected *}
10 1766 --
10 1767 --
10 1768 --      {*****************************************************************************}
10 1769 --      {*  TGridView.LastSelectedCell                                               *}
10 1770 --      {*      Returns the bottom right cell in the selected range, if any          *}
10 1771 --      {*****************************************************************************}
10 1772 --      {$S GVRes}
10 1773 -- A    FUNCTION  TGridView.LastSelectedCell: GridCell;
10 1774 0- A    BEGIN
10 1775 --          LastSelectedCell := fSelections^^.rgnBBox.botRight;
10 1776 -0 A    END; {* TGridView.LastSelectedCell *}
10 1777 --
10 1778 --
10 1779 --      {*****************************************************************************}
10 1780 --      {*  TGridView.RowToVRect                                                     *}
10 1781 --      {*      Calculate the bounding rectangle for the given row(s).  The left and *}
```

```
10 1782 --      {*      right sides of the rectangle are the same as the view.                  *}
10 1783 --      {*      The rectangle includes the cells in that row and the row inset.         *}
10 1784 --      {*****************************************************************************}
10 1785 --      {$S GVRes}
10 1786 -- A    PROCEDURE TGridView.RowToVRect (aRow: INTEGER; numOfRows: INTEGER; VAR aRect: VRect);
10 1787 --      VAR
10 1788 --          height:     LONGINT;
10 1789 --          Num:        INTEGER;
10 1790 --          theTotal:   LONGINT;
10 1791 --          Index:      INTEGER;
10 1792 --          i:          INTEGER;
10 1793 --
10 1794 0- A    BEGIN
10 1795 --          IF (aRow < 1) | (numOfRows < 1) | (aRow + numOfRows - 1 > fNumOfRows) THEN
10 1796 1-          BEGIN
10 1797 --      {$IFC qRangeCheck}
10 1798 --      {$IFC qDebug}
10 1799 --              Writeln('fNumOfRows = ', fNumOfRows:1, '  aRow = ', aRow:1);
10 1800 --      {$ENDC}
10 1801 --              ProgramBreak('Range Check in RowToVRect');
10 1802 --      {$ENDC}
10 1803 --              SetVRect (aRect, 0, 0, 0, 0);
10 1804 -1          END
10 1805 --          ELSE    {* all the params look OK *}
10 1806 1-          BEGIN
10 1807 --              height := 0;
10 1808 --
10 1809 --              IF (fNumOfRowChunks = 1) THEN  {* only one row height *}
10 1810 2-              BEGIN
10 1811 --                  num := 0;
10 1812 --                  theTotal := IntMultiply (fRowHeights^^[num].size, aRow-1);
10 1813 --                  height := IntMultiply (numOfRows, fRowHeights^^[num].size);
10 1814 -2              END
10 1815 --              ELSE
10 1816 2-              BEGIN
10 1817 --                  IF FindRowChunk (aRow, Num, theTotal, Index) THEN
10 1818 3-                  BEGIN
10 1819 --                      theTotal := theTotal + IntMultiply ((Index-1), fRowHeights^^[Num].size);
10 1820 --                      For i := aRow TO (aRow + numOfRows - 1) DO
10 1821 --                          height := height + GetRowHeight (i);
10 1822 -3                  END;
10 1823 -2              END;
10 1824 --
10 1825 --              WITH aRect DO
10 1826 2-              BEGIN
10 1827 --                  top    := theTotal;
10 1828 --                  left   := 0;
10 1829 --                  bottom := theTotal + height;
10 1830 --                  right  := fMaxHorizontal;
10 1831 -2              END;
10 1832 -1          END;
10 1833 -0 A    END; {* TGridView.RowToVRect *}
10 1834 --
10 1835 --
10 1836 --      {*****************************************************************************}
10 1837 --      {*  TGridView.SameCell                                                       *}
10 1838 --      {*      Check if the cells are the same.                                     *}
10 1839 --      {*****************************************************************************}
10 1840 --      {$S GVRes}
10 1841 -- A    FUNCTION TGridView.SameCell (aCell, bCell: GridCell): BOOLEAN;
10 1842 0- A    BEGIN
10 1843 --          SameCell := EqualPt (aCell, bCell);
10 1844 -0 A    END; {* TGridView.SameCell *}
10 1845 --
10 1846 --
10 1847 --      {*****************************************************************************}
10 1848 --      {*  TGridView.ScrollSelectionIntoView                                        *}
10 1849 --      {*      Scroll the selection (or a piece of it) into view.                   *}
10 1850 --      {*****************************************************************************}
10 1851 --      {$S GVRes}
10 1852 -- A    PROCEDURE TGridView.ScrollSelectionIntoView (redraw: BOOLEAN);
10 1853 --      VAR
10 1854 --          botRightCell:       GridCell;
10 1855 --          topLeftRect,
10 1856 --          botRightRect,
10 1857 --          selectionRect:      VRect;
10 1858 --          minToSee:           Point;
10 1859 0- A    BEGIN
10 1860 --          IF NOT (EmptyRgn (fSelections)) THEN
10 1861 1-          BEGIN
10 1862 --              CellToVRect(fSelections^^.rgnBBox.topLeft, topLeftRect);
10 1863 --              SetPt(botRightCell, fSelections^^.rgnBBox.right - 1,
10 1864 --                                  fSelections^^.rgnBBox.bottom - 1);
10 1865 --              CellToVRect(botRightCell, botRightRect);
10 1866 --              UnionVRect(topLeftRect, botRightRect, selectionRect);
10 1867 --              minToSee.v := Max(topLeftRect.bottom - topLeftRect.top,
10 1868 --                                botRightRect.bottom - botRightRect.top);
10 1869 --              minToSee.h := Max(topLeftRect.right - topLeftRect.left,
10 1870 --                                botRightRect.right - botRightRect.left);
10 1871 --              RevealRect (selectionRect, minToSee, redraw);
10 1872 -1          END;
10 1873 -0 A    END; {* TGridView.ScrollSelectionIntoView *}
10 1874 --
10 1875 --
10 1876 --      {*****************************************************************************}
10 1877 --      {*  TGridView.SetColWidth                                                    *}
10 1878 --      {*      Set the width of the specified column to aWidth.  This includes      *}
10 1879 --      {*      column insets.                                                       *}
10 1880 --      {*****************************************************************************}
```

```
10 1881 --      {$S GVNonRes}
10 1882 -- A    PROCEDURE TGridView.SetColWidth (aCol: INTEGER; numOfCols: INTEGER; aWidth: INTEGER);
10 1883 0- A    BEGIN
10 1884 --      {$IFC qRangeCheck}
10 1885 --          IF (aCol < 1) | (numOfCols < 1) | (aCol + numOfCols - 1 > fNumOfCols) THEN
10 1886 1-              BEGIN
10 1887 --      {$IFC qDebug}
10 1888 --              Writeln('fNumOfCols = ', fNumOfCols:1, '  aCol = ', aCol:1);
10 1889 --      {$ENDC}
10 1890 --              ProgramBreak('Range Check in SetColHeight');
10 1891 -1              END;
10 1892 --      {$ENDC}
10 1893 --
10 1894 --          IF NOT ((fNumOfColChunks = 1) & (fColWidths^^[0].size  = aWidth)) THEN
10 1895 1-          BEGIN
10 1896 --              DelColAt (aCol, numOfCols);
10 1897 --              InsColBefore (aCol, numOfCols, aWidth);
10 1898 -1          END;
10 1899 -0 A    END; {* TGridView.SetColWidth *}
10 1900 --
10 1901 --
10 1902 --      {*****************************************************************************}
10 1903 --      {*  TGridView.SetRowHeight                                                   *}
10 1904 --      {*      Set the height of the specified row to Height.  This includes        *}
10 1905 --      {*      column insets.                                                       *}
10 1906 --      {*****************************************************************************}
10 1907 --      {$S GVNonRes}
10 1908 -- A    PROCEDURE TGridView.SetRowHeight (aRow: INTEGER; numOfRows: INTEGER; aHeight: INTEGER);
10 1909 0- A    BEGIN
10 1910 --      {$IFC qRangeCheck}
10 1911 --          IF (aRow < 1) | (numOfRows < 1) | (aRow + numOfRows - 1 > fNumOfRows) THEN
10 1912 1-              BEGIN
10 1913 --      {$IFC qDebug}
10 1914 --              Writeln('fNumOfRows = ', fNumOfRows:1, '  aRow = ', aRow:1);
10 1915 --      {$ENDC}
10 1916 --              ProgramBreak('Range Check in SetRowHeight');
10 1917 -1              END;
10 1918 --      {$ENDC}
10 1919 --
10 1920 --          IF NOT ((fNumOfRowChunks = 1) & (fRowHeights^^[0].size  = aHeight)) THEN
10 1921 1-          BEGIN
10 1922 --              DelRowAt (aRow, numOfRows);
10 1923 --              InsRowBefore (aRow, numOfRows, aHeight);
10 1924 -1          END;
10 1925 -0 A    END; {* TGridView.SetRowHeight *}
10 1926 --
10 1927 --
10 1928 --      {*****************************************************************************}
10 1929 --      {*  TGridView.SelectCell                                                     *}
10 1930 --      {*      Set the current selection to the specified cell and call SetSelection.  *}
10 1931 --      {*****************************************************************************}
10 1932 --      {$S GVRes}
10 1933 -- A    PROCEDURE TGridView.SelectCell (theCell: GridCell; extend, highlight, select: BOOLEAN);
10 1934 0- A    BEGIN
10 1935 --          {* setup the region and then call SetSelection *}
10 1936 --          IF (theCell.h = 0) | (theCell.v = 0) THEN
10 1937 --              SetEmptyRgn (fTmpSelRgn)
10 1938 --          ELSE
10 1939 --              SetRectRgn (fTmpSelRgn, theCell.h, theCell.v, theCell.h + 1, theCell.v + 1);
10 1940 --
10 1941 --          fWholeRect := TRUE;
10 1942 --          SetSelection (fTmpSelRgn, extend, highlight, select);
10 1943 -0 A    END; {* TGridView.SelectCell *}
10 1944 --
10 1945 --
10 1946 --      {*****************************************************************************}
10 1947 --      {*  TGridView.SetEmptySelection                                              *}
10 1948 --      {*      Set the current selection to be empty (nothing). "highlight" causes  *}
10 1949 --      {*      re-draw.                                                             *}
10 1950 --      {*****************************************************************************}
10 1951 --      {$S GVRes}
10 1952 -- A    PROCEDURE TGridView.SetEmptySelection (highlight: BOOLEAN);
10 1953 0- A    BEGIN
10 1954 --          SetEmptyRgn (fTmpSelRgn);
10 1955 --          SetSelection (fTmpSelRgn, kDontExtend, highlight, kSelect);
10 1956 --          fWholeRect := TRUE;
10 1957 -0 A    END; {* TGridView.SetEmptySelection *}
10 1958 --
10 1959 --
10 1960 --      {*****************************************************************************}
10 1961 --      {*  TGridView.SetSelection                                                   *}
10 1962 --      {*      Set the current selections to the specified region.  If extend is    *}
10 1963 --      {*      kExtend, theRegion is "added" to that already selected. If highlight *}
10 1964 --      {*      is kHighlight, theRegion is highlighted ("highlight" causes an       *}
10 1965 --      {*      update to the screen.)                                               *}
10 1966 --      {*                                                                           *}
10 1967 --      {*      The field fWholeRect is used to determine if we need to do hiliteing *}
10 1968 --      {*      cell by cell, or for a whole rectangle of cells.  If you call this   *}
10 1969 --      {*      method explicitly, then be sure to set fWholeRect appropriately before *}
10 1970 --      {*      calling SetSelection.  fLastWholeRect is used to determine if un-    *}
10 1971 --      {*      hiliteing can be done by rectangle.  It is used and set by this method *}
10 1972 --      {*      only!   fWHoleRect = FALSE will work properly for ALL cases, but     *}
10 1973 --      {*      will be unnecessarily slow for the rectangle cases!                  *}
10 1974 --      {*                                                                           *}
10 1975 --      {*      The select parameter is used to select or de-select theRegion.  If   *}
10 1976 --      {*      select is kDeSelect then extend is ignored and hilight is used to    *}
10 1977 --      {*      determine is an update is necessary.                                 *}
10 1978 --      {*                                                                           *}
10 1979 --      {*      I assume here that if SetSelection is called explicitly (not by the  *}
```

```
10 1980 --      {*      mouse-tracking code) then you know what you're doing in regards to      *}
10 1981 --      {*      single selections i.e. there is no checking to enforce single selection!*}
10 1982 --      {*****************************************************************************}
10 1983 --      {$S GVRes}
10 1984 -- A    PROCEDURE TGridView.SetSelection (theRegion: RgnHandle;
10 1985 --                                        extend, highlight, select: BOOLEAN);
10 1986 --      VAR
10 1987 --          hlRect:     Rect;
10 1988 --          hlOffRect:  Rect;
10 1989 --          hlDesired:  HLState;
10 1990 --
10 1991 0- A    BEGIN
10 1992 --          IF highlight THEN
10 1993 1-          BEGIN
10 1994 --              {* get the ones to turn on *}
10 1995 --              CreateHighlightRgn (theRegion, fHLRegion, fWholeRect);
10 1996 --
10 1997 --              {* figure out what we need to turn off *}
10 1998 --              CreateHighlightRgn (fSelections, fTmpRgn, fLastWholeRect);
10 1999 --              CopyRgn (fTmpRgn, fTmpHLRgn);
10 2000 --
10 2001 --              IF fHLDesired = hlDim THEN
10 2002 --                  hlDesired := hlOff;
10 2003 --
10 2004 --              IF (NOT extend & select) THEN   {* we need to turn off the old ones *}
10 2005 2-              BEGIN
10 2006 --                  DiffRgn (fTmpRgn, fHLRegion, fTmpRgn);
10 2007 --                  DoHilite (fTmpRgn, fHLDesired, hlOFF);
10 2008 -2              END;
10 2009 --
10 2010 --              IF select THEN
10 2011 2-              BEGIN
10 2012 --                  DiffRgn (fHLRegion, fTmpHLRgn, fHLRegion);
10 2013 --                  DoHilite (fHLRegion, hlOFF, fHLDesired);
10 2014 -2              END
10 2015 --              ELSE
10 2016 2-              BEGIN
10 2017 --                  SectRgn (fHLRegion, fTmpHLRgn, fHLRegion);
10 2018 --                  DoHilite (fHLRegion, fHLDesired, hlOFF);
10 2019 -2              END;
10 2020 -1          END;
10 2021 --
10 2022 --          IF extend & select THEN {* extend the selection region *}
10 2023 --              UnionRgn (theRegion, fSelections, fSelections)
10 2024 --          ELSE IF select THEN     {* reset the selection region *}
10 2025 --              CopyRgn (theRegion, fSelections)
10 2026 --          ELSE    {* need to de-select the new region *}
10 2027 1-          BEGIN
10 2028 --              fWholeRect := FALSE;
10 2029 --              DiffRgn (fSelections, theRegion, fSelections);
10 2030 -1          END;
10 2031 --
10 2032 --          CopyRgn (fSelections, fHLRegion);
10 2033 --          fLastWholeRect := fWholeRect;
10 2034 -0 A    END; {* TGridView.SetSelection *}
10 2035 --
10 2036 --
10 2037 --      {*****************************************************************************}
10 2038 --      {*  TGridView.SetSelRect                                                     *}
10 2039 --      {*      Set the current selections to the specified rectangle.  If extend    *}
10 2040 --      {*      is kExtend  then the rectangle is "added" to that already selected,  *}
10 2041 --      {*      if highlight is kHighlight then the rectangle is highlighted as well.   *}
10 2042 --      {*****************************************************************************}
10 2043 --      {$S GVRes}
10 2044 -- A    PROCEDURE TGridView.SetSelRect (theTop, theLeft, theBottom, theRight: INTEGER;
10 2045 --                                      extend, highlight, select: BOOLEAN);
10 2046 0- A    BEGIN
10 2047 --          SetRectRgn (fTmpSelRgn, theLeft, theTop, theRight, theBottom);
10 2048 --          fWholeRect := TRUE;  {* we know the selection is a rectangle *}
10 2049 --          SetSelection (fTmpSelRgn, extend, highlight, select);
10 2050 -0 A    END; {* TGridView.SetSelRect *}
10 2051 --
10 2052 --
10 2053 --      {*****************************************************************************}
10 2054 --      {*  TGridView.SetSingleSelection                                             *}
10 2055 --      {*      If TRUE then only one item/cell can be selected at a time.           *}
10 2056 --      {*****************************************************************************}
10 2057 --      {$S GVNonRes}
10 2058 -- A    PROCEDURE TGridView.SetSingleSelection (theSetting: BOOLEAN);
10 2059 0- A    BEGIN
10 2060 --          fSingleSelection := theSetting;
10 2061 --          fWholeRect := TRUE;  {* we know the selection is a rectangle *}
10 2062 -0 A    END; {* TGridView.SetSingleSelection *}
10 2063 --
10 2064 --
10 2065 --      {*****************************************************************************}
10 2066 --      {*  TGridView.VPointToCell                                                   *}
10 2067 --      {*      Get the Cell in which the specified point lies.  Points in the insets   *}
10 2068 --      {*      of cells will be considered as inside that cell.                     *}
10 2069 --      {*      This method returns the cell (0,0) if the point is not within a cell.   *}
10 2070 --      {*****************************************************************************}
10 2071 --      {$S GVRes}
10 2072 -- A    FUNCTION  TGridView.VPointToCell (aPoint: VPoint): GridCell;
10 2073 --      VAR
10 2074 --          i:      INTEGER;
10 2075 --          aCell:  GridCell;
10 2076 0- A    BEGIN
10 2077 --          {* Decide what cell is indicated by this point.     *}
10 2078 --          IF aPoint.h >= fMaxHorizontal THEN
```

```
10 2079 --              aPoint.h := fMaxHorizontal - 1;
10 2080 --
10 2081 --          IF aPoint.v >= fMaxVertical THEN
10 2082 --              aPoint.v := fMaxVertical - 1;
10 2083 --
10 2084 --          IF fNumOfColChunks = 1 THEN   {* only one column height *}
10 2085 1-          BEGIN
10 2086 --              IF fColWidths^^[0].size = 0 THEN
10 2087 --                  aCell.h := 1
10 2088 --              ELSE
10 2089 --                  aCell.h := (aPoint.h DIV fColWidths^^[0].size) + 1;
10 2090 -1          END
10 2091 --          ELSE
10 2092 1-          BEGIN
10 2093 --              i := 0;
10 2094 --              aCell.h := 0;
10 2095 --              aPoint.h := aPoint.h + 1;
10 2096 --              WHILE aPoint.h > 0 DO
10 2097 2-              BEGIN
10 2098 --                  aPoint.h := aPoint.h -
10 2099 --                              (fColWidths^^[i].size * fColWidths^^[i].count);
10 2100 --                  aCell.h := aCell.h + fColWidths^^[i].count;
10 2101 --                  i := i + 1;
10 2102 -2              END;
10 2103 --              IF fColWidths^^[i-1].size <> 0 THEN
10 2104 --                  aCell.h := aCell.h - (ABS (aPoint.h) DIV fColWidths^^[i-1].size);
10 2105 -1          END;
10 2106 --
10 2107 --          IF fNumOfRowChunks = 1 THEN   {* only one row height *}
10 2108 1-          BEGIN
10 2109 --              IF fRowHeights^^[0].size = 0 THEN
10 2110 --                  aCell.v := 1
10 2111 --              ELSE
10 2112 --                  aCell.v := (aPoint.v DIV fRowHeights^^[0].size) + 1;
10 2113 -1          END
10 2114 --          ELSE
10 2115 1-          BEGIN
10 2116 --              i := 0;
10 2117 --              aCell.v := 0;
10 2118 --              aPoint.v := aPoint.v + 1;
10 2119 --              WHILE aPoint.v > 0 DO
10 2120 2-              BEGIN
10 2121 --                  aPoint.v := aPoint.v -
10 2122 --                              (fRowHeights^^[i].size * fRowHeights^^[i].count);
10 2123 --                  aCell.v := aCell.v + fRowHeights^^[i].count;
10 2124 --                  i := i + 1;
10 2125 -2              END;
10 2126 --              IF fRowHeights^^[i-1].size <> 0 THEN
10 2127 --                  aCell.v := aCell.v - (ABS (aPoint.v) DIV fRowHeights^^[i-1].size);
10 2128 -1          END;
10 2129 --          VPointToCell := aCell;
10 2130 -0 A   END; {* TGridView.VPointToCell *}
10 2131 --
10 2132 --
10 2133 --      {*****************************************************************************}
10 2134 --      {*  TGridView.Fields - for DEBUGGING                                         *}
10 2135 --      {*****************************************************************************}
10 2136 --      {$IFC qDebug}
10 2137 --      {$S GVFields}
10 2138 -- A    PROCEDURE TGridView.Fields (PROCEDURE DoToField (fieldName: Str255;
10 2139 --                                                      fieldAddr: Ptr;
10 2140 --                                                      fieldType: INTEGER)); OVERRIDE;
10 2141 0- A   BEGIN
10 2142 --          DoToField ('TGridView', NIL, bClass);
10 2143 --          DoToField ('fSelections', @fSelections, bHandle);
10 2144 --          DoToField ('fHLRegion', @fHLRegion, bHandle);
10 2145 --          DoToField ('fTmpSelRgn', @fTmpSelRgn, bHandle);
10 2146 --          DoToField ('fTmpRgn', @fTmpRgn, bHandle);
10 2147 --          DoToField ('fTmpHLRgn', @fTmpHLRgn, bHandle);
10 2148 --          DoToField ('fNumOfRows', @fNumOfRows, bInteger);
10 2149 --          DoToField ('fRowHeights', @fRowHeights, bHandle);
10 2150 --          DoToField ('fNumOfCols', @fNumOfCols, bInteger);
10 2151 --          DoToField ('fColWidths', @fColWidths, bHandle);
10 2152 --          DoToField ('fMaxHorizontal', @fMaxHorizontal, bLongInt);
10 2153 --          DoToField ('fMaxVertical', @fMaxVertical, bLongInt);
10 2154 --          DoToField ('fRowInset', @fRowInset, bInteger);
10 2155 --          DoToField ('fColInset', @fColInset, bInteger);
10 2156 --          DoToField ('fAdornRows', @fAdornRows, bBoolean);
10 2157 --          DoToField ('fAdornCols', @fAdornCols, bBoolean);
10 2158 --          DoToField ('fSingleSelection', @fSingleSelection, bBoolean);
10 2159 --          DoToField ('fAllowDimSelection', @fAllowDimSelection, bBoolean);
10 2160 --          DoToField ('fWholeRect', @fWholeRect, bBoolean);
10 2161 --          DoToField ('fLastWholeRect', @fLastWholeRect, bBoolean);
10 2162 --          DoToField ('fNumOfRowChunks', @fNumOfRowChunks, bInteger);
10 2163 --          DoToField ('fLastRow', @fLastRow, bInteger);
10 2164 --          DoToField ('fLastRowChunkNum', @fLastRowChunkNum, bInteger);
10 2165 --          DoToField ('fLastRowTotal', @fLastRowTotal, bLongInt);
10 2166 --          DoToField ('fLastRowIndex', @fLastRowIndex, bInteger);
10 2167 --          DoToField ('fNumOfColChunks', @fNumOfColChunks, bInteger);
10 2168 --          DoToField ('fLastCol', @fLastCol, bInteger);
10 2169 --          DoToField ('fLastColChunkNum', @fLastColChunkNum, bInteger);
10 2170 --          DoToField ('fLastColTotal', @fLastColTotal, bLongInt);
10 2171 --          DoToField ('fLastColIndex', @fLastColIndex, bInteger);
10 2172 --
10 2173 --          INHERITED Fields(DoToField);
10 2174 -0 A   END; {* TGridView.Fields *}
10 2175 --      {$ENDC qDebug}
10 2176 --
10 2177 --
```

```
10 2178 --      {*****************************************************************************}
10 2179 --      {*****************************************************************************}
10 2180 --      {*  TTextGridView Object Sub-Class                                           *}
10 2181 --      {*      This sub-class inplements a text-only version of the gridview.  The  *}
10 2182 --      {*      user must override the GetText routine and we'll take care of the rest! *}
10 2183 --      {*      TextGridView calculates the height and width of rows and columns     *}
10 2184 --      {*      based on the font and the string length.  Right now, the text in a cell *}
10 2185 --      {*      can be only one line.  (OVERRIDE the DrawCell method to do something a  *}
10 2186 --      {*      little slicker.)  If you don't want the original height or width,    *}
10 2187 --      {*      individual rows and columns can be changed by calling SetColWidth and  *}
10 2188 --      {*      SetRowHeight.                                                        *}
10 2189 --      {*****************************************************************************}
10 2190 --      {*****************************************************************************}
10 2191 --
10 2192 --
10 2193 --      {*****************************************************************************}
10 2194 --      {*  TTextGridView.ITextGridView                                              *}
10 2195 --      {*      Initialize the TextGridView Object.                                  *}
10 2196 --      {*****************************************************************************}
10 2197 --      {$IFC qProceduralViews}
10 2198 --      {$S GVOpen}
10 2199 -- A    PROCEDURE TTextGridView.ITextGridView (
10 2200 --          itsDocument:        TDocument;  {* Its document                  *}
10 2201 --          itsSuperView:       TView;      {* Its parent view               *}
10 2202 --          itsLocation:        VPoint;     {* Top, Left in parent's coords  *}
10 2203 --          numOfRows:          INTEGER;    {* Number of rows initially      *}
10 2204 --          numOfCols:          INTEGER;    {* Number of columns initially   *}
10 2205 --          colWidth:           INTEGER;    {* Width of items in the columns *}
10 2206 --          adornRows:          BOOLEAN;    {* Adornment for Rows?           *}
10 2207 --          adornCols:          BOOLEAN;    {* Adornment for Columns?        *}
10 2208 --          rowInset:           INTEGER;    {* horizontal space between cells *}
10 2209 --          colInset:           INTEGER;    {* vertical   space between cells *}
10 2210 --          singleSelection:    BOOLEAN;    {* single cell selection?        *}
10 2211 --          itsTextStyle:       TextStyle); {* size, color, etc. font info   *}
10 2212 0- A    BEGIN
10 2213 --          fTextStyle := itsTextStyle;
10 2214 --
10 2215 --          SetUpFont;
10 2216 --
10 2217 --          IGridView (itsDocument, itsSuperView, itsLocation, numOfRows, numOfCols,
10 2218 --                     fLineHeight + rowInset, colWidth, adornRows, adornCols,
10 2219 --                     rowInset, colInset, singleSelection);
10 2220 --
10 2221 --          IF (fNumOfCols = 1) & (fSizeDeterminer[h] <> sizeFixed) &
10 2222 --              (GetColWidth (1) = 0) & (fSuperView <> NIL) THEN
10 2223 --              SetColWidth (1, fNumOfCols, fSuperView.fSize.h);
10 2224 -0 A    END; {* TTextGridView.ITextGridView *}
10 2225 --      {$ENDC qProceduralViews}
10 2226 --
10 2227 --
10 2228 --      {*****************************************************************************}
10 2229 --      {*  TTextGridView.IRes                                                       *}
10 2230 --      {*      Initialize the GridView for use with view templates.                 *}
10 2231 --      {*****************************************************************************}
10 2232 --      {$IFC qTemplateViews}
10 2233 --      {$S GVOpen}
10 2234 -- A    PROCEDURE TTextGridView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
10 2235 --      VAR
10 2236 --          itsTextStyle:       TextStyle;
10 2237 --
10 2238 0- A    BEGIN
10 2239 --          INHERITED IRes (itsDocument, itsSuperView, itsParams);
10 2240 --
10 2241 --          WITH TextGridViewTemplatePtr (itsParams)^ DO
10 2242 1-          BEGIN
10 2243 --
10 2244 --              SetTextStyle (itsTextStyle, GetFontNum (itsFontName), itsFontFace,
10 2245 --                            itsFontSize, itsFontColor);
10 2246 --
10 2247 --              fTextStyle := itsTextStyle;
10 2248 --              SetUpFont;
10 2249 -1          END;
10 2250 --
10 2251 --          IF fNumOfRows > 0 THEN
10 2252 --              SetRowHeight (1, fNumOfRows, fLineHeight + fRowInset);
10 2253 --
10 2254 --          IF (fNumOfCols = 1) & (fSizeDeterminer[h] <> sizeFixed) & (GetColWidth (1) = 0) THEN
10 2255 1-          BEGIN
10 2256 --                SetColWidth (1, fNumOfCols, fSize.h);
10 2257 -1          END;
10 2258 --
10 2259 --          OffsetPtrWStr (itsParams, SIZEOF (TextGridViewTemplate));
10 2260 -0 A    END; {* TTextGridView.IRes *}
10 2261 --      {$ENDC qTemplateViews}
10 2262 --
10 2263 --
10 2264 --      {*****************************************************************************}
10 2265 --      {*  TTextGridView.WRes                                                       *}
10 2266 --      {*      Write the resource file.                                             *}
10 2267 --      {*****************************************************************************}
10 2268 --      {$IFC qWriteTemplates}
10 2269 --      {$S GVNonRes}
10 2270 -- A    PROCEDURE TTextGridView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
10 2271 --
10 2272 --      VAR
10 2273 --          theSize:    INTEGER;
10 2274 --          theFont:    Str255;
10 2275 --          tgPtr:      TextGridViewTemplatePtr;
10 2276 --
```

```
10 2277 0- A   BEGIN
10 2278 --         INHERITED WRes(theResource, itsParams);
10 2279 --
10 2280 --         theSize := fTextStyle.tsSize;
10 2281 --         GetPortFontInfo(fTextStyle.tsFont, theFont, theSize);
10 2282 --
10 2283 --         tgPtr := TextGridViewTemplatePtr(ExpandPtrWStr(theResource, itsParams,
10 2284 --           SIZEOF(TextGridViewTemplate), LENGTH(theFont)));
10 2285 --
10 2286 --         WITH tgPtr^, fTextStyle DO
10 2287 1-             BEGIN
10 2288 --             itsFontFace := tsFace;
10 2289 --             itsFontSize := theSize;
10 2290 --             itsFontColor := tsColor;
10 2291 --             (* itsFontName := theFont; *)
10 2292 -1              END;
10 2293 --         CopyStr255(theFont, PRStr(tgPtr^.itsFontName));
10 2294 -0 A   END; {* TTextGridView.WRes *}
10 2295 --
10 2296 --
10 2297 --     {*****************************************************************************}
10 2298 --     {*  TTextGridView.WriteRes                                                   *}
10 2299 --     {*      Set up for writing the resource file                                 *}
10 2300 --     {*****************************************************************************}
10 2301 --     {$S GVNonRes}
10 2302 -- A   PROCEDURE TTextGridView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
10 2303 0- A   BEGIN
10 2304 --         gWResSignature := 'txtg';   gWResType := 'TTextGridView';
10 2305 --         WRes(theResource, itsParams);
10 2306 -0 A   END; {* TTextGridView.WriteRes *}
10 2307 --     {$ENDC qWriteTemplates}
10 2308 --
10 2309 --
10 2310 --     {*****************************************************************************}
10 2311 --     {*  TTextGridView.DrawCell                                                   *}
10 2312 --     {*      Get the text that needs to be drawn and draw it.                     *}
10 2313 --     {*****************************************************************************}
10 2314 --     {$S GVRes}
10 2315 -- A   PROCEDURE TTextGridView.DrawCell(aCell: GridCell; aQDRect: Rect); OVERRIDE;
10 2316 --     VAR
10 2317 --         theText:    Str255;
10 2318 0- A   BEGIN
10 2319 --         GetText (aCell, theText);
10 2320 --
10 2321 --         IF (GetColWidth (aCell.h) > 0) THEN
10 2322 1-         BEGIN
10 2323 --             MoveTo (aQDRect.left, aQDRect.top + fLineAscent);
10 2324 --             DrawString (theText);
10 2325 -1          END;
10 2326 -0 A   END; {* TTextGridView.DrawCell *}
10 2327 --
10 2328 --
10 2329 --     {*****************************************************************************}
10 2330 --     {*  TTextGridView.Focus                                                      *}
10 2331 --     {*      Set the pen and font when focus'ing.                                 *}
10 2332 --     {*****************************************************************************}
10 2333 --     {$S GVRes}
10 2334 -- A   FUNCTION TTextGridView.Focus : BOOLEAN; OVERRIDE;
10 2335 0- A   BEGIN
10 2336 --         IF INHERITED Focus THEN
10 2337 1-         BEGIN
10 2338 --             SetPen;
10 2339 --             Focus := TRUE;
10 2340 -1         END
10 2341 --         ELSE
10 2342 --             Focus := FALSE;
10 2343 -0 A   END; {* TTextGridView.Focus *}
10 2344 --
10 2345 --
10 2346 --     {*****************************************************************************}
10 2347 --     {*  TTextGridView.SetUpFont                                                  *}
10 2348 --     {*      Sets the font for this view.                                         *}
10 2349 --     {*****************************************************************************}
10 2350 --     {$S GVOpen}
10 2351 -- A   PROCEDURE TTextGridView.SetUpFont;
10 2352 --     VAR
10 2353 --         theFontInfo:    FontInfo;
10 2354 --
10 2355 0- A   BEGIN
10 2356 --         SetPen;
10 2357 --         GetFontInfo (theFontInfo);
10 2358 --         WITH theFontInfo DO
10 2359 1-         BEGIN
10 2360 --             fLineHeight := ascent + descent + leading;
10 2361 --             fLineAscent := ascent + (leading DIV 2);
10 2362 -1          END;
10 2363 -0 A   END; {* TTextGridView.SetUpFont *}
10 2364 --
10 2365 --
10 2366 --     {*****************************************************************************}
10 2367 --     {*  TTextGridView.SetPen                                                     *}
10 2368 --     {*      Sets the font characteristics of the current port.  Called at        *}
10 2369 --     {*      the beginning of the Draw method.                                    *}
10 2370 --     {*****************************************************************************}
10 2371 --     {$S GVRes}
10 2372 -- A   PROCEDURE TTextGridView.SetPen;
10 2373 --     VAR
10 2374 --         itsTextStyle: TextStyle;
10 2375 --
```

```
10 2376 0- A   BEGIN
10 2377 --         itsTextStyle := fTextStyle;
10 2378 --         SetPortTextStyle (itsTextStyle);
10 2379 --         PenNormal;
10 2380 -0 A   END; {* TTextGridView.SetPen *}
10 2381 --
10 2382 --
10 2383 --     {*************************************************************************}
10 2384 --     {*  TTextGridView.GetText                                                *}
10 2385 --     {*      This method gets the text to be drawn in a given cell.  IT MUST BE *}
10 2386 --     {*      OVERRIDDEN !!                                                    *}
10 2387 --     {*************************************************************************}
10 2388 --     {$S GVRes}
10 2389 -- A   PROCEDURE TTextGridView.GetText (aCell: GridCell; VAR aString: Str255);
10 2390 0- A   BEGIN
10 2391 --         {* MUST be overridden! *}
10 2392 --
10 2393 --     {$IFC qDebug}
10 2394 --         WRITELN ('TTextGridView: GetText MUST be OVERRIDDEN!');
10 2395 --     {$ENDC qDebug}
10 2396 -0 A   END; {* TTextGridView.GetText *}
10 2397 --
10 2398 --
10 2399 --     {*************************************************************************}
10 2400 --     {*  TTextGridView.Fields                                                 *}
10 2401 --     {*      For debugging support of the inspector.                          *}
10 2402 --     {*************************************************************************}
10 2403 --     {$IFC qDebug}
10 2404 --     {$S GVFields}
10 2405 -- A   PROCEDURE TTextGridView.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 2406 --                 fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 2407 0- A   BEGIN
10 2408 --         DoToField ('TTextGridView', NIL, bClass);
10 2409 --         DoToField ('fLineHeight', @fLineHeight, bInteger);
10 2410 --         DoToField ('fLineAscent', @fLineAscent, bInteger);
10 2411 --         {$H-}
10 2412 --         TextStyleFields ('fTextStyle', fTextStyle, DoToField);
10 2413 --         {$H+}
10 2414 --
10 2415 --         INHERITED Fields (DoToField);
10 2416 -0 A   END; {* TTextGridView.Fields *}
10 2417 --     {$ENDC qDebug}
10 2418 --
10 2419 --
10 2420 --     {*************************************************************************}
10 2421 --     {*************************************************************************}
10 2422 --     {*  TTextListView Object Sub-Class                                       *}
10 2423 --     {*      This sub-class inplements a single column version of TextGridView. The  *}
10 2424 --     {*      user must override the GetItemText routine and the rest is done!         *}
10 2425 --     {*      TextListView calculates the height and width of items                    *}
10 2426 --     {*      based on the font and the string length.  Right now, the text in a cell *}
10 2427 --     {*      can be only one line.  (OVERRIDE the DrawCell method to do something a  *}
10 2428 --     {*      little slicker.)  If you don't want the original height, individual     *}
10 2429 --     {*      items can be changed by calling SetItemHeight.  The width of view can   *}
10 2430 --     {*      be modified by calling SetItemWidth.                                    *}
10 2431 --     {*************************************************************************}
10 2432 --     {*************************************************************************}
10 2433 --
10 2434 --
10 2435 --     {*************************************************************************}
10 2436 --     {*  TTextListView.ITextListView                                          *}
10 2437 --     {*      Initialize the TextListView object.                              *}
10 2438 --     {*************************************************************************}
10 2439 --     {$IFC qProceduralViews}
10 2440 --     {$S GVOpen}
10 2441 -- A   PROCEDURE TTextListView.ITextListView (
10 2442 --         itsDocument:        TDocument;      {* Its document                  *}
10 2443 --         itsSuperView:       TView;          {* Its parent view               *}
10 2444 --         itsLocation:        VPoint;         {* Top, Left in parent's coords  *}
10 2445 --         numOfItems:         INTEGER;        {* Number of items initially     *}
10 2446 --         adornRows:          BOOLEAN;        {* Draw the row adornments?      *}
10 2447 --         adornCols:          BOOLEAN;        {* Draw the col adornment?       *}
10 2448 --         rowInset:           INTEGER;        {* Amount to inset the rows      *}
10 2449 --         colInset:           INTEGER;        {* Amount to inset the column    *}
10 2450 --         singleSelection:    BOOLEAN;        {* single cell selection?        *}
10 2451 --         itsTextStyle:       TextStyle);     {* size, color, etc. font info   *}
10 2452 --
10 2453 --     CONST
10 2454 --         kOneColumn = 1;
10 2455 --
10 2456 0- A   BEGIN
10 2457 --         ITextGridView (itsDocument, itsSuperView, itsLocation, numOfItems, kOneColumn,
10 2458 --                        0, adornRows, adornCols, rowInset, colInset, singleSelection,
10 2459 --                        itsTextStyle);
10 2460 --
10 2461 --         fSizeDeterminer[h] := sizeRelSuperView;
10 2462 -0 A   END; {* TTextListView.ITextListView *}
10 2463 --     {$ENDC qProceduralViews}
10 2464 --
10 2465 --
10 2466 --     {*************************************************************************}
10 2467 --     {*  TTextListView.WriteRes                                               *}
10 2468 --     {*      Set up for writing the resource file                             *}
10 2469 --     {*************************************************************************}
10 2470 --     {$IFC qWriteTemplates}
10 2471 --     {$S GVNonRes}
10 2472 -- A   PROCEDURE TTextListView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
10 2473 0- A   BEGIN
10 2474 --         gWResSignature := 'lstg';   gWResType := 'TTextListView';
```

```
10 2475 --          WRes(theResource, itsParams);
10 2476 -0 A    END; {* TTextListView.WriteRes *}
10 2477 --      {$ENDC qWriteTemplates}
10 2478 --
10 2479 --
10 2480 --      {*****************************************************************************}
10 2481 --      {*  TTextListView.AllItemsDo                                                 *}
10 2482 --      {*      For all the items execute "DoToItem".                                *}
10 2483 --      {*****************************************************************************}
10 2484 --      {$S GVRes}
10 2485 -- A    PROCEDURE TTextListView.AllItemsDo (PROCEDURE DoToItem (anItem: INTEGER));
10 2486 0- A    BEGIN
10 2487 --          EachItemDo (1, fNumOfRows, DoToItem);
10 2488 -0 A    END; {* TTextListView.AllItemsDo *}
10 2489 --
10 2490 --
10 2491 --      {*****************************************************************************}
10 2492 --      {*  TTextListView.CanSelectCell                                              *}
10 2493 --      {*      Figure out if the cell can be selected.                              *}
10 2494 --      {*****************************************************************************}
10 2495 --      {$S GVRes}
10 2496 -- A    FUNCTION  TTextListView.CanSelectCell (aCell: GridCell) : BOOLEAN; OVERRIDE;
10 2497 0- A    BEGIN
10 2498 --          CanSelectCell := CanSelectItem (aCell.v);
10 2499 -0 A    END; {* TTextListView.CanSelectCell *}
10 2500 --
10 2501 --
10 2502 --      {*****************************************************************************}
10 2503 --      {*  TTextListView.CanSelectItem                                              *}
10 2504 --      {*      Figure out if the item can be selected.                              *}
10 2505 --      {*****************************************************************************}
10 2506 --      {$S GVRes}
10 2507 -- A    FUNCTION TTextListView.CanSelectItem (anItem: INTEGER) : BOOLEAN;
10 2508 0- A    BEGIN
10 2509 --          CanSelectItem := TRUE;
10 2510 -0 A    END; {* TTextListView.CanSelectItem *}
10 2511 --
10 2512 --
10 2513 --      {*****************************************************************************}
10 2514 --      {*  TTextListView.DelItemAt                                                  *}
10 2515 --      {*      Delete numOfItems starting at anItem.  This method causes a          *}
10 2516 --      {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2517 --      {*      (and adornments).                                                    *}
10 2518 --      {*****************************************************************************}
10 2519 --      {$S GVNonRes}
10 2520 -- A    PROCEDURE TTextListView.DelItemAt (anItem: INTEGER; numOfItems: INTEGER);
10 2521 0- A    BEGIN
10 2522 --          DelRowAt (anItem, numOfItems);
10 2523 -0 A    END; {* TTextListView.DelItemAt *}
10 2524 --
10 2525 --
10 2526 --      {*****************************************************************************}
10 2527 --      {*  TTextListView.DelItemFirst                                               *}
10 2528 --      {*      Delete numOfItems at the beginning.  This method causes a            *}
10 2529 --      {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2530 --      {*      (and adornments).                                                    *}
10 2531 --      {*****************************************************************************}
10 2532 --      {$S GVNonRes}
10 2533 -- A    PROCEDURE TTextListView.DelItemFirst (numOfItems: INTEGER);
10 2534 0- A    BEGIN
10 2535 --          DelItemAt (1, numOfItems);
10 2536 -0 A    END; {* TTextListView.DelItemFirst *}
10 2537 --
10 2538 --
10 2539 --      {*****************************************************************************}
10 2540 --      {*  TTextListView.DelItemLast                                                *}
10 2541 --      {*      Delete numOfItems at the end.  This method causes a                  *}
10 2542 --      {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2543 --      {*      (and adornments).                                                    *}
10 2544 --      {*****************************************************************************}
10 2545 --      {$S GVNonRes}
10 2546 -- A    PROCEDURE TTextListView.DelItemLast (numOfItems: INTEGER);
10 2547 0- A    BEGIN
10 2548 --          DelItemAt (fNumOfRows, numOfItems);
10 2549 -0 A    END; {* TTextListView.DelItemLast *}
10 2550 --
10 2551 --
10 2552 --      {*****************************************************************************}
10 2553 --      {*  TTextListView.EachItemDo                                                 *}
10 2554 --      {*      For each item from start to stop execute "DoToItem".                 *}
10 2555 --      {*****************************************************************************}
10 2556 --      {$S GVRes}
10 2557 -- A    PROCEDURE TTextListView.EachItemDo (start, stop: INTEGER;
10 2558 --                                          PROCEDURE DoToItem (anItem: INTEGER));
10 2559 --      VAR
10 2560 --          i:  INTEGER;
10 2561 --
10 2562 0- A    BEGIN
10 2563 --          FOR i := start TO stop DO
10 2564 --              DoToItem (i);
10 2565 -0 A    END; {* TTextListView.EachItemDo *}
10 2566 --
10 2567 --
10 2568 --      {*****************************************************************************}
10 2569 --      {*  TTextListView.EachSelectedItemDo                                         *}
10 2570 --      {*      For each item from start to stop execute "DoToItem".                 *}
10 2571 --      {*****************************************************************************}
10 2572 --      {$S GVRes}
10 2573 -- A    PROCEDURE TTextListView.EachSelectedItemDo (PROCEDURE DoToItem (anItem: INTEGER));
```

```
10 2574 -- B       PROCEDURE DoToCell (aCell: GridCell);
10 2575 0- B       BEGIN
10 2576 --             DoToItem (aCell.v);
10 2577 -0 B       END;
10 2578 --
10 2579 0- A   BEGIN
10 2580 --         EachInRgn (fSelections, DoToCell);
10 2581 -0 A   END; {* TTextListView.EachSelectedItemDo *}
10 2582 --
10 2583 --
10 2584 --     {*****************************************************************************}
10 2585 --     {*  TTextListView.GetItemHeight                                              *}
10 2586 --     {*      Get the height of the specified item.                                *}
10 2587 --     {*****************************************************************************}
10 2588 --     {$S GVRes}
10 2589 -- A   FUNCTION  TTextListView.GetItemHeight (anItem: INTEGER) : INTEGER;
10 2590 0- A   BEGIN
10 2591 --         GetItemHeight := GetRowHeight (anItem);
10 2592 -0 A   END; {* TTextListView.GetItemHeight *}
10 2593 --
10 2594 --
10 2595 --     {*****************************************************************************}
10 2596 --     {*  TTextListView.GetItemWidth                                               *}
10 2597 --     {*      Get the width of the view.  (All the items have the same width!)     *}
10 2598 --     {*****************************************************************************}
10 2599 --     {$S GVRes}
10 2600 -- A   FUNCTION  TTextListView.GetItemWidth: INTEGER;
10 2601 0- A   BEGIN
10 2602 --         GetItemWidth := GetColWidth (1);
10 2603 -0 A   END; {* TTextListView.GetItemWidth *}
10 2604 --
10 2605 --
10 2606 --     {*****************************************************************************}
10 2607 --     {*  TTextListView.GetItemText                                                *}
10 2608 --     {*      Get the text for the specified item.  THIS METHOD MUST BE OVERRIDDEN!!  *}
10 2609 --     {*****************************************************************************}
10 2610 --     {$S GVRes}
10 2611 -- A   PROCEDURE TTextListView.GetItemText (anItem: INTEGER; VAR aString: Str255);
10 2612 0- A   BEGIN
10 2613 --         {* MUST be overridden !!! *}
10 2614 --
10 2615 --     {$IFC qDebug}
10 2616 --         WRITELN ('TTextListView: GetItemText MUST be OVERRIDDEN!');
10 2617 --     {$ENDC qDebug}
10 2618 --
10 2619 -0 A   END; {* TTextListView.GetItemText *}
10 2620 --
10 2621 --
10 2622 --     {*****************************************************************************}
10 2623 --     {*  TTextListView.GetText                                                    *}
10 2624 --     {*      Figure out which item it is and get the text.                        *}
10 2625 --     {*****************************************************************************}
10 2626 --     {$S GVRes}
10 2627 -- A   PROCEDURE TTextListView.GetText (aCell: GridCell; VAR aString: Str255); OVERRIDE;
10 2628 0- A   BEGIN
10 2629 --         GetItemText (aCell.v, aString);
10 2630 -0 A   END; {* TTextListView.GetText *}
10 2631 --
10 2632 --
10 2633 --     {*****************************************************************************}
10 2634 --     {*  TTextListView.InsItemBefore                                              *}
10 2635 --     {*      Insert numOfItems starting at anItem.  This method causes a          *}
10 2636 --     {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2637 --     {*      (and adornments).                                                    *}
10 2638 --     {*****************************************************************************}
10 2639 --     {$S GVRes}
10 2640 -- A   PROCEDURE TTextListView.InsItemBefore (anItem: INTEGER; numOfItems: INTEGER);
10 2641 0- A   BEGIN
10 2642 --         InsRowBefore (anItem, numOfItems, fLineHeight + fRowInset);
10 2643 -0 A   END; {* TTextListView.InsItemBefore *}
10 2644 --
10 2645 --
10 2646 --     {*****************************************************************************}
10 2647 --     {*  TTextListView.InsItemFirst                                               *}
10 2648 --     {*      Insert numOfItems at the beginning.  This method causes a            *}
10 2649 --     {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2650 --     {*      (and adornments).                                                    *}
10 2651 --     {*****************************************************************************}
10 2652 --     {$S GVRes}
10 2653 -- A   PROCEDURE TTextListView.InsItemFirst (numOfItems: INTEGER);
10 2654 0- A   BEGIN
10 2655 --         InsItemBefore (1, numOfItems);
10 2656 -0 A   END; {* TTextListView.InsItemFirst *}
10 2657 --
10 2658 --
10 2659 --     {*****************************************************************************}
10 2660 --     {*  TTextListView.InsItemLast                                                *}
10 2661 --     {*      Insert numOfItems at the end.  This method causes a                  *}
10 2662 --     {*      re-calculation of the extent and redraws only the necessary cells    *}
10 2663 --     {*      (and adornments).                                                    *}
10 2664 --     {*****************************************************************************}
10 2665 --     {$S GVRes}
10 2666 -- A   PROCEDURE TTextListView.InsItemLast (numOfItems: INTEGER);
10 2667 0- A   BEGIN
10 2668 --         InsItemBefore (fNumOfRows + 1, numOfItems);
10 2669 -0 A   END; {* TTextListView.InsItemLast *}
10 2670 --
10 2671 --
10 2672 --     {*****************************************************************************}
```

```
10 2673 --      {*  TTextListView.IsItemSelected                                          *}
10 2674 --      {*      Check if the specified item is currently selected.               *}
10 2675 --      {*************************************************************************}
10 2676 --      {$S GVRes}
10 2677 -- A    FUNCTION  TTextListView.IsItemSelected (anItem: INTEGER) : BOOLEAN;
10 2678 --      VAR
10 2679 --          aCell:  GridCell;
10 2680 --
10 2681 0- A    BEGIN
10 2682 --          aCell.h := 1;
10 2683 --          aCell.v := anItem;
10 2684 --          IsItemSelected := IsCellSelected (aCell);
10 2685 -0 A    END; {* TTextListView.IsItemSelected *}
10 2686 --
10 2687 --
10 2688 --      {*************************************************************************}
10 2689 --      {*  TTextListView.Resize                                                 *}
10 2690 --      {*      Resize the view.                                                 *}
10 2691 --      {*************************************************************************}
10 2692 --      {$S GVNonRes}
10 2693 -- A    PROCEDURE TTextListView.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
10 2694 0- A    BEGIN
10 2695 --          INHERITED Resize (width, height, invalidate);
10 2696 --          IF fNumOfCols = 1 THEN
10 2697 1-          BEGIN
10 2698 --              fMaxHorizontal := fMaxHorizontal - fColWidths^^[0].size + width;
10 2699 --              fColWidths^^[0].size := width;          { We know there's only 1 column }
10 2700 -1          END;
10 2701 -0 A    END; {* TTextListView.Resize *}
10 2702 --
10 2703 --
10 2704 --      {*************************************************************************}
10 2705 --      {*  TTextListView.SelectCell                                             *}
10 2706 --      {*      Get the correct item number and call SelectItem.                 *}
10 2707 --      {*************************************************************************}
10 2708 --      {$S GVRes}
10 2709 -- A    PROCEDURE TTextListView.SelectCell (theCell: GridCell;
10 2710 --                                         extend, highlight, select: BOOLEAN); OVERRIDE;
10 2711 0- A    BEGIN
10 2712 --          SelectItem (theCell.v, extend, highlight, select);
10 2713 -0 A    END; {* TTextListView.SelectCell *}
10 2714 --
10 2715 --
10 2716 --      {*************************************************************************}
10 2717 --      {*  TTextListView.SelectItem                                             *}
10 2718 --      {*      Selects a single item in the view.                               *}
10 2719 --      {*************************************************************************}
10 2720 --      {$S GVRes}
10 2721 -- A    PROCEDURE TTextListView.SelectItem (anItem: INTEGER; extend, highlight, select: BOOLEAN);
10 2722 --      VAR
10 2723 --          aCell:  GridCell;
10 2724 0- A    BEGIN
10 2725 --          aCell.h := 1;
10 2726 --          aCell.v := anItem;
10 2727 --
10 2728 --          INHERITED SelectCell (aCell, extend, highlight, select);
10 2729 -0 A    END; {* TTextListView.SelectItem *}
10 2730 --
10 2731 --
10 2732 --      {*************************************************************************}
10 2733 --      {*  TTextListView.SetItemHeight                                          *}
10 2734 --      {*      Set the height of the specified item.                            *}
10 2735 --      {*************************************************************************}
10 2736 --      {$S GVNonRes}
10 2737 -- A    PROCEDURE TTextListView.SetItemHeight (anItem: INTEGER; numOfItems: INTEGER;
10 2738 --                          aHeight: INTEGER);
10 2739 0- A    BEGIN
10 2740 --          SetRowHeight (anItem, numOfItems, aHeight);
10 2741 -0 A    END; {* TTextListView.SetItemHeight *}
10 2742 --
10 2743 --
10 2744 --      {*************************************************************************}
10 2745 --      {*  TTextListView.SetItemWidth                                           *}
10 2746 --      {*      Set the width of the view.  (All the items have the same width!) *}
10 2747 --      {*************************************************************************}
10 2748 --      {$S GVNonRes}
10 2749 -- A    PROCEDURE TTextListView.SetItemWidth (aWidth: INTEGER);
10 2750 0- A    BEGIN
10 2751 --          SetColWidth (1, 1, aWidth);
10 2752 -0 A    END; {* TTextListView.SetItemWidth *}
10 2753 --
10 2754 --
10 2755 --      {*************************************************************************}
10 2756 --      {*  TTextListView.Fields                                                 *}
10 2757 --      {*      Debugging support for the inspector.                             *}
10 2758 --      {*************************************************************************}
10 2759 --      {$IFC qDebug}
10 2760 --      {$S GVFields}
10 2761 -- A    PROCEDURE TTextListView.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 2762 --                  fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 2763 0- A    BEGIN
10 2764 --          DoToField ('TTextListView', NIL, bClass);
10 2765 --          INHERITED Fields (DoToField);
10 2766 -0 A    END; {* TTextListView.Fields *}
10 2767 --      {$ENDC qDebug}
10 2768 --
10 2769 --
10 2770 --      {*************************************************************************}
10 2771 --      {*  TGridSelectCommand                                                   *}
```

```
10 2772 --      {*************************************************************************}
10 2773 --      {*************************************************************************}
10 2774 --      {*  TGridSelectCommand.IGridSelectCommand                                *}
10 2775 --      {*      Skeleton object used to build needed command objects.            *}
10 2776 --      {*************************************************************************}
10 2777 --      {$S GVOpen}
10 2778 -- A    PROCEDURE TGridSelectCommand.IGridSelectCommand (itsCmdNumber: CmdNumber;
10 2779 --                          itsView: TGridView; theShiftKey, theCmdKey: BOOLEAN);
10 2780 --      CONST
10 2781 --          kUseImmediateSuperView = TRUE;
10 2782 --      VAR
10 2783 --          tmpRgn: RgnHandle;
10 2784 0- A    BEGIN
10 2785 --          ICommand (itsCmdNumber, NIL, itsView, itsView.GetScroller (kUseImmediateSuperView));
10 2786 --          fCanUndo := FALSE;
10 2787 --          fCausesChange := FALSE;
10 2788 --
10 2789 --          fShiftKey := theShiftKey;
10 2790 --          fCmdKey := theCmdKey;
10 2791 --          fGridView := itsView;
10 2792 --          fConstrainsMouse := FALSE;
10 2793 --
10 2794 --          tmpRgn  := MakeNewRgn;
10 2795 --          fOldRgn     := tmpRgn;   {* temporary region for old selection *}
10 2796 --          tmpRgn  := MakeNewRgn;
10 2797 --          fNewRgn     := tmpRgn;   {* temporary region for new selection *}
10 2798 -0 A    END; {* TGridSelectCommand.IGridSelectCommand *}
10 2799 --
10 2800 --
10 2801 --      {*************************************************************************}
10 2802 --      {*  TGridSelectCommand.Free                                              *}
10 2803 --      {*      Free the temporary regions.                                      *}
10 2804 --      {*************************************************************************}
10 2805 --      {$S GVClose}
10 2806 -- A    PROCEDURE TGridSelectCommand.Free;
10 2807 0- A    BEGIN
10 2808 --          DisposIfHandle (fOldRgn);
10 2809 --          DisposIfHandle (fNewRgn);
10 2810 --
10 2811 --          INHERITED Free;
10 2812 -0 A    END; {* TGridSelectCommand.Free *}
10 2813 --
10 2814 --
10 2815 --      {*************************************************************************}
10 2816 --      {*  TGridSelectCommand.TrackFeedback                                     *}
10 2817 --      {*      Track the mouse movements when the mouse button is down.         *}
10 2818 --      {*************************************************************************}
10 2819 --      {$S GVRes}
10 2820 -- A    PROCEDURE TGridSelectCommand.TrackFeedback (anchorPoint, nextPoint: VPoint;
10 2821 --                          turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
10 2822 0- A    BEGIN
10 2823 -0 A        END; {* TGridSelectCommand.TrackFeedback *}
10 2824 --
10 2825 --
10 2826 --      {*************************************************************************}
10 2827 --      {*  TGridSelectCommand.Fields                                            *}
10 2828 --      {*      For Debugging support of the inspector.                          *}
10 2829 --      {*************************************************************************}
10 2830 --      {$IFC qDebug}
10 2831 --      {$S GVFields}
10 2832 -- A    PROCEDURE TGridSelectCommand.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 2833 --                  fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 2834 0- A    BEGIN
10 2835 --          DoToField ('TGridSelectCommand', NIL, bClass);
10 2836 --          DoToField ('fGridView', @fGridView, bObject);
10 2837 --          DoToField ('fShiftKey', @fShiftKey, bBoolean);
10 2838 --          DoToField ('fCmdKey', @fCmdKey, bBoolean);
10 2839 --
10 2840 --          INHERITED Fields(DoToField);
10 2841 -0 A    END; {* TGridSelectCommand.Fields *}
10 2842 --      {$ENDC qDebug}
10 2843 --
10 2844 --
10 2845 --      {*************************************************************************}
10 2846 --      {*  TCellSelectCommand                                                   *}
10 2847 --      {*************************************************************************}
10 2848 --      {*************************************************************************}
10 2849 --      {*  TCellSelectCommand.ICellSelectCommand                                *}
10 2850 --      {*      Initialize the cell select command object.                       *}
10 2851 --      {*************************************************************************}
10 2852 --      {$S GVOpen}
10 2853 -- A    PROCEDURE TCellSelectCommand.ICellSelectCommand (itsView: TGridView;
10 2854 --                      aCell: GridCell; theShiftKey, theCmdKey: BOOLEAN);
10 2855 0- A    BEGIN
10 2856 --          IGridSelectCommand (cCellSelect, itsView, theShiftKey, theCmdKey);
10 2857 --          fNewMouseDown := TRUE;
10 2858 -0 A    END; {* TCellSelectCommand.ICellSelectCommand *}
10 2859 --
10 2860 --
10 2861 --      {*************************************************************************}
10 2862 --      {*  TCellSelectCommand.HandleShiftKeyDown                                *}
10 2863 --      {*      This routine provides mouse down support for TCellSelectCommand  *}
10 2864 --      {*      TrackFeedback and TrackMouse.                                    *}
10 2865 --      {*      It implements the functionality of the List Manager's cell selection *}
10 2866 --      {*      algorithm.  (See _Inside_Macintosh_IV_ for more details.)        *}
10 2867 --      {*************************************************************************}
10 2868 --      {$S GVRes}
10 2869 -- A    PROCEDURE TCellSelectCommand.HandleShiftKeyDown (aView: TGridView;
10 2870 --                                  aCell, anchorCell: GridCell; mouseDidMove: BOOLEAN);
```

```
10 2871 --      VAR
10 2872 --          boundRect:  Rect;   {* Bounding Rect of the region *}
10 2873 --          newRect:    Rect;   {* new rectangle to highlight  *}
10 2874 --          newQDRect:  Rect;
10 2875 --          oldQDRect:  Rect;
10 2876 --          newVRect:   VRect;
10 2877 --          oldVRect:   VRect;
10 2878 --          newBRRect:  VRect;
10 2879 --          oldBRRect:  VRect;
10 2880 --
10 2881 0- A    BEGIN
10 2882 --          WITH aView DO
10 2883 1-          BEGIN
10 2884 --              boundRect := fHLRegion^^.rgnBBox;
10 2885 --              boundRect.bottom := boundRect.bottom - 1;
10 2886 --              boundRect.right  := boundRect.right - 1;
10 2887 --
10 2888 --              IF fNewMouseDown THEN
10 2889 2-              BEGIN {* shift click *}
10 2890 --                  newRect.top    := MIN (aCell.v, boundRect.top);
10 2891 --                  newRect.left   := MIN (aCell.h, boundRect.left);
10 2892 --                  newRect.bottom := MAX (aCell.v, boundRect.top) + 1;
10 2893 --                  newRect.right  := MAX (aCell.h, boundRect.left) + 1;
10 2894 -2              END
10 2895 --              ELSE
10 2896 2-              BEGIN {* dragging *}
10 2897 --                  newRect.top    := MIN (aCell.v, anchorCell.v);
10 2898 --                  newRect.left   := MIN (aCell.h, anchorCell.h);
10 2899 --                  newRect.bottom := MAX (aCell.v, anchorCell.v) + 1;
10 2900 --                  newRect.right  := MAX (aCell.h, anchorCell.h) + 1;
10 2901 -2              END;
10 2902 --
10 2903 --              {* Get the region to turn on *}
10 2904 --              RectRgn (fTmpRgn, newRect);
10 2905 --              CreateHighlightRgn (fTmpRgn, fNewRgn, kWholeRect);
10 2906 --
10 2907 --              {* Get the region to turn off *}
10 2908 --              CreateHighlightRgn (fHLRegion, fOldRgn, fWholeRect);
10 2909 --
10 2910 --              {* turn off the Old bits *}
10 2911 --              CopyRgn (fOldRgn, fTmpRgn);
10 2912 --              DiffRgn (fOldRgn, fNewRgn, fOldRgn);
10 2913 --              DoHilite (fOldRgn, hlON, hlOFF);
10 2914 --              DiffRgn (fNewRgn, fTmpRgn, fNewRgn);
10 2915 --
10 2916 --              {* turn on the new bits *}
10 2917 --              DoHilite (fNewRgn, hlOFF, hlON);
10 2918 --
10 2919 --              {* set fHLRegion to be correct *}
10 2920 --              RectRgn (fHLRegion, newRect);
10 2921 -1          END;
10 2922 -0 A    END; {* TCellSelectCommand.HandleShiftKeyDown *}
10 2923 --
10 2924 --
10 2925 --      {*****************************************************************************}
10 2926 --      {*  TCellSelectCommand.TrackFeedback                                         *}
10 2927 --      {*      Track the mouse movements when the mouse button is down.             *}
10 2928 --      {*****************************************************************************}
10 2929 --      {$S GVRes}
10 2930 -- A    PROCEDURE TCellSelectCommand.TrackFeedback (anchorPoint, nextPoint: VPoint;
10 2931 --                          turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
10 2932 --      VAR
10 2933 --          aCell:          GridCell;
10 2934 --          anchorCell:     GridCell;
10 2935 --          aRect:          VRect;
10 2936 --          hlRect:         Rect;
10 2937 --
10 2938 0- A    BEGIN
10 2939 --          WITH fGridView DO
10 2940 1-          BEGIN
10 2941 --              aCell := VPointToCell (nextPoint);
10 2942 --
10 2943 --              IF  NOT (SameCell (aCell, fOldCell)) &
10 2944 --                  (aCell.v > 0) & (aCell.h > 0) &
10 2945 --                  (aCell.v <= fNumOfRows) & (aCell.h <= fNumOfCols) &
10 2946 --                  mouseDidMove & CanSelectCell (aCell)  THEN
10 2947 2-              BEGIN
10 2948 --                  {* get the new selection *}
10 2949 --                  CellToVRect (aCell, aRect);
10 2950 --                  ViewToQDRect (aRect, hlRect);
10 2951 --                  RectRgn (fNewRgn, hlRect);
10 2952 --
10 2953 --                  {* set the highlighted region *}
10 2954 --                  hlRect.topleft := aCell;
10 2955 --                  hlRect.bottom  := aCell.v + 1;
10 2956 --                  hlRect.right   := aCell.h + 1;
10 2957 --
10 2958 --                  IF  fCmdKey & NOT (fSingleSelection)  THEN
10 2959 3-                  BEGIN
10 2960 --                      IF fNewMouseDown THEN
10 2961 4-                      BEGIN
10 2962 --                          {* turn cells on or off? *}
10 2963 --                          IF  PtInRgn (aCell, fHLRegion) THEN
10 2964 --                              fKeepSelecting := FALSE
10 2965 --                          ELSE
10 2966 --                              fKeepSelecting := TRUE;
10 2967 --                          fNewMouseDown := FALSE;
10 2968 -4                      END;
10 2969 --
```

```
10 2970 --                      {* set up the selection region containing the cell *}
10 2971 --                      RectRgn (fTmpRgn, hlRect);
10 2972 --
10 2973 --                      IF fKeepSelecting THEN
10 2974 4-                      BEGIN
10 2975 --                          IF NOT PtInRgn (aCell, fHLRegion) THEN
10 2976 5-                          BEGIN
10 2977 --                              UnionRgn (fHLRegion, fTmpRgn, fHLRegion);
10 2978 --                              DoHilite (fNewRgn, hlOFF, hlON);
10 2979 -5                          END;
10 2980 -4                      END
10 2981 --                      ELSE IF  PtInRgn (aCell, fHLRegion) THEN
10 2982 4-                      BEGIN
10 2983 --                          DiffRgn (fHLRegion, fTmpRgn, fHLRegion);
10 2984 --                          DoHilite (fNewRgn, hlON, hlOFF);
10 2985 -4                      END;
10 2986 --
10 2987 --                      fWholeRect := FALSE;  {* the selected region may not be a rectangle *}
10 2988 -3                  END
10 2989 --
10 2990 --                  ELSE IF fShiftKey  & NOT (fSingleSelection) THEN
10 2991 3-                  BEGIN
10 2992 --                      anchorCell := VPointToCell (anchorPoint);
10 2993 --                      HandleShiftKeyDown (fGridView, aCell, anchorCell, mouseDidMove);
10 2994 --                      fWholeRect := TRUE;  {* the selected region is a complete rectangle *}
10 2995 -3                  END
10 2996 --
10 2997 --                  ELSE  {* No recognized keys down with the mouse *}
10 2998 3-                  BEGIN
10 2999 --                      {* get the old selection *}
10 3000 --                      CreateHighlightRgn (fHLRegion, fOldRgn, fWholeRect);
10 3001 --
10 3002 --                      {* Turn off the old ones *}
10 3003 --                      DiffRgn (fOldRgn, fNewRgn, fOldRgn);
10 3004 --                      DoHilite (fOldRgn, hlON, hlOFF);
10 3005 --
10 3006 --                      {* highlight the new selection *}
10 3007 --                      IF NOT PtInRgn (aCell, fHLRegion) THEN
10 3008 --                          DoHilite (fNewRgn, hlOFF, hlON);
10 3009 --
10 3010 --                      RectRgn (fHLRegion, hlRect);
10 3011 --
10 3012 --                      fWholeRect := TRUE;  {* the selected region is a rectangle *}
10 3013 --                      fShiftKey := TRUE;   {* dragging keeps selecting the rectangle *}
10 3014 -3                  END;
10 3015 -2              END;
10 3016 -1          END;
10 3017 --          {* keep track of the old cell selection *}
10 3018 --          fOldCell := aCell;
10 3019 --          fNewMouseDown := FALSE;
10 3020 -0 A    END; {* TCellSelectCommand.TrackFeedback *}
10 3021 --
10 3022 --
10 3023 --      {*****************************************************************************}
10 3024 --      {*  TCellSelectCommand.TrackMouse                                            *}
10 3025 --      {*      Track the mouse when the mouse button comes up.                      *}
10 3026 --      {*****************************************************************************}
10 3027 --      {$S GVRes}
10 3028 -- A    FUNCTION TCellSelectCommand.TrackMouse (aTrackPhase: TrackPhase;
10 3029 --                      VAR anchorPoint, previousPoint, nextPoint: VPoint;
10 3030 --                      mouseDidMove: BOOLEAN): TCommand; OVERRIDE;
10 3031 0- A    BEGIN
10 3032 --          WITH fGridView DO
10 3033 1-          BEGIN
10 3034 --              IF (aTrackPhase = TrackRelease) THEN
10 3035 2-              BEGIN
10 3036 --                  {* set the new selection *}
10 3037 --                  IF CanSelectCell (fOldCell) THEN
10 3038 3-                  BEGIN
10 3039 --                      IF fSingleSelection THEN
10 3040 --                          SelectCell (fOldCell, kDontExtend, kDontHighlight, kSelect)
10 3041 --                      ELSE
10 3042 --                          SetSelection (fHLRegion, kDontExtend, kDontHighlight, kSelect);
10 3043 -3                  END;
10 3044 --                  fNewMouseDown := TRUE;
10 3045 -2              END;
10 3046 -1          END; {* with *}
10 3047 --
10 3048 --          TrackMouse := SELF;
10 3049 -0 A    END; {* TCellSelectCommand.TrackMouse *}
10 3050 --
10 3051 --
10 3052 --      {*****************************************************************************}
10 3053 --      {*  TCellSelectCommand.Fields                                                *}
10 3054 --      {*      For Debugging support of the inspector.                              *}
10 3055 --      {*****************************************************************************}
10 3056 --      {$IFC qDebug}
10 3057 --      {$S GVFields}
10 3058 -- A    PROCEDURE TCellSelectCommand.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 3059 --                      fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 3060 0- A    BEGIN
10 3061 --          DoToField ('TCellSelectCommand', NIL, bClass);
10 3062 --          DoToField ('fOldCell', @fOldCell, bInteger);
10 3063 --          DoToField ('fNewMouseDown', @fNewMouseDown, bBoolean);
10 3064 --          DoToField ('fKeepSelecting', @fKeepSelecting, bBoolean);
10 3065 --
10 3066 --          INHERITED Fields(DoToField);
10 3067 -0 A    END; {* TCellSelectCommand.Fields *}
10 3068 --      {$ENDC qDebug}
```

```
10 3069 --
10 3070 --
10 3071 --      {*****************************************************************************}
10 3072 --      {*  TRowSelectCommand                                                        *}
10 3073 --      {*****************************************************************************}
10 3074 --      {*****************************************************************************}
10 3075 --      {*  TRowSelectCommand.IRowSelectCommand                                      *}
10 3076 --      {*      Initialize the row select command object.                            *}
10 3077 --      {*****************************************************************************}
10 3078 --      {$S GVOpen}
10 3079 -- A    PROCEDURE TRowSelectCommand.IRowSelectCommand (itsView: TGridView;
10 3080 --                          aRow: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
10 3081 0- A    BEGIN
10 3082 --          IGridSelectCommand (cRowSelect, itsView, theShiftKey, theCmdKey);
10 3083 --          fTheRow := aRow;
10 3084 -0 A    END; {* TRowSelectCommand.IRowSelectCommand *}
10 3085 --
10 3086 --
10 3087 --      {*****************************************************************************}
10 3088 --      {*  TRowSelectCommand.Fields                                                 *}
10 3089 --      {*      For Debugging support of the inspector.                              *}
10 3090 --      {*****************************************************************************}
10 3091 --      {$IFC qDebug}
10 3092 --      {$S GVFields}
10 3093 -- A    PROCEDURE TRowSelectCommand.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 3094 --                  fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 3095 0- A    BEGIN
10 3096 --          DoToField ('TRowSelectCommand', NIL, bClass);
10 3097 --          DoToField ('fTheRow', @fTheRow, bInteger);
10 3098 --
10 3099 --          INHERITED Fields(DoToField);
10 3100 -0 A    END; {* TRowSelectCommand.Fields *}
10 3101 --      {$ENDC qDebug}
10 3102 --
10 3103 --
10 3104 --      {*****************************************************************************}
10 3105 --      {*  TColSelectCommand                                                        *}
10 3106 --      {*****************************************************************************}
10 3107 --      {*****************************************************************************}
10 3108 --      {*  TColSelectCommand.IColumnSelectCommand                                   *}
10 3109 --      {*      Initialize the column select command object.                         *}
10 3110 --      {*****************************************************************************}
10 3111 --      {$S GVOpen}
10 3112 -- A    PROCEDURE TColSelectCommand.IColSelectCommand (itsView: TGridView;
10 3113 --                          aColumn: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
10 3114 0- A    BEGIN
10 3115 --          IGridSelectCommand (cColumnSelect, itsView, theShiftKey, theCmdKey);
10 3116 --          fTheColumn := aColumn;
10 3117 -0 A    END; {* TColSelectCommand.IColSelectCommand *}
10 3118 --
10 3119 --
10 3120 --      {*****************************************************************************}
10 3121 --      {*  TColSelectCommand.Fields                                                 *}
10 3122 --      {*      For Debugging support of the inspector.                              *}
10 3123 --      {*****************************************************************************}
10 3124 --      {$IFC qDebug}
10 3125 --      {$S GVFields}
10 3126 -- A    PROCEDURE TColSelectCommand.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 3127 --                  fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 3128 0- A    BEGIN
10 3129 --          DoToField ('TColSelectCommand', NIL, bClass);
10 3130 --          DoToField ('fTheColumn', @fTheColumn, bInteger);
10 3131 --
10 3132 --          INHERITED Fields(DoToField);
10 3133 -0 A    END; {* TColSelectCommand.Fields *}
10 3134 --      {$ENDC qDebug}
10 3135 --
10 3136 --
10 3137 --      {*****************************************************************************}
10 3138 --      {*  TVertexSelectCommand                                                     *}
10 3139 --      {*****************************************************************************}
10 3140 --      {*****************************************************************************}
10 3141 --      {*  TVertexSelectCommand.IVertexSelectCommand                                *}
10 3142 --      {*      Initialize the vertex select command object.                         *}
10 3143 --      {*      The "vertex" is the place where rows and columns meet.               *}
10 3144 --      {*****************************************************************************}
10 3145 --      {$S GVOpen}
10 3146 -- A    PROCEDURE TVertexSelectCommand.IVertexSelectCommand (itsView: TGridView;
10 3147 --                          aRow, aColumn: INTEGER; theShiftKey, theCmdKey: BOOLEAN);
10 3148 0- A    BEGIN
10 3149 --          IGridSelectCommand (cVertexSelect, itsView, theShiftKey, theCmdKey);
10 3150 --          fTheRow := aRow;
10 3151 --          fTheColumn := aColumn;
10 3152 -0 A    END; {* TVertexSelectCommand.IVertexSelectCommand *}
10 3153 --
10 3154 --
10 3155 --      {*****************************************************************************}
10 3156 --      {*  TVertexSelectCommand.Fields                                              *}
10 3157 --      {*      For Debugging support of the inspector.                              *}
10 3158 --      {*****************************************************************************}
10 3159 --      {$IFC qDebug}
10 3160 --      {$S GVFields}
10 3161 -- A    PROCEDURE TVertexSelectCommand.Fields ( PROCEDURE DoToField (fieldName: Str255;
10 3162 --                  fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
10 3163 0- A    BEGIN
10 3164 --          DoToField ('TVertexSelectCommand', NIL, bClass);
10 3165 --          DoToField ('fTheRow', @fTheRow, bInteger);
10 3166 --          DoToField ('fTheColumn', @fTheColumn, bInteger);
10 3167 --
```

```
10 3168 --          INHERITED Fields (DoToField);
10 3169 -0 A    END; {* TVertexSelectCommand.Fields *}
10 3170 --      {$ENDC qDebug}
10 3171 --
10 3172 --
10 3173 --      {*****************************************************************************}
10 3174 --      {*  End Of File: UGridView.incl.p                                            *}
10 3175 --      {*****************************************************************************}
 9  782 --
 9  783 --      END.
 9  784 --
 9  785 --      {*****************************************************************************}
 9  786 --      {*  End of File: UGridView.p                                                 *}
 9  787 --      {*****************************************************************************}
```

```
11    1 --    {$P}
11    2 --    { UList.p }
11    3 --    { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
11    4 --
11    5 --    UNIT UList;
11    6 --
11    7 --    { Unit UList defines object type TList, a simple dynamic array whose elements
11    8 --       are references to TObjects }
11    9 --
11   10 --    INTERFACE
11   11 --
11   12 --    {-----------------------------------+
11   13 --    |   Uses                            |
11   14 --    +-----------------------------------}
11   15 --    USES
11   16 --        {$LOAD MacIntf.LOAD}
11   17 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
11   18 --        {$LOAD}
11   19 --            UMAUtil,
11   20 --            UFailure,
11   21 --            UMemory,
11   22 --            UObject;
11   23 --
11   24 --
11   25 --    {-----------------------------------+
11   26 --    |   Global constants                |
11   27 --    +-----------------------------------}
11   28 --    CONST
11   29 --        kDeletedElement =   $FFFFFFFF;
11   30 --
11   31 --        kALessThanB =       -1;             { Constants for TSortedList.Compare and Search. }
11   32 --        kAEqualB =          0;
11   33 --        kAGreaterThanB =    1;
11   34 --
11   35 --
11   36 --    {-----------------------------------+
11   37 --    |   Global types                    |
11   38 --    +-----------------------------------}
11   39 --    TYPE
11   40 --            {RW means OK for other objects to Read and Write this field}
11   41 --            {Rx means OK for other objects to Read, but NOT TO WRITE this field}
11   42 --            {xx means other objects should NEITHER READ NOR WRITE this field}
11   43 --
11   44 --            {A dynamic list of TObjects}
11   45 --
11   46 --        {-----------------------------------+
11   47 --        |   TList                           |
11   48 --        +-----------------------------------}
11   49 --        TList = OBJECT(TObject)         {It IS permissible for descendants to add
11   50 --                                            more named fields}
11   51 --            fSize:          INTEGER;    {Rx}{number of elements in the list, from 0
11   52 --                                            to the limit of memory}
11   53 --            fFirstOffset:   LONGINT;    {xx}{= SIZEOF(SELF) - no. of bytes of named
11   54 --                                            fields before first element}
11   55 --            fDeletions:     INTEGER;    {Rx}{The number of deleted elements in the
11   56 --                                            list. These have value
11   57 --                                            kDeletedElement;  fSize always
11   58 --                                            reflects # of real elements.}
11   59 --            fEachLevel:     INTEGER;    {Rx}{Number of Each calls in progress.}
11   60 --    {$IFC qDebug}
11   61 --            fEltClass:      String8;    {xx}{if <> '', then the name of the class
11   62 --                                            of the elements of the list}
11   63 --            fObjPtr:        LONGINT;    {xx}{internal}
11   64 --    {$ENDC}
11   65 --
11   66 --          {Creation/Destruction Methods}
11   67 --
11   68 --            PROCEDURE TList.IList;
11   69 --                {Initialize a new list with no elements, i.e., fSize = 0.  Always
11   70 --                    call it once before calling any other method.  Never call it
11   71 --                    twice.}
11   72 --
11   73 --            PROCEDURE TList.FreeList;
11   74 --                {Frees each object in the list, then frees the list.}
11   75 --
11   76 --          {Utility Methods}
11   77 --
11   78 --            FUNCTION  TList.GetSize: INTEGER;
11   79 --                {Returns the number of elements in the list.}
11   80 --
11   81 --            FUNCTION  TList.At(index: INTEGER): TObject;
11   82 --                {Return the index'th element of the list.  It is typical for the
11   83 --                    caller to coerce the result into a descendant of TObject.
11   84 --                    All lists are indexed from 1.  Range check only if the
11   85 --                    compile-flag qRangeCheck is TRUE.
11   86 --                The static-array equivalent of:
11   87 --                    object := objList.At(index);
11   88 --                is:
11   89 --                    object := objArray[index];}
11   90 --
11   91 --            FUNCTION TList.GetItemNumber(item: TObject): INTEGER;
11   92 --                {Return the index of the first occurance of item in the list,
11   93 --                    or fSize + 1 if item is not in the list.}
11   94 --
11   95 --            FUNCTION  TList.First: TObject;
11   96 --                {Return the first element of the list.  It is typical for the
11   97 --                    caller to coerce the result.  Returns NIL if the size is <= 0.
11   98 --
```

```
11   99 --                The static-array equivalent of:
11  100 --                   object := objList.First;
11  101 --                is:
11  102 --                   object := objArray[1];}
11  103 --
11  104 --             FUNCTION  TList.Last: TObject;
11  105 --                 {Return the last element of the list.  It is typical for the
11  106 --                    caller to coerce the result.  Returns NIL if the size is <= 0.
11  107 --
11  108 --                The static-array equivalent of:
11  109 --                   object := objList.Last;
11  110 --                is:
11  111 --                   object := objArray[number of elements];}
11  112 --
11  113 --          {Iterator Methods}
11  114 --
11  115 --             PROCEDURE TList.Each(PROCEDURE DoToItem(item: TObject));
11  116 --                 {Call DoToItem once for each element of the list, in order.  The
11  117 --                    actual parameter is typically a procedure whose argument is a
11  118 --                    descendant of TObject.  If DoToItem calls InsertLast, the newly
11  119 --                    added element will NOT be enumerated.  If TestItem calls AtPut,
11  120 --                    InsertBefore, InsertFirst, Delete, or DeleteAll,  misbehavior
11  121 --                    will ensue.  If DoToItem calls Delete, the item is replaced by
11  122 --                    the constant kDeletedElement; fSize is decremented and
11  123 --                    fDeletions is incremented.  At the end of all calls to Each,
11  124 --                    TList.RemoveDeletions will be called.
11  125 --
11  126 --                The static-array equivalent of:
11  127 --                   objList.Each(Proc)
11  128 --                is:
11  129 --                   FOR index := 1 TO fSize DO
11  130 --                       Proc(objArray[index])}
11  131 --
11  132 --             FUNCTION TList.FirstThat(FUNCTION TestItem(item: TObject): BOOLEAN):
11  133 --                    TObject;
11  134 --                 {Call TestItem once for each element of the list, in order, until
11  135 --                    TestItem returns TRUE.  Return the element that satisfied the
11  136 --                    test.  If none satisfied the test, return NIL.  The actual
11  137 --                    parameter is typically a procedure whose argument is a
11  138 --                    descendant of TObject.  It is typical for the caller to coerce
11  139 --                    the result into that descendant of TObject.  If TestItem calls
11  140 --                    InsertLast, the newly added element will NOT be enumerated.
11  141 --                    If TestItem calls AtPut, InsertBefore, InsertFirst, Delete, or
11  142 --                    DeleteAll, misbehavior will ensue.
11  143 --
11  144 --                The static-array equivalent of:
11  145 --                   object := objList.FirstThat(Func)
11  146 --                is:
11  147 --                   object := NIL;
11  148 --                   FOR index := 1 TO fSize DO IF Func(objArray[index]) THEN
11  149 --                       BEGIN
11  150 --                       object := objArray[index];
11  151 --                       LEAVE;
11  152 --                       END}
11  153 --
11  154 --             FUNCTION TList.LastThat(FUNCTION TestItem(item: TObject): BOOLEAN):
11  155 --                    TObject;
11  156 --                 {Call TestItem once for each element of the list, starting with
11  157 --                    the last item in the list and working toward the first item,
11  158 --                    until TestItem returns TRUE.  Return the element that satisfied
11  159 --                    the test.  If none satisfied the test, return NIL.  The actual
11  160 --                    parameter is typically a procedure whose argument is a
11  161 --                    descendant of TObject.  It is typical for the caller to coerce
11  162 --                    the result into that descendant of TObject.  If TestItem calls
11  163 --                    InsertLast, the newly added element will NOT be enumerated.
11  164 --                    If TestItem calls AtPut, InsertBefore, InsertFirst, Delete, or
11  165 --                    DeleteAll, misbehavior will ensue.
11  166 --
11  167 --                The static-array equivalent of:
11  168 --                   object := objList.LastThat(Func)
11  169 --                is:
11  170 --                   object := NIL;
11  171 --                   FOR index := fSize DOWNTO 1 DO IF Func(objArray[index]) THEN
11  172 --                       BEGIN
11  173 --                       object := objArray[index];
11  174 --                       LEAVE;
11  175 --                       END}
11  176 --
11  177 --          {Item Insertion Methods}
11  178 --
11  179 --             PROCEDURE TList.AtPut(index: INTEGER; newItem: TObject);
11  180 --                 {Replace the index'th item of the list.  Does not free the old
11  181 --                    item.  If you do this from within an Each the results will be
11  182 --                    unpredictable.}
11  183 --
11  184 --             PROCEDURE TList.InsertBefore(index: INTEGER; item: TObject);
11  185 --                 {Insert a reference to an item before the indicated item.  The
11  186 --                    index of the new element will be index.
11  187 --
11  188 --                 If index = 1 this inserts at the start of the list.
11  189 --                 If index = fSize+1 this inserts at the end of the list.
11  190 --
11  191 --                 Signals Failure if unable to increase the size of the list.}
11  192 --
11  193 --
11  194 --             PROCEDURE TList.InsertFirst(item: TObject);
11  195 --                 {Insert a reference to item at the front of the list.  If the
11  196 --                    compile-flag qDebug is TRUE & SetEltType was called, verify
11  197 --                    item's type.  Increase fSize by 1.  The index of the new
```

```
11  198 --                  element will be 1.  Signals Failure if unable to increase the
11  199 --                  size of the list.}
11  200 --
11  201 --
11  202 --          PROCEDURE TList.InsertLast(item: TObject);
11  203 --              {Insert a reference to item at the back of the list.  If the
11  204 --                  compile-flag qDebug is TRUE & SetEltType was called, verify
11  205 --                  item's type.  Increase fSize by 1.  The index of the new
11  206 --                  element will be fSize.
11  207 --
11  208 --              Signals Failure if unable to increase the size of the list.}
11  209 --
11  210 --          {Item Deletion Methods}
11  211 --
11  212 --          PROCEDURE TList.Delete(item: TObject);
11  213 --              {Delete the first reference to item from the list, but do not free
11  214 --                  item.  If item does not occur, do nothing.  If item does occur,
11  215 --                  reduce fSize by 1.}
11  216 --
11  217 --          PROCEDURE TList.DeleteAll;
11  218 --              {Delete every element from the list, but do not free any objects.
11  219 --                  Leave fSize at 0.}
11  220 --
11  221 --          PROCEDURE TList.FreeAll;
11  222 --              {Frees each element in the list.  Leave fSize at 0.}
11  223 --
11  224 --          {Internal Methods}
11  225 --
11  226 --          FUNCTION  TList.ComputeAddress (index: INTEGER): LONGINT;
11  227 --              {Returns the memory address of the index-th element in the list.
11  228 --                  Used internally.}
11  229 --
11  230 --          FUNCTION  TList.MaxAddress: LONGINT;
11  231 --              {Returns the memory address after the last item in the list.
11  232 --                  Used internally.}
11  233 --
11  234 --          PROCEDURE TList.RemoveDeletions;
11  235 --              {Remove the deleted elements from the list.}
11  236 --
11  237 --          {Debugging Methods}
11  238 --
11  239 --      {$IFC qDebug}
11  240 --          PROCEDURE TList.SetEltType(toClass: Str255);
11  241 --              {Call this once and pass the kind of objects you intend to insert
11  242 --                  into the list.  The TList insert methods will do a coercion to
11  243 --                  the type you specify, to ensure that the list contains only
11  244 --                  elements of the right type.}
11  245 --
11  246 --          PROCEDURE TList.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
11  247 --          PROCEDURE TList.Fields (PROCEDURE DoToField (fieldName: Str255;
11  248 --                                                       fieldAddr: Ptr;
11  249 --                                                       fieldType: INTEGER)); OVERRIDE;
11  250 --      {$ENDC}
11  251 --
11  252 --          END {TList};
11  253 --
11  254 --
11  255 --      {-----------------------------------+
11  256 --      |   TSortedList                     |
11  257 --      +-----------------------------------}
11  258 --      TSortedList = OBJECT(TList)
11  259 --
11  260 --          FUNCTION  TSortedList.GetItemNumber(item: TObject): INTEGER; OVERRIDE;
11  261 --              { If item is in the list, GetItemNumber returns an index number to an occurance
11  262 --                  of item.  If the item is in the list more than once, then the index returned
11  263 --                  is not guarenteed to be the first occurance of the item because a binary
11  264 --                  search is used.
11  265 --
11  266 --                  If the item is not in the list, then GetItemNumber returns the index at
11  267 --                  which the item should be inserted into the list.  This GetItemNumber
11  268 --                  does not tell you whether the item is in fact in the list.  This can be
11  269 --                  determined by calling the Search method or by calling the At method on
11  270 --                  the result of GetItemNumber.}
11  271 --
11  272 --          PROCEDURE TSortedList.Insert (item: TObject);
11  273 --              { Inserts the given item into the list in sorted order by using a binary
11  274 --                  search.  The Compare method is used to determine the items location in the
11  275 --                  list.}
11  276 --
11  277 --          FUNCTION  TSortedList.Compare (item1, item2: TObject): INTEGER;
11  278 --              { This method compares to items in the list.  It must be overridden.
11  279 --                  A negative result indicates that item1 < item2.  A result of zero indicates
11  280 --                  that item1 = item2.  A positive result indicates that item1 > item2.
11  281 --                  You may wish to use the constants kALessThanB, kAEqualB, kAGreaterThanB. }
11  282 --
11  283 --          FUNCTION  TSortedList.Search (FUNCTION TestItem(anItem: TObject): INTEGER): TObject;
11  284 --              { This method performs a binary search on the list in which you provide
11  285 --                  the function that tests items in the list.  This is useful for cases
11  286 --                  in which you are not comparing objects in the list with another object,
11  287 --                  as does Compare.  For example, suppose each object has an fTitle field
11  288 --                  and the objects are inserted into the list in order of fTitle.  You
11  289 --                  can use the search method to look for an object whose fTitle is equal
11  290 --                  to any string.
11  291 --
11  292 --                  A negative TestItem result indicates that anItem < your search criteria,
11  293 --                  a result of zero indicates the item has been found and Search returns that
11  294 --                  item.  A positive TestItem result indicates that anItem > your search
11  295 --                  criteria. }
11  296 --
```

```
11  297 --              END {TSortedList};
11  298 --
11  299 --
11  300 --          {-----------------------------------+
11  301 --          |   Global Routines                 |
11  302 --          +-----------------------------------}
11  303 --          FUNCTION  NewList: TList;
11  304 --              {A convenience function.  Create a TList, initialize it, and
11  305 --                  return a reference to it.
11  306 --
11  307 --               Signals Failure if it cannot allocate the object.}
11  308 --
11  309 --
11  310 --      IMPLEMENTATION
11  311 --
11  312 --      {$I UList.incl.p}
```

```
12    1 --      {$P}
12    2 --      { UList.incl.p }
12    3 --      { Copyright © 1986-1988 by Apple Computer, Inc.  All rights reserved. }
12    4 --      (*
12    5 --          Need more testing of Deletions & At calls during Each; multiple Each's
12    6 --      *)
12    7 --
12    8 --      {-------------------------------------+
12    9 --      |   Local constants                   |
12   10 --      +-------------------------------------}
12   11 --      CONST
12   12 --          kElemSize = SIZEOF(Handle);         { Each element is really a handle }
12   13 --
12   14 --      {-------------------------------------+
12   15 --      |   Local types                       |
12   16 --      +-------------------------------------}
12   17 --      TYPE
12   18 --          PObj =      ^TObject;
12   19 --
12   20 --      {-------------------------------------+
12   21 --      |   External declarations             |
12   22 --      +-------------------------------------}
12   23 --      {$%+ }
12   24 -- A    FUNCTION  %_InObj(ordObject, jumpTablePtr: LONGINT): BOOLEAN; EXTERNAL;
12   25 --      {$%-}
12   26 --
12   27 --
12   28 --      (*****************************************************************************************)
12   29 --      (*      G l o b a l   R o u t i n e s                                                    *)
12   30 --      (*****************************************************************************************)
12   31 --      {$S ListRes}
12   32 --      {-------------------------------------+
12   33 --      |   NewList                           |
12   34 --      +-------------------------------------}
12   35 -- A    FUNCTION  NewList: TList;
12   36 --              { A convenience function.
12   37 --                Create a TList, initialize it, and return a reference to it. }
12   38 --      VAR
12   39 --          list:   TList;
12   40 0- A    BEGIN
12   41 --          New(list);
12   42 --          FailNil(list);
12   43 --
12   44 --          list.IList;
12   45 --          NewList := list;
12   46 -0 A    END;
12   47 --
12   48 --
12   49 --      (*****************************************************************************************)
12   50 --      (*      T L i s t                                                                        *)
12   51 --      (*****************************************************************************************)
12   52 --      {$S ListRes}
12   53 --      {-------------------------------------+
12   54 --      |   IList                             |
12   55 --      +-------------------------------------}
12   56 -- A    PROCEDURE TList.IList;
12   57 --          { Initialize a new list with no elements, i.e., fSize = 0 }
12   58 0- A    BEGIN
12   59 --          fSize := 0;
12   60 --          { We store the items in the list after the last declared field of the object.
12   61 --              fFirstOffset is the offset into the object where the first item is stored. }
12   62 --          fFirstOffset := GetHandleSize(Handle(SELF));
12   63 --          fEachLevel := 0;
12   64 --          fDeletions := 0;
12   65 --      {$IFC qDebug}
12   66 --          fEltClass := '';
12   67 --          fObjPtr := 0;
12   68 --      {$ENDC}
12   69 -0 A    END;
12   70 --
12   71 --
12   72 --      {$S ListRes}
12   73 --      {-------------------------------------+
12   74 --      |   At                                |
12   75 --      +-------------------------------------}
12   76 -- A    FUNCTION  TList.At(index: INTEGER): TObject;
12   77 --          { Return the index'th element of the list.
12   78 --               Range check only if the compile-flag qRangeCheck is TRUE. }
12   79 --      VAR
12   80 --          p:          LONGINT;
12   81 --          item:       LONGINT;
12   82 --          x:          INTEGER;
12   83 0- A    BEGIN
12   84 --          {$IFC qRangeCheck}
12   85 --          IF (index < 1) OR (index > fSize) THEN
12   86 1-              BEGIN
12   87 --      {$IFC qDebug}
12   88 --              Writeln('fSize = ', fSize:1, '  index = ', index:1);
12   89 --              ProgramBreak('Range Check in TList.At');
12   90 --      {$ENDC}     .
12   91 -1              END;
12   92 --          {$ENDC}
12   93 --
12   94 --          At := PObj(ComputeAddress(index))^;
12   95 -0 A    END;
12   96 --
12   97 --
12   98 --      {$S ListRes}
```

```
12   99 --     {-----------------------------------+
12  100 --     |   AtPut                           |
12  101 --     +-----------------------------------}
12  102 -- A   PROCEDURE TList.AtPut(index: INTEGER; newItem: TObject);
12  103 --         { Replace the index'th element of the list.
12  104 --             Range check only if the compile-flag qRangeCheck is TRUE. }
12  105 --     VAR
12  106 --         p:          PObj;
12  107 --         x:          INTEGER;
12  108 --         item:       LONGINT;
12  109 0- A   BEGIN
12  110 --         {$IFC qRangeCheck}
12  111 --         IF (index < 1) OR (index > fSize) THEN
12  112 1-             BEGIN
12  113 --     {$IFC qDebug}
12  114 --             Writeln('fSize = ', fSize:1, '  index = ', index:1);
12  115 --             ProgramBreak('Range Check in TList.AtPut');
12  116 --     {$ENDC}
12  117 -1             END;
12  118 --         {$ENDC}
12  119 --
12  120 --     {$%+}
12  121 --     {$IFC qDebug}
12  122 --         IF fObjPtr > 0 THEN
12  123 --             IF NOT %_InObj(ORD(newItem), fObjPtr) THEN
12  124 1-                 BEGIN
12  125 --                 WrLblPtr('newItem', newItem);
12  126 --                 Writeln;
12  127 --                 ProgramBreak('In TList.AtPut: inserting invalid object type');
12  128 -1                 END;
12  129 --     {$ENDC}
12  130 --     {$%-}
12  131 --
12  132 --         p := PObj(ComputeAddress(index));
12  133 --         p^ := newItem;
12  134 -0 A   END;
12  135 --
12  136 --
12  137 --     { ELEMENT-ADDRESS-COMPUTATION used in many TList methods:
12  138 --         Ord(Handle(SELF)^)  = address of the class pointer within the heap block
12  139 --         fFirstOffset        = no. of bytes of fixed fields = offset from class pointer to first element
12  140 --         (kElemSize * index) - kElemSize =
12  141 --             kElemSize * (index - 1) = no. of bytes in elements preceding the index'th element
12  142 --      NOTES:
12  143 --         Direct heap pointers are not preserved across procedure calls that might compact the heap.
12  144 --         The ROM routine Munger can do fast inserts and deletes. }
12  145 --
12  146 --
12  147 --     {$S ListRes}
12  148 --     {-----------------------------------+
12  149 --     |   ComputeAddress                  |
12  150 --     +-----------------------------------}
12  151 -- A   FUNCTION  TList.ComputeAddress (index: INTEGER): LONGINT;
12  152 --     VAR
12  153 --         p:      LONGINT;
12  154 --         x:      INTEGER;
12  155 --         item:   LONGINT;
12  156 0- A   BEGIN
12  157 --         p := Ord(Handle(SELF)^) + fFirstOffset;     { Address of first physical item           }
12  158 --
12  159 --         IF fDeletions = 0 THEN                      { If no deletions, index to proper element  }
12  160 --             ComputeAddress := p + BSL(index - 1, 2) { p + ((index - 1) * 4)                     }
12  161 --         ELSE                                        { Ignore deleted elements                   }
12  162 1-             BEGIN
12  163 --             x := index;
12  164 --             p := p - kElemSize;                     { we increment first in loop                }
12  165 --
12  166 --             WHILE x > 0 DO                          { Skip elements until item is found         }
12  167 2-                 BEGIN
12  168 --                 p := p + kElemSize;
12  169 --                 item := PLongInt(p)^;
12  170 --                 IF item <> kDeletedElement THEN
12  171 --                     x := x - 1;
12  172 -2                 END;
12  173 --
12  174 --             ComputeAddress := p;
12  175 -1             END
12  176 -0 A   END;
12  177 --
12  178 --
12  179 --     {$S ListRes}
12  180 --     {-----------------------------------+
12  181 --     |   Delete                          |
12  182 --     +-----------------------------------}
12  183 -- A   PROCEDURE TList.Delete(item: TObject);
12  184 --         { Delete the first reference to item in the list, but do not free item.
12  185 --             If item does not occur, do nothing. }
12  186 --     VAR
12  187 --         p:          LONGINT;
12  188 --         maxP:       LONGINT;
12  189 --         offset:     LONGINT;
12  190 0- A   BEGIN
12  191 --         { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  192 --         p := ComputeAddress(1);         { Address of first element }
12  193 --         maxP := MaxAddress;             { Address past last element }
12  194 --
12  195 --         { Skip elements until item is found }
12  196 --         WHILE (p < maxP) & (PObj(p)^ <> item) DO
12  197 --             p := p + kElemSize;
```

```
12  198 --
12  199 --          IF p < maxP THEN { Item was found }
12  200 1-              BEGIN
12  201 --              IF fEachLevel > 0 THEN
12  202 2-                  BEGIN
12  203 --                  fDeletions := fDeletions + 1;
12  204 --                  PLongInt(p)^ := kDeletedElement;
12  205 -2                  END
12  206 --              ELSE
12  207 --                  offset := Munger(Handle(SELF), p - Ord(Handle(SELF)^), NIL,
12  208 --                                  kElemSize, Ptr(@item), 0);
12  209 --
12  210 --              fSize := fSize - 1;
12  211 -1              END;
12  212 -0 A    END;
12  213 --
12  214 --
12  215 --      {$S ListRes}
12  216 --      {-----------------------------------+
12  217 --      |   DeleteAll                       |
12  218 --      +-----------------------------------}
12  219 -- A    PROCEDURE TList.DeleteAll;
12  220 --          { Delete every element from the list, but do not free any objects. }
12  221 0- A    BEGIN
12  222 --          {$IFC qDebug}
12  223 --          IF fEachLevel > 0 THEN
12  224 --              ProgramBreak('In TList.DeleteAll: currently doing an Each call! (fEachLevel > 0)');
12  225 --          {$ENDC}
12  226 --
12  227 --          { Delete the elements from the heap block }
12  228 --          SetHandleSize(Handle(SELF), fFirstOffset);
12  229 --          fSize := 0;
12  230 -0 A    END;
12  231 --
12  232 --
12  233 --      {$S ListRes}
12  234 --      {-----------------------------------+
12  235 --      |   Each                            |
12  236 --      +-----------------------------------}
12  237 -- A    PROCEDURE TList.Each(PROCEDURE DoToItem(item: TObject));
12  238 --          { Call DoToItem once for each element of the list, in order. }
12  239 --      VAR
12  240 --          index:      INTEGER;
12  241 --          offset:     LONGINT;
12  242 --          item:       LONGINT;
12  243 --          fi:         FailInfo;
12  244 --
12  245 --          {-------------------------------+
12  246 --          |   FinishEach                  |
12  247 --          +-------------------------------}
12  248 -- B        PROCEDURE FinishEach;
12  249 0- B        BEGIN
12  250 --              fEachLevel := fEachLevel - 1;
12  251 --              IF fEachLevel = 0 THEN
12  252 --                  IF fDeletions > 0 THEN
12  253 --                      RemoveDeletions;
12  254 -0 B        END;
12  255 --
12  256 --          {-------------------------------+
12  257 --          |   HdlEach                     |
12  258 --          +-------------------------------}
12  259 -- B        PROCEDURE HdlEach(error: OSErr; message: LONGINT);
12  260 0- B        BEGIN
12  261 --              FinishEach;
12  262 -0 B        END;
12  263 --
12  264 0- A    BEGIN
12  265 --          IF fSize > 0 THEN
12  266 1-              BEGIN
12  267 --              fEachLevel := fEachLevel + 1;
12  268 --
12  269 --              CatchFailures(fi, HdlEach);
12  270 --
12  271 --              offset := fFirstOffset;
12  272 --
12  273 --              { Note that upper bound of the loop is the value of fSize at the
12  274 --                  time the loop is begun.  Changing fSize in the loop has no
12  275 --                  effect on the number of times the loop is executed. }
12  276 --
12  277 --              FOR index := 1 TO fSize DO
12  278 2-                  BEGIN
12  279 --                  { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  280 --                  { must recompute every time because of heap compaction }
12  281 --                  item := PLongInt(Ord(Handle(SELF)^) + offset)^;
12  282 --                  IF item <> kDeletedElement THEN
12  283 --                      DoToItem(TObject(item));
12  284 --                  offset := offset + kElemSize;
12  285 -2                  END;
12  286 --
12  287 --              Success(fi);
12  288 --
12  289 --              FinishEach;
12  290 -1              END;
12  291 -0 A    END;
12  292 --
12  293 --
12  294 --      {$S ListRes}
12  295 --      {-----------------------------------+
12  296 --      |   First                           |
```

```
12  297 --      +-----------------------------------}
12  298 -- A    FUNCTION  TList.First: TObject;
12  299 --              { Return the first element of the list.
12  300 --                If the compile-flag qDebug is TRUE & SetEltType was called, verify item's type. }
12  301 0- A    BEGIN
12  302 --          IF fSize <= 0 THEN
12  303 --              First := NIL
12  304 --          ELSE
12  305 --              First := At(1);
12  306 -0 A    END;
12  307 --
12  308 --
12  309 --      {$S ListRes}
12  310 --      {-----------------------------------+
12  311 --      |   FirstThat                       |
12  312 --      +-----------------------------------}
12  313 -- A    FUNCTION  TList.FirstThat(FUNCTION TestItem(item: TObject): BOOLEAN): TObject;
12  314 --      VAR
12  315 --          index:      INTEGER;
12  316 --          offset:     LONGINT;
12  317 --          anObject:   TObject;
12  318 --
12  319 0- A    BEGIN
12  320 --          offset := fFirstOffset;
12  321 --
12  322 --          FOR index := 1 TO fSize DO
12  323 1-              BEGIN
12  324 --              { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  325 --              { must recompute every time because of heap compaction }
12  326 --              anObject := PObj(Ord(Handle(SELF)^) + offset)^;
12  327 --              IF TestItem(anObject) THEN { Found }
12  328 2-                  BEGIN
12  329 --                  FirstThat := anObject;
12  330 --                  EXIT(FirstThat);
12  331 -2                  END;
12  332 --              offset := offset + kElemSize;
12  333 -1              END;
12  334 --          FirstThat := NIL;
12  335 -0 A    END;
12  336 --
12  337 --
12  338 --      {$S ListRes}
12  339 --      {-----------------------------------+
12  340 --      |   FreeAll                         |
12  341 --      +-----------------------------------}
12  342 -- A    PROCEDURE TList.FreeAll;
12  343 --
12  344 -- B        PROCEDURE FreeObject (theObject: TObject);
12  345 --
12  346 0- B        BEGIN
12  347 --              theObject.Free;
12  348 -0 B        END;
12  349 --
12  350 0- A    BEGIN
12  351 --          Each(FreeObject);
12  352 --          DeleteAll;
12  353 -0 A    END;
12  354 --
12  355 --
12  356 --      {$S ListRes}
12  357 --      {-----------------------------------+
12  358 --      |   FreeList                        |
12  359 --      +-----------------------------------}
12  360 -- A    PROCEDURE TList.FreeList;
12  361 --
12  362 -- B    .   PROCEDURE FreeObject (theObject: TObject);
12  363 --
12  364 0- B        BEGIN
12  365 --              theObject.Free;
12  366 -0 B        END;
12  367 --
12  368 0- A    BEGIN
12  369 --          Each(FreeObject);
12  370 --          Free;
12  371 -0 A    END;
12  372 --
12  373 --
12  374 --      {$S ListRes}
12  375 --      {-----------------------------------+
12  376 --      |   GetItemNumber                   |
12  377 --      +-----------------------------------}
12  378 -- A    FUNCTION  TList.GetItemNumber(item: TObject): INTEGER;
12  379 --          { Find the first reference to item in the list.
12  380 --            If item does not occur, return fSize + 1. }
12  381 --      VAR
12  382 --          p:          LONGINT;
12  383 --          i:          LONGINT;
12  384 --          maxP:       LONGINT;
12  385 --          offset:     LONGINT;
12  386 0- A    BEGIN
12  387 --          { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  388 --          p := ComputeAddress(1);     { Address of first element }
12  389 --          i := 1;
12  390 --          maxP := MaxAddress;         { Address past last element }
12  391 --
12  392 --          { Skip elements until item is found }
12  393 1-          WHILE (p < maxP) & (PObj(p)^ <> item) DO BEGIN
12  394 --              p := p + kElemSize;
12  395 --              IF p <> kDeletedElement THEN
```

```
12  396 --                  i := i + 1;
12  397 -1          END;
12  398 --
12  399 --          GetItemNumber := i;
12  400 -0 A    END;
12  401 --
12  402 --      {$S ListRes}
12  403 --      {-----------------------------------+
12  404 --      |   GetSize                         |
12  405 --      +-----------------------------------}
12  406 -- A    FUNCTION  TList.GetSize: INTEGER;
12  407 --          { Returns the number of elements in the list. }
12  408 --
12  409 0- A    BEGIN
12  410 --          GetSize := fSize;
12  411 -0 A    END;
12  412 --
12  413 --
12  414 --      {$S ListRes}
12  415 --      {-----------------------------------+
12  416 --      |   InsertBefore                    |
12  417 --      +-----------------------------------}
12  418 -- A    PROCEDURE TList.InsertBefore(index :INTEGER; item: TObject);
12  419 --          { Insert a reference to item at the indicated index.
12  420 --              If the compile-flag qDebug is TRUE & SetEltType was called, verify item's type. }
12  421 --
12  422 --      VAR
12  423 --          oldSize:    Size;
12  424 --          offset:     LONGINT;
12  425 --          permFlag:   BOOLEAN;
12  426 --
12  427 0- A    BEGIN
12  428 --      {$IFC qDebug}
12  429 --          IF fEachLevel > 0 THEN
12  430 --              ProgramBreak('In TList.InsertBefore: currently doing an Each call! (fEachLevel > 0)');
12  431 --
12  432 --          IF (index < 1) OR (index > fSize+1) THEN
12  433 1-              BEGIN
12  434 --              Writeln('fSize = ', fSize:1, '  index = ', index:1);
12  435 --              ProgramBreak('Range Check in TList.InsertBefore');
12  436 -1              END;
12  437 --
12  438 --      {$%+}
12  439 --          IF fObjPtr > 0 THEN
12  440 --              IF NOT %_InObj(ORD(item), fObjPtr) THEN
12  441 1-                  BEGIN
12  442 --                  WrLblPtr('item', item);
12  443 --                  Writeln;
12  444 --                  ProgramBreak('In TList.InsertBefore: inserting invalid object type');
12  445 -1                  END;
12  446 --      {$%-}
12  447 --      {$ENDC qDebug}
12  448 --
12  449 --          { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  450 --          oldSize := GetHandleSize(Handle(SELF));
12  451 --
12  452 --          permFlag := PermAllocation(TRUE);
12  453 --
12  454 --          { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  455 --          offset := Munger(Handle(SELF),
12  456 --                           fFirstOffset + BSL(index-1, 2),     { index-1 * 4 }
12  457 --                           NIL, 0, Ptr(@item), kElemSize);
12  458 --
12  459 --          permFlag := PermAllocation(permFlag);
12  460 --
12  461 --          IF GetHandleSize(Handle(SELF)) <= oldSize THEN
12  462 --              Failure(memFullErr, 0);
12  463 --
12  464 --          fSize := fSize + 1;
12  465 -0 A    END;
12  466 --
12  467 --
12  468 --      {$S ListRes}
12  469 --      {-----------------------------------+
12  470 --      |   InsertFirst                     |
12  471 --      +-----------------------------------}
12  472 -- A    PROCEDURE TList.InsertFirst(item: TObject);
12  473 --          { Insert a reference to item at the front of the list.
12  474 --              If the compile-flag qDebug is TRUE & SetEltType was called, verify item's type. }
12  475 0- A    BEGIN
12  476 --          InsertBefore(1, item);
12  477 -0 A    END;
12  478 --
12  479 --
12  480 --      {-----------------------------------+
12  481 --      |   InsertLast                      |
12  482 --      +-----------------------------------}
12  483 -- A    PROCEDURE TList.InsertLast(item: TObject);
12  484 --          { Insert a reference to item at the back of the list.
12  485 --              If qDebug is TRUE & SetEltType was called, check that item's type is correct. }
12  486 0- A    BEGIN
12  487 --          InsertBefore(fSize+1, item);
12  488 -0 A    END;
12  489 --
12  490 --
12  491 --      {$S ListRes}
12  492 --      {-----------------------------------+
12  493 --      |   Last                            |
12  494 --      +-----------------------------------}
```

```
12  495 -- A   FUNCTION  TList.Last: TObject;
12  496 --         { Return the last element of the list.
12  497 --             If the compile-flag qDebug is TRUE & SetEltType was called, verify item's type. }
12  498 0- A   BEGIN
12  499 --         IF fSize <= 0 THEN
12  500 --             Last := NIL
12  501 --         ELSE
12  502 --             Last := At(fSize);
12  503 -0 A   END;
12  504 --
12  505 --
12  506 --     {$S ListRes}
12  507 --     {-----------------------------------+
12  508 --     |   LastThat                        |
12  509 --     +-----------------------------------}
12  510 -- A   FUNCTION  TList.LastThat(FUNCTION TestItem(item: TObject): BOOLEAN): TObject;
12  511 --     VAR
12  512 --         index:      INTEGER;
12  513 --         offset:     LONGINT;
12  514 --         anObject:   TObject;
12  515 --
12  516 0- A   BEGIN
12  517 --         offset := fFirstOffset + (fSize - 1) * kElemSize;
12  518 --
12  519 --         FOR index := 1 TO fSize DO
12  520 1-             BEGIN
12  521 --             { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  522 --             { must recompute every time because of heap compaction }
12  523 --             anObject := PObj(Ord(Handle(SELF)^) + offset)^;
12  524 --             IF TestItem(anObject) THEN { Found }
12  525 2-                 BEGIN
12  526 --                 LastThat := anObject;
12  527 --                 EXIT(LastThat);
12  528 -2                 END;
12  529 --             offset := offset - kElemSize;
12  530 -1             END;
12  531 --         LastThat := NIL;
12  532 -0 A   END;
12  533 --
12  534 --
12  535 --         { Return address after last element of list }
12  536 --     {$S ListRes}
12  537 --     {-----------------------------------+
12  538 --     |   MaxAddress                      |
12  539 --     +-----------------------------------}
12  540 -- A   FUNCTION  TList.MaxAddress: LONGINT;
12  541 0- A   BEGIN
12  542 --         MaxAddress := Ord(Handle(SELF)^) + GetHandleSize(Handle(SELF));
12  543 -0 A   END;
12  544 --
12  545 --
12  546 --     {$S ListRes}
12  547 --     {-----------------------------------+
12  548 --     |   RemoveDeletions                 |
12  549 --     +-----------------------------------}
12  550 -- A   PROCEDURE TList.RemoveDeletions;
12  551 --     VAR
12  552 --         srcP:   LONGINT;
12  553 --         dstP:   LONGINT;
12  554 --         maxP:   LONGINT;
12  555 --         elt:    LONGINT;
12  556 0- A   BEGIN
12  557 --         { See ELEMENT-ADDRESS-COMPUTATION comment above }
12  558 --         { Can't use ComputeAddress because it would bypass deleted elements. }
12  559 --         srcP := Ord(Handle(SELF)^) + fFirstOffset;  { Address of first physical item }
12  560 --         dstP := srcP;
12  561 --         maxP := MaxAddress;                         { Address past last element }
12  562 --
12  563 --         { Compact the non deleted elements to start of list. }
12  564 --         WHILE srcP < maxP DO
12  565 1-             BEGIN
12  566 --             elt := PLongInt(srcP)^;
12  567 --             IF elt <> kDeletedElement THEN
12  568 2-                 BEGIN
12  569 --                 PLongInt(dstP)^ := elt;
12  570 --                 dstP := dstP + kElemSize;
12  571 -2                 END;
12  572 --             srcP := srcP + kElemSize;
12  573 -1             END;
12  574 --
12  575 --         fDeletions := 0;
12  576 --         SetHandleSize(Handle(SELF), fFirstOffset + BSL(fSize, 2));  {fSize * 4}
12  577 -0 A   END;
12  578 --
12  579 --
12  580 --     {$%+}
12  581 --     {$IFC qDebug}
12  582 --     {$S ListDebug}
12  583 --     {-----------------------------------+
12  584 --     |   SetEltType                      |
12  585 --     +-----------------------------------}
12  586 -- A   PROCEDURE TList.SetEltType(toClass: Str255);
12  587 --         { Initialize fEltClass to toClass. fEltClass is a STRING[8] because identifiers
12  588 --             are unique to 8 chars }
12  589 --     VAR
12  590 --         s:  String8;
12  591 0- A   BEGIN
12  592 --         { Truncate to 8 characters, and force upper case }
12  593 --         IF Length(toClass) > 8 THEN
```

```
12  594 --              s := Copy(toClass, 1, 8)
12  595 --          ELSE
12  596 --              s := toClass;
12  597 --          UprString8(s);
12  598 --
12  599 --          { Forgive the caller for calling this twice only if the same type was specified both times }
12  600 --          IF fEltClass <> '' THEN
12  601 --              IF fEltClass <> s THEN
12  602 1-                  BEGIN
12  603 --                  Writeln('The list already contains ', fEltClass, '; trying to change to ', s);
12  604 --                  ProgramBreak('In TList.SetEltType');
12  605 -1                  END;
12  606 --
12  607 --          { Assign the field }
12  608 --          fEltClass := s;
12  609 --          fObjPtr := %_GetA5 + FindObjID(s);
12  610 -0 A    END;
12  611 --      {$ENDC}
12  612 --      {$%-}
12  613 --
12  614 --
12  615 --      {$IFC qDebug}
12  616 --      {$S MAFields}
12  617 --      {-----------------------------------+
12  618 --      |   Fields                          |
12  619 --      +-----------------------------------}
12  620 -- A    PROCEDURE TList.Fields (PROCEDURE DoToField (fieldName: Str255;
12  621 --                                                   fieldAddr: Ptr;
12  622 --                                                   fieldType: INTEGER)); OVERRIDE;
12  623 --      VAR
12  624 --          i:          INTEGER;
12  625 --          aString:    Str255;
12  626 --
12  627 0- A    BEGIN
12  628 --          DoToField('TList', NIL, bClass);
12  629 --          DoToField('fSize', @fSize, bInteger);
12  630 --          DoToField('fFirstOffset', @fFirstOffset, bLongInt);
12  631 --          DoToField('fDeletions', @fDeletions, bInteger);
12  632 --          DoToField('fEachLevel', @fEachLevel, bInteger);
12  633 --          DoToField('fEltClass', @fEltClass, bString);
12  634 --          DoToField('fObjPtr', @fObjPtr, bHexLongInt);
12  635 --          FOR i := 1 TO Min(fSize, 50) DO
12  636 1-              BEGIN
12  637 --              NumToString(i, aString);
12  638 --              aString := CONCAT('.At[', aString, ']');
12  639 --              DoToField(aString, Ptr(ComputeAddress(i)), bObject);
12  640 -1              END;
12  641 --          INHERITED Fields(DoToField);
12  642 -0 A    END;
12  643 --      {$ENDC}
12  644 --
12  645 --
12  646 --      {$IFC qInspector}
12  647 --      {$S MAInspector}
12  648 --      {-----------------------------------+
12  649 --      |   GetInspectorName                |
12  650 --      +-----------------------------------}
12  651 -- A    PROCEDURE TList.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
12  652 --
12  653 0- A    BEGIN
12  654 --          IF fEltClass <> '' THEN
12  655 --              inspectorName := CONCAT('Of ', fEltClass);
12  656 -0 A    END;
12  657 --      {$ENDC}
12  658 --
12  659 --
12  660 --      {$S MANever}
12  661 --      {-----------------------------------+
12  662 --      |   Compare                         |
12  663 --      +-----------------------------------}
12  664 -- A    FUNCTION  TSortedList.Compare (item1, item2: TObject): INTEGER;
12  665 --
12  666 0- A    BEGIN
12  667 --      {$IFC qDebug}
12  668 --          ProgramBreak('TSortedList.Compare must be overridden');
12  669 --      {$ENDC}
12  670 -0 A    END;
12  671 --
12  672 --
12  673 --      {$S ListRes}
12  674 --      {-----------------------------------+
12  675 --      |   GetItemNumber                   |
12  676 --      +-----------------------------------}
12  677 -- A    FUNCTION TSortedList.GetItemNumber (item: TObject): INTEGER; OVERRIDE;
12  678 --
12  679 --      VAR
12  680 --          index:          INTEGER;
12  681 --          p:              LONGINT;
12  682 --          low,
12  683 --          high:           INTEGER;
12  684 --          compareResult:  INTEGER;
12  685 --
12  686 0- A    BEGIN
12  687 --          IF fSize = 0 THEN
12  688 --              index := 1
12  689 --          ELSE IF fDeletions > 0 THEN
12  690 --              index := INHERITED GetItemNumber(item)
12  691 --          ELSE
12  692 1-              BEGIN
```

```
12  693 --              low := 1;
12  694 --              high := fSize;
12  695 2-              REPEAT
12  696 --                  index := BSR(low + high, 1);            { (low + high) DIV 2 }
12  697 --                  compareResult := Compare(item, At(index));
12  698 --                  IF compareResult < 0 THEN
12  699 --                      high := index - 1
12  700 --                  ELSE
12  701 --                      low := index + 1;
12  702 -2              UNTIL (compareResult = 0) | (low > high);
12  703 --              IF compareResult > 0 THEN
12  704 --                  index := index + 1;
12  705 -1              END;
12  706 --
12  707 --          GetItemNumber := index;
12  708 -0 A     END;
12  709 --
12  710 --
12  711 --      {$S ListRes}
12  712 --      {-----------------------------------+
12  713 --      |   Insert                          |
12  714 --      +-----------------------------------}
12  715 -- A    PROCEDURE TSortedList.Insert (item: TObject);
12  716 --
12  717 0- A    BEGIN
12  718 --          InsertBefore(GetItemNumber(item), item);
12  719 -0 A    END;
12  720 --
12  721 --
12  722 --      {$S ListRes}
12  723 --      {-----------------------------------+
12  724 --      |   Search                          |
12  725 --      +-----------------------------------}
12  726 -- A    FUNCTION TSortedList.Search (FUNCTION TestItem(anItem: TObject): INTEGER): TObject;
12  727 --
12  728 --      VAR
12  729 --          index:          INTEGER;
12  730 --          p:              LONGINT;
12  731 --          low,
12  732 --          high:           INTEGER;
12  733 --          theItem:        TObject;
12  734 --          compareResult:  INTEGER;
12  735 --
12  736 -- B        FUNCTION IsThisTheItem (anItem: TObject): BOOLEAN;
12  737 0- B       BEGIN
12  738 --              IsThisTheItem := TestItem(anItem) = 0;
12  739 -0 B       END;
12  740 --
12  741 0- A    BEGIN
12  742 --          IF fSize > 0 THEN
12  743 --              IF fDeletions > 0 THEN
12  744 1-                  BEGIN
12  745 --                  Search := FirstThat(IsThisTheItem);
12  746 --                  EXIT(Search);
12  747 -1                  END
12  748 --              ELSE
12  749 1-                  BEGIN
12  750 --                  low := 1;
12  751 --                  high := fSize;
12  752 --                  WHILE low <= high DO
12  753 2-                      BEGIN
12  754 --                      index := BSR(low + high, 1);            { (low + high) DIV 2 }
12  755 --                      theItem := At(index);
12  756 --                      compareResult := TestItem(theItem);
12  757 --                      IF compareResult = 0 THEN
12  758 3-                          BEGIN
12  759 --                          Search := theItem;
12  760 --                          EXIT(Search);
12  761 -3                          END
12  762 --                      ELSE IF compareResult < 0 THEN
12  763 --                          high := index - 1
12  764 --                      ELSE
12  765 --                          low := index + 1;
12  766 -2                      END;
12  767 -1                  END;
12  768 --
12  769 --          Search := NIL;
12  770 -0 A    END;
11  313 --
11  314 --      END.
11  315 --
```

```
13    1 --    {$P}
13    2 --    { UMacApp.p }
13    3 --    { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
13    4 --
13    5 --    { ??? Comments that contain '???' (such as this one) indicate issues that need
13    6 --       to be resolved. ??? }
13    7 --
13    8 --               { You should always override TApplication.DoMakeDocument.
13    9 --                 You should always override TDocument.DoMakeViews.
13   10 --                 You should always override TDocument.DoMakeWindows.
13   11 --                 -
13   12 --                  }
13   13 --
13   14 --
13   15 --    UNIT UMacApp;
13   16 --    { Unit UMacApp defines object types:
13   17 --        TEvtHandler         An abstract type that can handle events and commands.
13   18 --        TApplication        Its sole instance polls the user and network for events,
13   19 --                            and manages the desktop.
13   20 --        TDocument           Corresponds to a Finder document.  Manages document
13   21 --                            data in files and in main memory.
13   22 --        TWindow             Corresponds to a desktop Window.  The outermost of a
13   23 --                            nested set of frames.
13   24 --        TView               Renders an image of a document or other data, and
13   25 --                            identifies mouse-clicked objects.
13   26 --        TScroller           A view which can "scroll" its contents by effecting a
13   27 --                            coordinate transformation.
13   28 --        TPrintHandler       Handles printing at a specified resolution on behalf of
13   29 --                            a view.
13   30 --        TCommand            Handles a non-trivial user command, including undoing
13   31 --                            and mouse-tracking if required.
13   32 --        TDeskScrapView      The default view in the Clipboard, able to display only
13   33 --                            standard text and pictures.
13   34 --
13   35 --     as well as numerous other types, constants and global procedures for use by
13   36 --     MacApp and applications}
13   37 --
13   38 --    INTERFACE
13   39 --
13   40 --    {-----------------------------------+
13   41 --    |   Uses                            |
13   42 --    +-----------------------------------}
13   43 --    USES
13   44 --        {$LOAD MacIntf.LOAD}
13   45 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
13   46 --        {$LOAD}
13   47 --            Script,
13   48 --            UMAUtil,
13   49 --            UViewCoords,
13   50 --            UPatch,
13   51 --            UBusyCursor,
13   52 --            UFailure,
13   53 --            UMemory,
13   54 --            UMenuSetup,
13   55 --            UObject,
13   56 --            UList,
13   57 --            UAssociation
13   58 --    {$IFC qTrace}
13   59 --            ,UTrace
13   60 --    {$ENDC}
13   61 --    {$IFC qWWNeeded}
13   62 --            ,UWritelnWindow
13   63 --    {$ENDC}
13   64 --            ;
13   65 --
13   66 --    {-----------------------------------+
13   67 --    |   Constants                       |
13   68 --    +-----------------------------------}
13   69 --    CONST
13   70 --        kCopyright       =  'Copyright © 1984, 1985, 1986, Apple Computer Inc.';
13   71 --
13   72 --    { M E N U S }
13   73 --        kMBarDisplayed      = 128;  { Default 'MBAR' id of the menus that are
13   74 --                                      read in and installed in menu bar }
13   75 --        kMBarNotDisplayed   = 129;  { Default 'MBAR' id of the menus that are
13   76 --                                      read in but not installed }
13   77 --        kMBarHierarchical   = 130;  { Default 'MBAR' id of the  menus that pop
13   78 --                                      up when a menu item is choosen }
13   79 --
13   80 --
13   81 --    { C O M M A N D S }
13   82 --
13   83 --       { Predefined command numbers ??? Renumber these more logically ??? }
13   84 --       { Not all of these are caught by MacApp.
13   85 --            "<!>" means MacApp catches it.
13   86 --            "<&>" means TEView catches it.
13   87 --            "<->" means the application must catch it if it is used as a
13   88 --                  command number in any menu."
13   89 --
13   90 --    {Special command codes }
13   91 --
13   92 --        cNoCommand      =        0;     { <!> Command number representing no
13   93 --                                          command }
13   94 --        cCantUndo       =       -1;     { <!> Passed to TApplication.SetUndoText
13   95 --                                          to indicate that the command cannot
13   96 --                                          be undone. }
13   97 --
13   98 --    { Apple-menu commands }
```

```
13   99 --
13  100 --          cAboutApp       =        1;      { <!> "About <appname>…" }
13  101 --
13  102 --      { File-menu filing commands }
13  103 --
13  104 --          cNew            =       10;      { <!> "New" (??? reserve a range of NEW
13  105 --                                               commands ???) }
13  106 --          cNewLast        =       19;
13  107 --          cOpen           =       20;      { <!> "Open …" (??? reserve a range of
13  108 --                                               OPEN commands ???) }
13  109 --          cOpenLast       =       29;
13  110 --
13  111 --          cSave           =       30;      { <!> "Save" }
13  112 --          cClose          =       31;      { <!> "Close" }
13  113 --          cSaveAs         =       32;      { <!> "Save as…" }
13  114 --          cSaveCopy       =       33;      { <!> "Save a Copy in …" }
13  115 --          cRevert         =       34;      { <!> "Revert" (to previous version) }
13  116 --          cShowClipboard  =       35;      { <!> "Show Clipboard"/"Hide Clipboard" }
13  117 --          cQuit           =       36;      { <!> "Quit" }
13  118 --
13  119 --
13  120 --      { Edit-menu commands }
13  121 --
13  122 --              { For the following command numbers, we must guarantee that
13  123 --                <command number> - cEditBase = <appropriate number to pass to
13  124 --                SystemEdit>.  This relationship is enforced in
13  125 --                TApplication.IApplication. }
13  126 --          cEditBase       =       101;     { start of standard editing commands }
13  127 --          cUndo           =       101;     { <!> "Undo <command>"/"Redo <command>" }
13  128 --          cEditSep        =       102;     { line separating UNDO from CUT }
13  129 --          cCut            =       103;     { <&> "Cut" }
13  130 --          cCopy           =       104;     { <&> "Copy" }
13  131 --          cPaste          =       105;     { <&> "Paste" }
13  132 --          cClear          =       106;     { <-> "Clear" }
13  133 --          cEditLast       =       cClear;
13  134 --
13  135 --          cSelectAll      =       110;     { <&> "Select All" }
13  136 --
13  137 --          cTyping         =       120;     { for use in a TTypingCommand }
13  138 --          cMouseCommand   =       121;     { generic mouse command }
13  139 --
13  140 --          cStyleChange    =       130;     { "Menu" command for style change }
13  141 --          cRememberStyle  =       131;     { "Menu" command for remembering present style }
13  142 --
13  143 --      { Finder pseudo-commands }
13  144 --
13  145 --          cFinderNew      =       40;      { <!> The user selected the tool icon in
13  146 --                                               the Finder and chose "Open" }
13  147 --          cFinderPrint    =       41;      { <!> The user selected document icons
13  148 --                                               in the Finder and chose "Print" }
13  149 --          cFinderOpen     =       42;      { <!> The user selected document icons
13  150 --                                               in the Finder and chose "Open" }
13  151 --
13  152 --
13  153 --      { File-menu printing commands  }
13  154 --
13  155 --          cPrFileBase     =      176;      { Command numbers between cPrFileBase
13  156 --                                               and cPrFileMax are sent to a
13  157 --                                               document's fDocPrintHandler even if
13  158 --                                               it is not in the fTarget chain }
13  159 --          cPrFileMax      =      195;
13  160 --
13  161 --          cPageSetup      =      176;      { <!> "Page Setup…" }
13  162 --          cPrintOne       =      177;      { <!> "Print One" }
13  163 --          cPrint          =      178;      { <!> "Print…" }
13  164 --          cPrintToFile    =      179;      { <!> "Print to file…" }
13  165 --
13  166 --          cPrintSpoolFile =      190;      { <-> "Print spooled file…" }
13  167 --
13  168 --          cPrViewBase     =      201;      { Command numbers between cPrViewBase and
13  169 --                                               cPrViewMax are printing commands
13  170 --                                               applied to a displayed view in the
13  171 --                                               fTarget chain }
13  172 --          cPrViewMax      =      250;
13  173 --
13  174 --          cShowBorders    =      199;      { <!> Toggle. "Show view borders"--also
13  175 --                                               usable outside Debug menu }
13  176 --
13  177 --
13  178 --      { Zooming commands }    { Not being used at present }
13  179 --
13  180 --          cReduce50       =      301;      { <-> "Reduce 50%" }
13  181 --          cReduceToFit    =      302;      { <-> "Reduce to Fit" }
13  182 --          cShowFullSize   =      303;      { <-> "Show Full Size" }
13  183 --
13  184 --        { Control tracking }
13  185 --          cTrackingControl    =  400;
13  186 --
13  187 --        { Debug-menu commands.  Flag toggled by "cFoo" is called "gFoo" for any
13  188 --          "Foo" }
13  189 --
13  190 --      {$IFC qDebug}
13  191 --          cIdentifySoftware   =  900;      { <!> Action. "Show Software Version"--
13  192 --                                               WriteLn version info }
13  193 --          cExperimenting      =  901;      { <!> Toggle. "Enable experimental
13  194 --                                               features" }
13  195 --          cReportMenuChoices  =  902;      { <!> Toggle. "Trace menu commands"--
13  196 --                                               WriteLn menu choices }
13  197 --          cIntenseDebugging   =  904;      { <!> Toggle. "Intense Debugging" }
```

```
13  198 --          cTraceSetupMenus    =   905;    { <!> Toggle. "Allow Trace of Menu
13  199 --                                              Setups" if gTrace is also on }
13  200 --          cTraceIdle          =   906;    { <!> Toggle. "Allow Trace during Idle"
13  201 --                                              if gTrace is also on }
13  202 --          cDebugPrinting      =   912;    { <!> Toggle. "Debug Printing" }
13  203 --          cReportEvt          =   914;    { <!> Toggle. "Report Events"--WriteLn
13  204 --                                              events received from queue }
13  205 --          cDoFirstClick       =   915;    { <!> Toggle. "'Do First Click' for this
13  206 --                                              window" }
13  207 --          cVarClipPicSize     =   916;    { <!> Toggle. "Scale Pictures in
13  208 --                                              Clipboard to Window" }
13  209 --          cRefreshFrontWindow =   917;    { <!> Action. "Refresh Front Window" }
13  210 --          cNewInspectorWindow =   918;    { <!> Action. "Open new inspector window" }
13  211 --          cModalToggle =          919;    { <!> Toggle. "Make front window modal/modeless"}
13  212 --      {$ENDC}
13  213 --      {$IFC qWWNeeded}
13  214 --          cDebugWind          =   913;    { <!> Action. "Show Debug Window" }
13  215 --      {$ENDC}
13  216 --
13  217 --      { D E B U G  F L A G S }
13  218 --
13  219 --      {$IFC NOT qDebug} { These start with 'g' because they are global variables when
13  220 --                          qDebug is TRUE }
13  221 --          gExperimenting      =       FALSE;
13  222 --          gReportMenuChoices  =       FALSE;
13  223 --          gIntenseDebugging   =       FALSE;
13  224 --          gDebugPrinting      =       FALSE;
13  225 --          gReportEvt          =       FALSE;
13  226 --
13  227 --          gMemMgtReport       =       FALSE;
13  228 --      {$ENDC}
13  229 --
13  230 --
13  231 --
13  232 --      { A L E R T S }
13  233 --
13  234 --          errReasonID     =       128;    { resource ID of errs resource describing
13  235 --                                              error messages }
13  236 --          errRecoveryID   =       129;    { resource ID of errs resource describing
13  237 --                                              recovery messages }
13  238 --          errOperationsID =       130;    { resource ID of errs resource containing
13  239 --                                              operation strings }
13  240 --          errAppTable     =       1000;   { added to the MacApp resource ID to get
13  241 --                                            the ID of an application error table }
13  242 --
13  243 --          msgCmdErr       =       0;      { when added to a command number, gives
13  244 --                                              a message value to display the
13  245 --                                              alert "Could not complete the …
13  246 --                                              command …" }
13  247 --          msgAlert        =       $FFFF0000;
13  248 --                                      { when added to an alert number, gives a
13  249 --                                          message value to display that alert }
13  250 --          msgLookup       =       $FFFE0000;
13  251 --                                      { when added to an INTEGER, gives a
13  252 --                                          message value that will look up the
13  253 --                                          INTEGER in table errOperationsID
13  254 --                                          and display the alert "Could not …
13  255 --                                          because …" }
13  256 --          msgAltRecovery  =       $FFFD0000;
13  257 --                                      { same as msgLookup, except the INTEGER
13  258 --                                          is used instead of the error number
13  259 --                                          to lookup the recovery string }
13  260 --
13  261 --          msgStrList      =       200 * $10000;   { Standard list of operation
13  262 --                                                      strings used by MacApp }
13  263 --          msgCancelled    =       msgStrList;     { used with error = noErr if user
13  264 --                                                      cancels Save, Quit, … }
13  265 --          msgInitFailed   =       msgStrList + 1;
13  266 --          msgSaveFailed   =       msgStrList + 2;
13  267 --          msgRevertFailed =       msgStrList + 3;
13  268 --          msgPrintFailed  =       msgStrList + 4;
13  269 --          msgNewFailed    =       msgStrList + 5;
13  270 --          msgOpenFailed   =       msgStrList + 6;
13  271 --          msgSaveAsFailed =       msgStrList + 7;
13  272 --          msgSaveCopyFailed =     msgStrList + 8;
13  273 --          msgDrawFailed   =       msgStrList + 9;
13  274 --          msgImportClipFailed =   msgStrList + 10;
13  275 --          msgExportClipFailed =   msgStrList + 11;
13  276 --
13  277 --
13  278 --      { error codes used in MacApp; applications should use errors between
13  279 --          -25000 and -29999 }
13  280 --
13  281 --          errSpooling     =       -20000;     { print spooling failed OSErr }
13  282 --          errRevertFNF    =       -20001;     { Revert failed because file was
13  283 --                                                  deleted }
13  284 --          errFileChanged  =       -20002;     { disk file modification date
13  285 --                                                  changed }
13  286 --          errSaveAgain    =       -20003;     { Save As / Save a Copy in an opened
13  287 --                                                  document }
13  288 --          errFTypeChanged =       -20004;     { disk file type changed }
13  289 --          errNoPrintDrvr  =       -20005;     { print driver file not found }
13  290 --          errNotMyType    =       -20006;     { can't open this file type }
13  291 --
13  292 --          { Alerts 0 - 249 are reserved by MacApp. }
13  293 --
13  294 --          phGenError      =       100;    { 'Could not ^2, because ^0.  ^1' }
13  295 --          phCmdErr        =       101;    { 'Could not complete the "^2" command because ^0.  ^1'. }
13  296 --          phUnknownErr    =       102;    { 'Could not complete your request
```

```
13  297 --                                              because ^0.  ^1'; only used if no
13  298 --                                              operation is supplied. }
13  299 --          phSaveChanges   =       110;    { 'Save Changes Before Closing?' }
13  300 --          phRevert        =       111;    { 'Revert to last version saved?' }
13  301 --          phFileChanged   =       112;    { 'File changed since last save.' }
13  302 --          phPurgeOld      =       113;    { 'OK to purge old version before saving new one?' }
13  303 --          phReopenDoc     =       114;    { 'Document is already open.' }
13  304 --          phSpaceIsLow    =       115;    { 'Space is low' }
13  305 --          phUnsupportedROMs =     116;    { 'Can't start app because your ROMs are too old.' }
13  306 --          phOfferReadOnly =       117;    { 'Can't open for file for write.  Open read-only?' }
13  307 --
13  308 --          phTooManyChars  =       150;    { 'Too Many Characters', used to reject
13  309 --                                              paste or keystrokes in UDialog and
13  310 --                                              UTEView }
13  311 --
13  312 --          phStylesTooBig  =       151;    { 'Too Many Styles', used to reject the
13  313 --                                              styles portion of a paste in UTEView  }
13  314 --
13  315 --          phAboutApp      =       201;    { Can hold the 'About <appName>…'
13  316 --                                              message }
13  317 --
13  318 --      {$IFC qDebug}
13  319 --          phUnimplemented =       202;    { 'This feature not yet implemented' }
13  320 --      {$ENDC}
13  321 --
13  322 --      { M A C A P P   B U Z Z - S T R I N G   R E S O U R C E }
13  323 --          kIDBuzzString   =       250;    { std list of buzzwords stored as
13  324 --              'STR#' resources.  This string list is reserved for MacApp; you
13  325 --              should choose a different resource ID for your strings. }
13  326 --
13  327 --          { Indices for strings in the buzz-string resource }
13  328 --          bzSaveAs        =       1;      { 'Save This Document As:' }
13  329 --          bzSaveCopy      =       2;      { 'Save a Copy In:' }
13  330 --          bzShowClip      =       3;      { 'Show Clipboard' }
13  331 --          bzHideClip      =       4;      { 'Hide Cllpboard' }
13  332 --          bzUndo          =       5;      { 'Undo x' }
13  333 --          bzRedo          =       6;      { 'Redo x' }
13  334 --          bzCantDraw      =       7;      { 'Unable to draw contents of window' }
13  335 --          bzUntitled      =       8;      { name of untitled document; if blank
13  336 --              then this is taken from the window title template; if this includes
13  337 --              the string '<<<>>>' then MacApp will substitute a number for the
13  338 --              string (e.g., if the buzz string contains 'Untitled-<<<>>>' then
13  339 --              untitled windows will be named 'Untitled-1', Untitled-2, … }
13  340 --          bzClosing       =       9;      { 'closing' }
13  341 --          bzQuitting      =       10;     { 'quitting' }
13  342 --          bzCantUndo      =       11;     { 'Can't Undo' }
13  343 --          bzSaveAnyways   =       12;     { 'save' }
13  344 --          bzRevertAnyways =       13;     { 'revert' }
13  345 --
13  346 --      {$IFC qDebug}
13  347 --      { D E B U G   B U Z Z - S T R I N G   R E S O U R C E }
13  348 --          kDebugBuzzStrings =     251;    { Debug buzzwords STR# resource }
13  349 --          bzMakeModal     =       1;      { 'Make Front Window Modal' }
13  350 --          bzMakeModeless  =       2;      { 'Make Front Window Modeless' }
13  351 --          bzDoFirstClick  =       3;      { 'Do First Click For This Window' }
13  352 --          bzDontDoFirstClick =    4;      { 'Don't Do First Click For This Window' }
13  353 --      {$ENDC}
13  354 --
13  355 --      { R E S O U R C E   I D S }
13  356 --          kIDClipWindow       =       200;    { window displaying the Clipboard }
13  357 --      {$IFC qWWNeeded}
13  358 --          kDebugParamsID      =       300;    { debug window }
13  359 --      {$ENDC}
13  360 --
13  361 --
13  362 --
13  363 --      { H I G H L I G H T I N G }
13  364 --          hlOff       =   1;  { the selection is to be unhighlighted }
13  365 --          hlDim       =   2;  { the selection is to be dimly highlighted }
13  366 --          hlOn        =   4;  { the selection is to be fully highlighted }
13  367 --
13  368 --          { tnese constants can be used to test for combinations of fromHL+toHL,
13  369 --              when you do not care which is from and which is to }
13  370 --          hlOffDim    =   hlOff + hlDim;
13  371 --          hlDimOff    =   hlOffDim;
13  372 --          hlDimOn     =   hlDim + hlOn;
13  373 --          hlOnDim     =   hlDimOn;
13  374 --          hlOffOn     =   hlOff + hlOn;
13  375 --          hlOnOff     =   hlOffOn;
13  376 --
13  377 --
13  378 --      { C H O I C E S }
13  379 --          mOKHit              =   1;
13  380 --          mCancelHit          =   2;
13  381 --          mButtonHit          =   3;
13  382 --          mCheckBoxHit        =   4;
13  383 --          mClusterHit         =   5;
13  384 --          mEditTextHit        =   6;
13  385 --          mIconHit            =   7;
13  386 --          mListItemHit        =   8;
13  387 --          mListScrollBarHit   =   9;
13  388 --          mPictureHit         =   10;
13  389 --          mPopupHit           =   11;
13  390 --          mRadioHit           =   12;
13  391 --          mStaticTextHit      =   13;
13  392 --          mHScrollBarHit      =   14;
13  393 --          mVScrollBarHit      =   15;
13  394 --          mEditTabKey         =   16;
13  395 --          mEditReturnKey      =   17;
```

```
13  396 --          mEditEnterKey       =   18;
13  397 --          mPatternHit         =   19;
13  398 --          mControlHit         =   20;
13  399 --
13  400 --          kNoIdentifier       =   '    ';
13  401 --          kNoTemplate         =   -1;
13  402 --          kNoResource         =   -1;
13  403 --
13  404 --      { M I S C E L L A N E O U S }
13  405 --
13  406 --          kYesButton              =   1;    { the 'Yes' button in various dialogs }
13  407 --          kNoButton               =   3;    { the 'No' button in various dialogs }
13  408 --
13  409 --          kLowSpaceInterval   =       2*60*60;    { default low space msg interval (2 seconds) }
13  410 --
13  411 --          kStdScrollUnit      =       16;     { default value of fScrollUnit used by scrollers }
13  412 --
13  413 --          kSBarSize           =       16;
13  414 --          kSBarSizeMinus1     =       kSBarSize - 1;
13  415 --                                          { !!! Prefer the above names }
13  416 --          kStdSzSBar          =       16;     { width/height of a standard
13  417 --                                              vertical/horizontal scroll bar }
13  418 --          kStdSzMinus1SBar    =       kStdSzSBar - 1;
13  419 --
13  420 --          kStdStaggerAmount   =       16;     { standard # of pixels to stagger windows }
13  421 --
13  422 --          kPrintInfoSize      =       120;    { size in bytes of printInfo record }
13  423 --
13  424 --              { size in bytes of the overhead for a resource file;
13  425 --                  kRsrcFileOverhead is added in TDocument.DoNeedDiskSpace if the
13  426 --                  document uses the rsrc fork.
13  427 --                  You should add kRsrcTypeOverhead for each unique type you use
13  428 --                  plus kRsrcOverhead for each resoruce you use
13  429 --                  plus the total bytes in all the resources
13  430 --                  plus the length of all the names for named resources. }
13  431 --          kRsrcFileOverhead   =       256 +   { resource header & stuff }
13  432 --                                      30      { fixed part of resource map };
13  433 --          kRsrcTypeOverhead   =       8;      { overhead for each resource type }
13  434 --          kRsrcOverhead       =       16;     { overhead for each resource }
13  435 --
13  436 --      { ??? } kViewRsrcExpandAmt  = $400; { ??? Is 1K enough for expansion, or too much maybe??? }
13  437 --
13  438 --
13  439 --      {$IFC qWWNeeded}
13  440 --      { debugging info }
13  441 --          kDebugFont          =       monaco; { Font for Debug Window }
13  442 --          kDebugSize          =       9;      { Font size for Debug Window }
13  443 --      {$ENDC}
13  444 --
13  445 --              { argument of TDocument.DoMakeViews }
13  446 --          kForDisplay         =   FALSE;
13  447 --          kForPrinting        =   NOT kForDisplay;
13  448 --
13  449 --              { arguments of TFrame.IScroller }
13  450 --          kWantHScrollBar     =   TRUE;
13  451 --          kWantVScrollBar     =   TRUE;
13  452 --
13  453 --              { arguments of TDocument.Save }
13  454 --          kAskForFilename     =   TRUE;
13  455 --          kMakingCopy         =   TRUE;
13  456 --          kSwitchToTarget     =   NOT kMakingCopy;
13  457 --
13  458 --              { argument of NewPaletteWindow }
13  459 --          kLeftPalette        =   h;
13  460 --          kTopPalette         =   v;
13  461 --
13  462 --              { argument of TApplication.SetUndoText }
13  463 --          kShowUndo           =   TRUE;
13  464 --          kShowRedo           =   FALSE;
13  465 --          kShowCantUndo       =   kShowUndo;
13  466 --
13  467 --              { argument of TDocument.IDocument }
13  468 --          kUsesDataFork       =   TRUE;
13  469 --          kUsesRsrcFork       =   TRUE;
13  470 --          kDataOpen           =   TRUE;
13  471 --          kRsrcOpen           =   TRUE;
13  472 --
13  473 --              { argument of TDocument.OpenDocFile }
13  474 --          kKeepingOpen        =   TRUE;
13  475 --          kReadingFile        =   TRUE;
13  476 --
13  477 --          kVisible            =   TRUE;
13  478 --          kInvisible          =   FALSE;
13  479 --
13  480 --          kRedraw         =   TRUE;               { redraw after a requested state change         }
13  481 --          kDontRedraw     =   FALSE;              { don't redraw after a requested state change   }
13  482 --
13  483 --              { MultiFinder event message masks }
13  484 --          kSuspendOrResume =      $01;
13  485 --          kMouseMovedMessage =    $FA;
13  486 --
13  487 --          kMaxSignatures =        32;
13  488 --
13  489 --      {-----------------------------------+
13  490 --      |   Types                           |
13  491 --      +-----------------------------------}
13  492 --      TYPE
13  493 --          Integer8            = 0..7;             { For packed fields in view templates           }
13  494 --          Integer256          = 0..255;
```

```
13  495 --
13  496 --      { E V E N T S }
13  497 --
13  498 --        PEventRecord =  ^EventRecord;           { EventRecord is a ToolBox type                 }
13  499 --
13  500 --        EventInfo =     RECORD   { stores more information about the event in an unpacked form  }
13  501 --                            thePEvent:        PEventRecord; { pointer to the original
13  502 --                                                              packed ToolBox event record       }
13  503 --                            theBtnState:      BOOLEAN;  { bits found in the
13  504 --                                                          modifiers field of an EventRecord     }
13  505 --                            theCmdKey:        BOOLEAN;  { Was Command key depressed?          }
13  506 --                            theShiftKey:      BOOLEAN;  { Was Shift key depressed?            }
13  507 --                            theAlphaLock:     BOOLEAN;  { Was Alpha Lock depressed?           }
13  508 --                            theOptionKey:     BOOLEAN;  { Was Option key depressed?           }
13  509 --                            theControlKey:    BOOLEAN;  { Was Control key depressed?          }
13  510 --                            theAutoKey:       BOOLEAN;  { TRUE if this was an auto key event  }
13  511 --
13  512 --                            theClickCount:    INTEGER;  { 0  = event was not a mouse
13  513 --                                                            down;
13  514 --                                                        1-N = # of multiple clicks          }
13  515 --                        END;
13  516 --
13  517 --        IdlePhase = (idleBegin, idleContinue, idleEnd);
13  518 --        TrackPhase = (trackPress, trackMove, trackRelease);
13  519 --
13  520 --
13  521 --      { V I E W   C O O R D I N A T E S }
13  522 --
13  523 --        SizeDeterminer =    { tells how a view's size is to be determined; specified
13  524 --                              separately in each dimension. }
13  525 --            (sizeSuperView,     { View is the same size as its superview }
13  526 --            sizeRelSuperView,   { View size is relative to the superview's size }
13  527 --            sizePage,           { View to be the size of one page }
13  528 --            sizeFillPages,      { View to grow upward to fill an exact number of pages }
13  529 --            sizeVariable,       { View size fluctuates according to app-specific criteria }
13  530 --            sizeFixed);         { No special default handling of size issues }
13  531 --
13  532 --
13  533 --      { H I G H L I G H T I N G }
13  534 --        HLState = hlOff..hlOn;
13  535 --
13  536 --
13  537 --      { D O C U M E N T S }
13  538 --            { How to save a document in place, if MacApp decides this is needed.
13  539 --                sipNever:   never save on top of old file (ie., save will fail if
13  540 --                            there is not enough disk space)
13  541 --                sipAlways:  save on top of old file without asking user
13  542 --                sipAskUser: save on top, but only if user confirms
13  543 --                 }
13  544 --        SIPChoice =     (sipNever, sipAlways, sipAskUser);
13  545 --
13  546 --
13  547 --      { O T H E R }
13  548 --        PAppFile = ^AppFile;    { AppFile is defined in the Std File (Package
13  549 --                                  Manager) section of Inside Macintosh }
13  550 --
13  551 --        HTypeList   = ^PTypeList;
13  552 --        PTypeList   = ^ArrTypeList;
13  553 --        ArrTypeList = ARRAY[1..8000] OF OSType;
13  554 --
13  555 --
13  556 --
13  557 --      { C L A S S E S }
13  558 --
13  559 --        { The TEvtHandler type represents abstract objects that handle certain kinds
13  560 --            of events:
13  561 --                Key events:     both key down and auto key events
13  562 --                menu events:    both enabling menus and menu items, and processing
13  563 --                                menu commands
13  564 --                Read to/write from Disk
13  565 --                Idle events:    when there are no other events to handle
13  566 --                Actual events:  EvtHandlers, through their method 'DoHandleEvent',
13  567 --                                may intercept and handle actual ToolBox events
13  568 --                Termination:    when the application quits
13  569 --
13  570 --         TEvtHandler are linked into a list with the most specific object (usually
13  571 --            a selection) at the head of the list.  The global variable 'gTarget'
13  572 --            contains the head of the list.  (When a window is deactivated, this
13  573 --            global variable is cached in the window object, and retrieved when it
13  574 --            is later activated.)
13  575 --
13  576 --         For these kinds of events, the target (gTarget) gets the first crack at
13  577 --            handling the event.  The default implementation of the methods of
13  578 --            TEvtHandler is to pass the event to the next element of the list. }
13  579 --
13  580 --        {-----------------------------------+
13  581 --        |   TEvtHandler                     |
13  582 --        +-----------------------------------}
13  583 --        TEvtHandler = OBJECT(TObject)
13  584 --
13  585 --            fNextHandler:   TEvtHandler;    { the next element of the list, or NIL }
13  586 --            fIdleFreq:      LONGINT;        { the minimum number of ticks that can
13  587 --                                              pass before my DoIdle is called.
13  588 --                                              0=never call my DoIdle. }
13  589 --            fLastIdle:      LONGINT;        { the tick count the last time my DoIdle
13  590 --                                              was called. }
13  591 --
13  592 --          { Init & Free }
13  593 --            PROCEDURE TEvtHandler.IEvtHandler(itsNextHandler: TEvtHandler);
```

```
13  594 --              PROCEDURE TEvtHandler.Free; OVERRIDE;
13  595 --
13  596 --            { Termination }
13  597 --              PROCEDURE TEvtHandler.Terminate;
13  598 --
13  599 --          {$IFC qDebug}
13  600 --            { Debugging }
13  601 --              PROCEDURE TEvtHandler.IdentifySoftware;
13  602 --              FUNCTION  TEvtHandler.LookupSymbol(VAR sym: String8): LONGINT;
13  603 --                  { Called by the debugger to translate a symbol into a number;
13  604 --                      default passes to the fNextHandler.  sym will be uppercased.
13  605 --                      Return of -1 means symbol was not found.  If sym = '?' then
13  606 --                      writeln a list of available symbols. }
13  607 --              PROCEDURE TEvtHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
13  608 --                                                                 fieldAddr: Ptr;
13  609 --                                                                 fieldType: INTEGER)); OVERRIDE;
13  610 --          {$ENDC}
13  611 --
13  612 --            { Events }
13  613 --              PROCEDURE TEvtHandler.DoIdle(phase: IdlePhase);
13  614 --
13  615 --              FUNCTION  TEvtHandler.DoHandleEvent(nextEvent: PEventRecord;
13  616 --                      VAR commandToPerform: TCommand): BOOLEAN;
13  617 --                  { I Handle the event and return TRUE if I want it, else I don't
13  618 --                      handle it, and return FALSE }
13  619 --
13  620 --
13  621 --            { Double/Triple Clicks }
13  622 --              FUNCTION  TEvtHandler.DoMultiClick(lastDownPt, newDownPt: Point): BOOLEAN;
13  623 --                  { Called by TApplication.CountClicks.  Should return TRUE if the 2
13  624 --                      points are close enough to be considered part of a double/triple
13  625 --                      click.  (Both points are in global coordinates.)  This is only
13  626 --                      called if the mouse down was within the proper time range of the
13  627 --                      previous mouse up.  Default is to pass message to the
13  628 --                      nextEventHandler (if one exists), otherise to require that the
13  629 --                      sum of the x & y distances is <= 5.  ??? choose a default ??? }
13  630 --
13  631 --
13  632 --            { Key Events }
13  633 --              FUNCTION  TEvtHandler.DoKeyCommand(ch: Char; aKeyCode: INTEGER;
13  634 --                      VAR info: EventInfo): TCommand;
13  635 --                  { Handles all keystrokes except those with the command key held down. }
13  636 --
13  637 --              FUNCTION  TEvtHandler.DoCommandKey (ch: CHAR; VAR info: EventInfo): TCommand;
13  638 --                  { Handles command-key combinations only. }
13  639 --
13  640 --
13  641 --            { MenuEvents }
13  642 --              FUNCTION  TEvtHandler.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
13  643 --                  { Returns NIL if command does not change the document (also perform
13  644 --                      the command), otherwise return a TCommand object. }
13  645 --
13  646 --              PROCEDURE TEvtHandler.DoSetupMenus;
13  647 --
13  648 --          {$IFC qTemplateViews}
13  649 --            { View Creation from Templates}
13  650 --              FUNCTION  TEvtHandler.DoCreateViews (itsDocument: TDocument; parentView: TView; itsRsrcID: INTEGER): TView;
13  651 --
13  652 --              FUNCTION  TEvtHandler.CreateAView (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr): TView;
13  653 --          {$ENDC}
13  654 --
13  655 --            { Miscellaneous }
13  656 --              FUNCTION  TEvtHandler.HandlesPrintingCommands: BOOLEAN;
13  657 --
13  658 --              PROCEDURE TEvtHandler.InstallSelection(wasActive, beActive: BOOLEAN);
13  659 --                  { The Target is told to do this after a window is activated or
13  660 --                      deactivated }
13  661 --
13  662 --              END;
13  663 --
13  664 --
13  665 --              (*  The purpose of TApplication is to implement the main event loop that
13  666 --                  all Macintosh applications must have.   There is only one instance of
13  667 --                  TApplication.  Your main program should look similar to the following:
13  668 --
13  669 --                          InitToolbox(_, _);     { essential Toolbox initialization &
13  670 --                                                   other one-time initialization }
13  671 --                          NewObject(myApplication);
13  672 --                          myApplication.Run;
13  673 --
13  674 --                One thing that TApplication does is to interpret the raw events that are
13  675 --                  posted and convert them into higher-level events.  For example, it
13  676 --                  converts a click in the menu bar into a 'menu event' that contains the
13  677 --                  menu ID/item number that was chosen by the user. *)
13  678 --
13  679 --          {-----------------------------------+
13  680 --          |   TApplication                    |
13  681 --          +-----------------------------------}
13  682 --          TApplication = OBJECT(TEvtHandler)
13  683 --
13  684 --              { There are no TApplication variabes--they are global variables }
13  685 --
13  686 --            { Init & Free }
13  687 --              PROCEDURE TApplication.IApplication(itsMainFileType: OSType);
13  688 --                  { itsMainFileType should be the the 4-byte file type that this
13  689 --                      application reads/writes.  We require this here so that
13  690 --                      programmers do not have to explicitly override
13  691 --                      TApplication.SFGetParms and supply the type there.  That is the
13  692 --                      only place where the main file type is used. }
```

```
13  693 --
13  694 --          {$IFC qDebug}
13  695 --              PROCEDURE TApplication.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13  696 --
13  697 --              PROCEDURE TApplication.Fields (PROCEDURE DoToField (fieldName: Str255;
13  698 --                                                                  fieldAddr: Ptr;
13  699 --                                                                  fieldType: INTEGER)); OVERRIDE;
13  700 --          {$ENDC}
13  701 --
13  702 --            { Event Handling }
13  703 --              FUNCTION  TApplication.GetEvent(eventMask: INTEGER; sleep: LONGINT; cursorRgn: RgnHandle;
13  704 --                                              VAR anEvent: EventRecord): BOOLEAN;
13  705 --                  { Calls WaitNextEvent (or GetNextEvent) with the given event mask, sleep, and cursor
13  706 --                    region parameters.  You can override this if you have an alternate source for events. }
13  707 --
13  708 --              FUNCTION TApplication.HandleAlienEvent(VAR theEventInfo: EventInfo): TCommand;
13  709 --                  { This method is called upon to deal with network events and
13  710 --                    'application-specific' events }
13  711 --
13  712 --              PROCEDURE TApplication.MainEventLoop;
13  713 --
13  714 --              PROCEDURE TApplication.DispatchEvent(VAR theEventInfo: EventInfo; VAR commandToPerform: TCommand);
13  715 --
13  716 --              PROCEDURE TApplication.HandleEvent (VAR theEvent: EventRecord);
13  717 --                  { When this is called by MacApp, theEvent is a pointer to the global
13  718 --                    variable gLastEvent.  In special circumstances, you might
13  719 --                    'fake' an event record and pass a pointer to it. }
13  720 --
13  721 --              PROCEDURE TApplication.PostHandleEvent (VAR theEventInfo: EventInfo);
13  722 --
13  723 --              FUNCTION TApplication.HandleActivateEvent (VAR theEventInfo: EventInfo): TCommand;
13  724 --                  { Handles activate (and deactivate) events.  Calls the
13  725 --                    window's Activate method, and sets gRedrawMenuBar to true if
13  726 --                    activating the window.  Returns gNoChanges. }
13  727 --
13  728 --              FUNCTION TApplication.HandleDiskEvent (VAR theEventInfo: EventInfo): TCommand;
13  729 --                  { Handles disk events.  Calls DIBadMount if the high word of
13  730 --                    theEvent's message is <> noErr.  Returns gNoChanges. }
13  731 --
13  732 --              FUNCTION TApplication.HandleKeyDownEvent (VAR theEventInfo: EventInfo): TCommand;
13  733 --                  { Handles key down events.  If the command key is down, then this
13  734 --                    calls CommandKey.  Otherwise it calls gTarget.DoKeyCommand.
13  735 --                    Returns the result of CommandKey or DoKeyCommand. }
13  736 --
13  737 --              FUNCTION  TApplication.HandleMouseDown(VAR theEventInfo: EventInfo): TCommand;
13  738 --                  { Called when a mouse down occurs; this cases on whereMouseDown
13  739 --                    and does the right thing. }
13  740 --
13  741 --              FUNCTION TApplication.TrackMouse (globalMouse, hysteresis: Point;
13  742 --                                                theCommand: TCommand): TCommand;
13  743 --
13  744 --              FUNCTION TApplication.HandleMouseUp (VAR theEventInfo: EventInfo): TCommand;
13  745 --                  { Sets gLastUpTime and sets gEventInfo.theClickCount to gClickCount.
13  746 --                    Returns gNoChanges. }
13  747 --
13  748 --              FUNCTION TApplication.HandleSystemEvent (VAR theEventInfo: EventInfo): TCommand;
13  749 --                  { Handles "system" events, in particular suspend/resume events and mouse
13  750 --                    moved events.  Calls RegainControl if switching in, AboutToLoseControl
13  751 --                    if switching out, or TrackCursor if it is a mouse-moved event.
13  752 --                    Returns gNoChanges. }
13  753 --
13  754 --              FUNCTION TApplication.HandleUpdateEvent (VAR theEventInfo: EventInfo): TCommand;
13  755 --                  { Handles window update events.  Calls UpdateEvent for the window
13  756 --                    indicate by the event, and returns gNoChanges. }
13  757 --
13  758 --              FUNCTION  TApplication.IsDeskAccessory (aWMgrWindow: WindowPtr): BOOLEAN;
13  759 --                  { Returns true if the given Window Manager window is a desk accessory.
13  760 --                    We assume that if the window has a negative windowKind then it
13  761 --                    is a desk accessory. }
13  762 --
13  763 --              PROCEDURE TApplication.OpenDeskAccessory (deskAccName: Str255);
13  764 --                  { Attempts to open the desk accessory whose name is deskAccName.
13  765 --                    Called from TApplication.MenuEvent when an Apple menu item
13  766 --                    is chosen, and the item has a negative command number. }
13  767 --
13  768 --              PROCEDURE TApplication.PollEvent;
13  769 --
13  770 --              PROCEDURE TApplication.Run;
13  771 --                  { HandleFinderRequest, and if not finder printing, call
13  772 --                    MainEventLoop }
13  773 --
13  774 --              PROCEDURE TApplication.UpdateAllWindows;
13  775 --                  { Process all update events in the queue. }
13  776 --
13  777 --              FUNCTION  TApplication.InModalState: BOOLEAN;
13  778 --                  { Returns true if the front window is modal. }
13  779 --
13  780 --              FUNCTION  TApplication.InModalMenuState: BOOLEAN;
13  781 --                  { Returns true if the front window does not allow menu commands. }
13  782 --
13  783 --          {$IFC qDebug}
13  784 --              PROCEDURE TApplication.ReportEvent(VAR theEventInfo: EventInfo);
13  785 --                  { Displays information about an event in the debugging window. }
13  786 --
13  787 --              PROCEDURE TApplication.ReportMouseDown(where: INTEGER);
13  788 --                  { Displays the location of the mouse in terms of a FindWindow result. }
13  789 --          {$ENDC}
13  790 --
13  791 --
```

```
13  792 --
13  793 --          { Target and Cohandlers }
13  794 --            PROCEDURE TApplication.InstallCohandler(aCohandler: TEvtHandler;
13  795 --                    addIt: BOOLEAN);
13  796 --                { Used to add (addIt = TRUE) or delete (addIt = FALSE) a cohandler
13  797 --                  to/from the global list of cohandlers }
13  798 --
13  799 --            PROCEDURE TApplication.EachHandler(aFirstHandler: TEvtHandler;
13  800 --                                PROCEDURE DoToEvtHandler(anEvtHandler: TEvtHandler));
13  801 --                { Performs 'DoToEvtHandler' to aFirstHandler, then to its fNextHandler,
13  802 --                  etc., onward until the fNextHandler chain ends at NIL }
13  803 --
13  804 --            FUNCTION TApplication.FirstHandlerThat(aFirstHandler: TEvtHandler;
13  805 --                        FUNCTION TestEvtHandler(anEvtHandler: TEvtHandler): BOOLEAN):
13  806 --                        TEvtHandler;
13  807 --                { Calls TestEvtHandler for each event handler until TestEvtHandler
13  808 --                  returns true.  Returns the TEvtHandler object for which
13  809 --                  TestEvtHandler returned true, or NIL if TestEvtHandler never
13  810 --                  returned true. }
13  811 --
13  812 --            PROCEDURE TApplication.SetTarget (newTarget: TEvtHandler);
13  813 --                { Called to set the current target for the application. }
13  814 --
13  815 --          { Finder Requests }
13  816 --            FUNCTION  TApplication.CanOpenDocument(itsCmdNumber: CmdNumber;
13  817 --                    VAR anAppFile: AppFile): BOOLEAN;
13  818 --                { Simulates the filtering done by Std File; this is only called when
13  819 --                  opening/printing documents from the finder; when using
13  820 --                  Std File, Std File does the filtering. }
13  821 --
13  822 --            PROCEDURE TApplication.HandleFinderRequest;
13  823 --                { Gets the info from the finder about what files to open/print and
13  824 --                  opens/prints them. }
13  825 --
13  826 --
13  827 --          { Opening / Printing Documents }
13  828 --            FUNCTION  TApplication.AlreadyOpen(fileName: Str255;
13  829 --                    volRefnum: INTEGER): TDocument;
13  830 --                { Given a name/volRefnum, if that document is already opened,
13  831 --                  returns the TDocument.  Otherwise returns NIL. }
13  832 --
13  833 --            FUNCTION  TApplication.ChooseDocument(itsCmdNumber: CmdNumber;
13  834 --                    VAR anAppFile: AppFile): BOOLEAN;
13  835 --                { Call this to make a Std File Get call; returns TRUE is
13  836 --                  user selected a file. }
13  837 --
13  838 --            FUNCTION  TApplication.DoMakeDocument(itsCmdNumber: CmdNumber): TDocument;
13  839 --                { Must be overridden.  Based on itsCmdNumber create a document
13  840 --                  object of the appropriate kind. }
13  841 --
13  842 --            FUNCTION  TApplication.KindOfDocument(itsCmdNumber: CmdNumber;
13  843 --                    itsPAppFile: PAppFile): CmdNumber;
13  844 --                { Given a cmd number and a specification of the file, return the cmd
13  845 --                  number to pass to DoMakeDocument that indicates the type of
13  846 --                  document to make;  Default is to return itsCmdNumber.  If you
13  847 --                  have multiple document types, a good convention is to return
13  848 --                  the cmd numbers assigned to create new documents of each kind.
13  849 --                  itsPAppFile will be NIL if we are creating a brand new document,
13  850 --                  rather than opening an existing document. }
13  851 --
13  852 --            PROCEDURE TApplication.OpenNew(itsCmdNumber: CmdNumber);
13  853 --                { Called when the application is opened (itsCmdNumber = cFinderNew),
13  854 --                  or when NEW is chosen from menu (itsCmdNumber = cNew).  If you
13  855 --                  do not want to create a new document when the user opens the
13  856 --                  application, override this and do nothing if itsCmdNumber =
13  857 --                  cFinderNew. }
13  858 --
13  859 --            PROCEDURE TApplication.OpenOld(itsOpenCmd: CmdNumber;
13  860 --                    anAppFile: AppFile);
13  861 --                { Called when opening an existing document from finder (itsCmdNumber
13  862 --                  = cFinderOpen) or if OPEN is chosen from menu (itsCmdNumber =
13  863 --                  cOpen). }
13  864 --
13  865 --            FUNCTION  TApplication.PrintDocument(anAppFile: AppFile): BOOLEAN;
13  866 --                { Called to print a document from the finder.  Returns TRUE if the
13  867 --                  user did not cancel printing of the rest of the documents. }
13  868 --
13  869 --            PROCEDURE TApplication.SFGetParms(itsCmdNumber: CmdNumber;
13  870 --                    VAR dlgID: INTEGER; VAR where: Point;
13  871 --                    VAR fileFilter, dlgHook, filterProc: ProcPtr;
13  872 --                    typeList: HTypeList);
13  873 --                { Called to return all the parameters that should be passed to
13  874 --                  SFGetFile;  when called, typeList is a valid handle to a
13  875 --                  0-length block.
13  876 --                Defaults to
13  877 --                  dlgID = getDlgID;
13  878 --                  where = (100, 100);
13  879 --                  fileFilter = NIL;
13  880 --                  dlgHook = NIL;
13  881 --                  filterProc = NIL;
13  882 --                  typeList = list of just the main file type supported by the
13  883 --                    application.  (If type list ends up with nothing in it,
13  884 --                    then all file types are supported. }
13  885 --
13  886 --          { Termination }
13  887 --            PROCEDURE TApplication.Close;
13  888 --                { Called when the user chooses Quit from the menu, and tries to save
13  889 --                  all the open documents.  If all succeed the application
13  890 --                  terminates.  Signals Failure with error = noErr and message =
```

```
13  891 --                      msgCancelled if user cancels }
13  892 --
13  893 --            { Hierarchy }
13  894 --              PROCEDURE TApplication.AddDocument(aNewDocument: TDocument);
13  895 --                  { Add another document to my list of documents, and make it the
13  896 --                      current one }
13  897 --              PROCEDURE TApplication.AddFreeWindow(aWindow: TWindow);
13  898 --                  { Add the window to the free-window list }
13  899 --              PROCEDURE TApplication.DeleteDocument(docToDelete: TDocument);
13  900 --                  { Delete a document from my list of documents, and adjust gDocument
13  901 --                      and activeFrame if needed }
13  902 --              PROCEDURE TApplication.DeleteFreeWindow(windowToDelete: TWindow);
13  903 --                  { Delete a window from my list of free windows }
13  904 --              PROCEDURE TApplication.EachFreeWindow(PROCEDURE DoToWindow(aWindow: TWindow));
13  905 --                  { Perform DoToWindow on each TWindow object in the list
13  906 --                      gFreeWindowList }
13  907 --              PROCEDURE TApplication.ForAllDocumentsDo(PROCEDURE DoToDoc(aDocument: TDocument));
13  908 --                  { Perform the given procedure on all open documents currently owned
13  909 --                      by the application }
13  910 --              PROCEDURE TApplication.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
13  911 --                  { Perform the given procedure on all windows of all documents, as
13  912 --                      well as on all free-standing (document-less) windows }
13  913 --
13  914 --
13  915 --            { Window Manager Windows }
13  916 --              PROCEDURE TApplication.SelectWMgrWindow (aWMgrWindow: WindowPtr);
13  917 --                  { "Selects" aWMgrWindow by calling the toolbox routine SelectWindow.  When any
13  918 --                      window is selected, it goes through here.  We never call SelectWindow
13  919 --                      directly.}
13  920 --
13  921 --              FUNCTION  TApplication.GetRsrcWindow(storage: Ptr; rsrcId: INTEGER;
13  922 --                                                   VAR isResizable, isClosable: BOOLEAN): WindowPtr;
13  923 --                  { Get a Window Manger window resource whose resource id is rsrId.
13  924 --                      Return isResizable and isClosable based upon the 'WIND'
13  925 --                      resource. }
13  926 --
13  927 --              FUNCTION TApplication.WMgrToWindow (aWMgrWindow: WindowPtr): TWindow;
13  928 --                  { Returns the window object that represents the given Window Manager
13  929 --                      window, or NIL if there is no window object. }
13  930 --
13  931 --
13  932 --            { Idle Events }
13  933 --              PROCEDURE TApplication.Idle(phase: IdlePhase);
13  934 --
13  935 --
13  936 --            { Cursor }
13  937 --              FUNCTION  TApplication.TrackCursor: BOOLEAN;
13  938 --                  { Returns true if the cursor is set by a view.  Otherwise the cursor is set
13  939 --                      to an arrow and TrackCursor returns false.}
13  940 --
13  941 --
13  942 --            { Menu Events }
13  943 --              FUNCTION  TApplication.DoCommandKey(ch: CHAR; VAR info: EventInfo): TCommand; OVERRIDE;
13  944 --                  { Called when a keyDown event is received and the command key is down. }
13  945 --
13  946 --              FUNCTION  TApplication.MenuEvent(menuItem: LONGINT): TCommand;
13  947 --                  { Given a value returned by MenuKey or MenuSelect, figure out the
13  948 --                      command number that was chosen, have the application create a
13  949 --                      command object, and return it. }
13  950 --
13  951 --              PROCEDURE TApplication.SetUndoText(cmdDone: BOOLEAN;
13  952 --                      aCmdNumber: CmdNumber);
13  953 --                  { Called to setup the Undo menu item; if cmdDone is TRUE it reads
13  954 --                      Undo, otherwise it reads Redo.  aCmdNumber indicates the
13  955 --                      command that goes after the Undo/Redo. }
13  956 --
13  957 --              PROCEDURE TApplication.SetupTheMenus;
13  958 --                  { Initiates the process of enabling & checking menu items. }
13  959 --
13  960 --
13  961 --            { Opening and Closing Windows }
13  962 --              PROCEDURE TApplication.CloseWmgrWindow(aWmgrWindow: WindowPtr);
13  963 --                  { Called when user closes a window either with a menu item or with
13  964 --                      a GoAway box.  Signals Failure with error = noErr and message =
13  965 --                      msgCancelled if user cancels }
13  966 --
13  967 --
13  968 --            { Command Management }
13  969 --              PROCEDURE TApplication.CommitLastCommand;
13  970 --
13  971 --              PROCEDURE TApplication.PerformCommand(command: TCommand);
13  972 --                  { This DOES NOT check to see if command is gNoChanges. }
13  973 --
13  974 --
13  975 --            { Double/Triple Clicks }
13  976 --              FUNCTION  TApplication.CountClicks(aPDownEvent:     PEventRecord;
13  977 --                                                 whereMouseDown:  INTEGER): INTEGER;
13  978 --                  { Called by TApplication.HandleMouseDown.  Returns the number of
13  979 --                      clicks that can be considered multiple clicks  (e.g. if it
13  980 --                      returns 1 the mouse down should be treated as a single click, or
13  981 --                      possibly the first click of a multi-click sequence.  If it returns
13  982 --                      2 the click should be considered a double-click, etc.  A
13  983 --                      click is considered part of a multi-click sequence if the
13  984 --                      mouse down was within the proper time range of the previous
13  985 --                      mouse up, and was within the proper number of pixels of the last
13  986 --                      mouse down.  TApplication.HandleMouseDown sets theClickCount
13  987 --                      of its EventInfo record to the result of this function.
13  988 --
13  989 --                      aPDownEvent is a pointer to the mouse down event.
```

```
13  990 --                    whereMouseDown is the result of FindWindow on the event. }
13  991 --
13  992 --
13  993 --          { Command Handlers }
13  994 --            FUNCTION  TApplication.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
13  995 --                    OVERRIDE;
13  996 --
13  997 --            PROCEDURE TApplication.DoSetupMenus; OVERRIDE;
13  998 --
13  999 --
13 1000 --          { Clipboard stuff }  { NB: Perhaps the categorization of TApp methods needs
13 1001 --                                 redoing }
13 1002 --            PROCEDURE TApplication.AbandonUndoClipboard;
13 1003 --                { Called when the undo clipboard is no longer needed.  Calls
13 1004 --                  gClipUndoView.FreeFromClipboard and sets gClipUndoView to NIL. }
13 1005 --
13 1006 --            PROCEDURE TApplication.AboutToLoseControl(convertClipboard: BOOLEAN);
13 1007 --                { Called when about to activate a Desk Accessory, switch out to
13 1008 --                  another application, or Terminate.  convertClipboard is generally
13 1009 --                  true, except it will be FALSE if we get a Swticher event that does
13 1010 --                  not require Clipboard conversion }
13 1011 --
13 1012 --            PROCEDURE TApplication.AbsorbScrapStuff;
13 1013 --                { Called to retrieve the current InfoScrap record from low memory.
13 1014 --                  This will be used to determine if the scrap has changed. }
13 1015 --
13 1016 --            PROCEDURE TApplication.CheckDeskScrap;
13 1017 --                { Checks to see if the desk scrap has changed by calling
13 1018 --                  AbsorbScrapStuff and comparing the current scrap change count
13 1019 --                  with the previous change count.  If the scrap has changed, then
13 1020 --                  the current clipboard becomes the undo clipboard and
13 1021 --                  ReadFromDeskScrap is called to read in the new desk scrap. }
13 1022 --
13 1023 --            PROCEDURE TApplication.ClaimClipboard(clipView: TView);
13 1024 --                { Makes clipView the new view displayed by the Clipboard.  Typically
13 1025 --                  called by a Cut or Copy command. }
13 1026 --
13 1027 --            FUNCTION  TApplication.GetDataToPaste(aDataHandle: Handle;
13 1028 --                    VAR dataType: ResType): LONGINT;
13 1029 --                { Returns the number of bytes in the paste data.  Will signal
13 1030 --                  Failure if it gets an error. }
13 1031 --
13 1032 --            PROCEDURE TApplication.LaunchClipboard;
13 1033 --                { Called by TApplication.Run, before the main event loop. }
13 1034 --
13 1035 --            FUNCTION TApplication.MakeClipboardWindow: TWindow;
13 1036 --                { Called by TApplication.IApplication to make the clipboard window
13 1037 --                  and associated frame(s).  clipView is the view to be initially
13 1038 --                  installed in the clipboard--it is gClipOrphanage. }
13 1039 --
13 1040 --            PROCEDURE TApplication.ReadFromDeskScrap;
13 1041 --                { Called at launch and when the desk scrap changes.  Calls
13 1042 --                  TApplication.MakeViewForAlienClipboard, then ClaimClipboard }
13 1043 --
13 1044 --            FUNCTION  TApplication.MakeViewForAlienClipboard: TView;
13 1045 --                { Returns gClipOrphanage, which is capable of displaying scrap
13 1046 --                  of type TEXT or PICT.  Override this to create a view to
13 1047 --                  handle other types of scrap data. }
13 1048 --
13 1049 --            PROCEDURE TApplication.RegainControl(checkClipboard: BOOLEAN);
13 1050 --                { Called when switching in or when leaving a desk accessory }
13 1051 --
13 1052 --            PROCEDURE TApplication.SetClipView(clipView: TView);
13 1053 --                { Installs the given view into the clipboard window. }
13 1054 --
13 1055 --            PROCEDURE TApplication.SwapClipViews;
13 1056 --                { Swaps the undo clipboard view with the current clipboard view.
13 1057 --                  Used during an Undo or Redo. }
13 1058 --
13 1059 --        {$IFC qDebug}
13 1060 --          { Debugging }
13 1061 --          PROCEDURE TApplication.IdentifySoftware; OVERRIDE;
13 1062 --        {$ENDC}
13 1063 --
13 1064 --          { Error Alerts }
13 1065 --          PROCEDURE TApplication.ShowError(error: OSerr; message: LONGINT);
13 1066 --          PROCEDURE TApplication.SpaceIsLow;
13 1067 --              { Called when space is low, at intervals of kLowSpaceInterval
13 1068 --                ticks.  Displays an alerting informing the user that
13 1069 --                memory space is low.  Override this if you wish to take
13 1070 --                more appropriate action. }
13 1071 --            END;
13 1072 --
13 1073 --
13 1074 --        {-----------------------------------+
13 1075 --        |   TDocument                       |
13 1076 --        +-----------------------------------}
13 1077 --        TDocument = OBJECT(TEvtHandler)
13 1078 --
13 1079 --            fWindowList:        TList;          { list of windows belonging to this
13 1080 --                                                    document }
13 1081 --            fViewList:          TList;          { list of views belonging to document }
13 1082 --            fDocPrintHandler:   TPrintHandler;  { the object to be told to PRINT or
13 1083 --                                                  DoSetUpMenus when 'Print',
13 1084 --                                                  'Print One', or 'Page Setup…'
13 1085 --                                                  is selected from the menu while
13 1086 --                                                  this is the active document. }
13 1087 --
13 1088 --            fChangeCount:       LONGINT;        { master count of changes since last
```

```
13 1089 --                                                  Save }
13 1090 --          fSavePrintInfo:     BOOLEAN;        { if TRUE for a document saved on
13 1091 --                                                  disk, the 'print info' record
13 1092 --                                                  of the fDocPrintHandler will be
13 1093 --                                                  written out to the data fork
13 1094 --                                                  before other writing takes
13 1095 --                                                  place }
13 1096 --          fSharePrintInfo:    BOOLEAN;        { if TRUE, then all printHandlers
13 1097 --                                                  associated with views belonging
13 1098 --                                                  to the same document will share
13 1099 --                                                  the same 'print info' record }
13 1100 --          fPrintInfo:         Handle;         { if non-NIL, this is a handle to a
13 1101 --                                                  120-byte 'print info' record }
13 1102 --
13 1103 --          fTitle:             STRING[63];     { file name }
13 1104 --          fFileType:          OSType;         { file type }
13 1105 --          fCreator:           OSType;         { creator ID }
13 1106 --          fVolRefNum:         INTEGER;        { volume refNum }
13 1107 --          fModDate:           LONGINT;        { file mod date when last read/saved }
13 1108 --          fReopenAlert:       BOOLEAN;        { whether to give an alert if user
13 1109 --                                                  reopens doc }
13 1110 --          fSaveExists:        BOOLEAN;        { whether a disk file representing
13 1111 --                                                  this document exists }
13 1112 --          fCommitOnSave:      BOOLEAN;        { commit the last command before
13 1113 --                                              saving the document, if it affected the
13 1114 --                                              document.  This defaults to TRUE, but
13 1115 --                                              carefully written applications could set
13 1116 --                                              this FALSE. }
13 1117 --          fUsesDataFork:      BOOLEAN;        { whether the document uses the data
13 1118 --                                                  fork of the file }
13 1119 --          fUsesRsrcFork:      BOOLEAN;        { whether the document uses the rsrc
13 1120 --                                                  fork of the file }
13 1121 --          fDataOpen:          BOOLEAN;        { whether the data fork should be
13 1122 --                                                  open all the time }
13 1123 --          fRsrcOpen:          BOOLEAN;        { whether the resources fork be open
13 1124 --                                                  all the time }
13 1125 --          fDataPerm:          INTEGER;        { permission to use for reading data
13 1126 --                                                  fork }
13 1127 --          fRsrcPerm:          INTEGER;        { permission to use for reading the
13 1128 --                                                  rsrc fork }
13 1129 --          fDataRefnum:        INTEGER;        { refnum; valid only if data fork is
13 1130 --                                                  left open }
13 1131 --          fRsrcRefnum:        INTEGER;        { refnum; valid only if resource
13 1132 --                                                  fork is left open }
13 1133 --                                              { these will be -1 if the fork is
13 1134 --                                                  not open or if the corresponding
13 1135 --                                                  fXXXOpen flag is FALSE }
13 1136 --          fSaveInPlace:       SIPChoice;      { how to save files in place (if
13 1137 --                                                  desired) }
13 1138 --
13 1139 --        { Init & Free }
13 1140 --          PROCEDURE TDocument.IDocument(itsFileType, itsCreator: OSType;
13 1141 --                  usesDataFork, usesRsrcFork: BOOLEAN;
13 1142 --                  keepsDataOpen, keepsRsrcOpen: BOOLEAN);
13 1143 --          PROCEDURE TDocument.Free; OVERRIDE;
13 1144 --              { This does not call FreeData by default, since you may need to
13 1145 --                  control the order in which things are freed.    Your override
13 1146 --                  of TDocument.Free can call FreeData if convenient. }
13 1147 --
13 1148 --              PROCEDURE TDocument.FreeData;
13 1149 --                  { Called when a document is reverted, in order to free it's data
13 1150 --                      objects.  You may also wish to call this from you document's
13 1151 --                      Free method, if convenient. }
13 1152 --
13 1153 --              PROCEDURE TDocument.FreeFromClipboard;
13 1154 --                  { Called to free a Clipboard document.  Simply calls Free. }
13 1155 --
13 1156 --        {$IFC qDebug}
13 1157 --              PROCEDURE TDocument.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13 1158 --
13 1159 --              PROCEDURE TDocument.Fields (PROCEDURE DoToField (fieldName: Str255;
13 1160 --                                                               fieldAddr: Ptr;
13 1161 --                                                               fieldType: INTEGER)); OVERRIDE;
13 1162 --        {$ENDC}
13 1163 --
13 1164 --        { Opening / Printing Documents }
13 1165 --          PROCEDURE TDocument.DoInitialState;
13 1166 --              { Called for 'New', 'Revert' to blank, Open Tool. Default is to do
13 1167 --                  nothing;  You may need to do further initialization that you
13 1168 --                  would not do in case of an Open... }
13 1169 --
13 1170 --              PROCEDURE TDocument.DoMakeWindows;
13 1171 --                  { Should take the views created by DoMakeViews and create windows
13 1172 --                      and frames to display them. }
13 1173 --
13 1174 --              PROCEDURE TDocument.DoMakeViews(forPrinting: BOOLEAN);
13 1175 --                   { Create all necessary views for this document (based on
13 1176 --                      forPrinting flag) and store into fields of your document.  The
13 1177 --                      document will then be sent either DoMakeWindows or Print. }
13 1178 --
13 1179 --              PROCEDURE TDocument.DoRead(aRefNum: INTEGER;
13 1180 --                      rsrcExists, forPrinting: BOOLEAN);
13 1181 --              PROCEDURE TDocument.DoWrite(aRefNum: INTEGER; makingCopy: BOOLEAN);
13 1182 --                  { These just read/write the print state if necessary.
13 1183 --                      rsrcExists indicates whether the document has a resource
13 1184 --                          fork opened or not; it is FALSE if fUsesRsrcFork is FALSE.
13 1185 --                      makingCopy indicates whether this save is for making a
13 1186 --                          copy of the document (vs. a regular Save or Save As...);
13 1187 --                          disk-based documents (especially) might optimize their
```

```
13 1188 --                          behavior if this is TRUE. }
13 1189 --
13 1190 --              FUNCTION  TDocument.HandlesPrintingCommands: BOOLEAN; OVERRIDE;
13 1191 --
13 1192 --              FUNCTION TDocument.OpenAFile(name: Str255; volRefnum: INTEGER;
13 1193 --                      openData, openRsrc: BOOLEAN; dataPerm, rsrcPerm: INTEGER;
13 1194 --                      VAR dataRefnum, rsrcRefnum: INTEGER): OSErr;
13 1195 --                  { Called from ReadFromFile to open the document's data and/or
13 1196 --                      resource forks.}
13 1197 --
13 1198 --              PROCEDURE TDocument.OpenAgain(itsCmdNumber: CmdNumber;
13 1199 --                      openingDoc: TDocument);
13 1200 --                  { Called if the user tries to open the same document twice.
13 1201 --                      openingDoc will be the document object being opened.  Default
13 1202 --                      brings SELF to the front and calls Failure(noErr, 0), which
13 1203 --                      will abort opening the document.  (The goal is to prevent the
13 1204 --                      user from having 2 copies of the same document open, in which
13 1205 --                      case s/he could make incompatible changes to both.)
13 1206 --
13 1207 --                  You might override this if you want to open a second window on the
13 1208 --                      same document.  In the ideal case, you should make both windows
13 1209 --                      display exactly the same data, and changes made to one should
13 1210 --                      be reflected in the other.  You could also override this to
13 1211 --                      create a read-only copy of the document. }
13 1212 --
13 1213 --              PROCEDURE TDocument.ReadFromFile(VAR anAppFile: AppFile;
13 1214 --                      forPrinting: BOOLEAN);
13 1215 --                  { Called to read an existing file for display or printing.  If
13 1216 --                      anAppFile.fName = '' THEN we read from the file specified in
13 1217 --                      the instance variables instead of the parameter, and the
13 1218 --                      parameter will be updated to match the document. }
13 1219 --
13 1220 --              PROCEDURE TDocument.ShowWindows;
13 1221 --                  { Called when opening the document initially; default is to call
13 1222 --                      OpenWindow for all windows that have fFlags.openInitially TRUE }
13 1223 --
13 1224 --              PROCEDURE TDocument.UntitledName(VAR noName: Str255);
13 1225 --                  { Called to supply a name for an untitled document.  If this returns
13 1226 --                      '' then the windows are not renamed from the value specified
13 1227 --                      in their titles. }
13 1228 --
13 1229 --
13 1230 --            { Saving Documents }
13 1231 --              PROCEDURE TDocument.AboutToSave(itsCmd: CmdNumber; VAR newName: Str255;
13 1232 --                      VAR newVolRefnum: INTEGER; VAR makingCopy: BOOLEAN);
13 1233 --                  { Called right after the SFPutFile (if any) and before deciding
13 1234 --                      between SaveViaTemp & SaveInPlave.  Applications can use the
13 1235 --                      name in an alert and/or modify the makingCopy parameter if they
13 1236 --                      do not want to have the document renamed. }
13 1237 --
13 1238 --              PROCEDURE TDocument.DoNeedDiskSpace(VAR dataForkBytes,
13 1239 --                      rsrcForkBytes: LONGINT);
13 1240 --                  { Bytes required to store SAVE file(s) on disk. When called, both
13 1241 --                      parameters are set to 0 for you. }
13 1242 --
13 1243 --              PROCEDURE TDocument.FreeFile;
13 1244 --                  { Called to free in memory data associated with the currently opened
13 1245 --                      document.  The default is to call CloseFile(fDataRefnum,
13 1246 --                      fRsrcRefnum) if fDataOpen OR fRsrcOpen. }
13 1247 --
13 1248 --              FUNCTION  TDocument.GetSaveInfo(itsCmdNumber: CmdNumber;
13 1249 --                      copyFInfo: BOOLEAN; VAR cInfo: CInfoPBRec): BOOLEAN;
13 1250 --                  { Called to get the desired file info for saving the file.  Default
13 1251 --                      copies the info from the current file if fSaveExists AND
13 1252 --                      copyFInfo, otherwise it just sets the file type and creator
13 1253 --                      based on the fields of SELF.  If an error occurs calling the
13 1254 --                      file system, the cInfo record will not be filled in. Returns
13 1255 --                      TRUE if it successfully gets the info from the file. }
13 1256 --
13 1257 --              PROCEDURE TDocument.GetTempName(VAR filename: Str255);
13 1258 --                  { Return a temporary name for saving the file.  The filename is
13 1259 --                      constructed in two parts:  the first part is the document's
13 1260 --                      name, or the application's name if the document is untitled.
13 1261 --                      The second part, appended to the first part, is a pseudo-
13 1262 --                      random number based on the tick count and number of seconds
13 1263 --                      since boot. }
13 1264 --
13 1265 --              PROCEDURE TDocument.MakeNewCopy(makingCopy: BOOLEAN;
13 1266 --                      validFInfo: BOOLEAN; VAR cInfo: CInfoPBRec);
13 1267 --                  { Creates a brand new copy of the document in the file specified by
13 1268 --                      cInfo.  The name, volume refnum, file type, and creator must
13 1269 --                      be specified in cInfo.  The other info fields of cInfo are used
13 1270 --                      only if makingCopy is FALSE.
13 1271 --
13 1272 --                      makingCopy is TRUE if this save is making a copy of the
13 1273 --                          document (vs. a regular Save or Save As...)
13 1274 --
13 1275 --                      Caller is responsible for calling FlushVol after this. }
13 1276 --
13 1277 --              FUNCTION TDocument.PoseSaveDialog: INTEGER;
13 1278 --                  { If the document has been changed (fChangeCount > 0), then the
13 1279 --                      user is asked to save the document.  PoseSaveDialog returns
13 1280 --                      cancel, kYesButton or kNoButton, depending on the button
13 1281 --                      chosen by the user.  If the document has not been changed then
13 1282 --                      kNoButton is returned. }
13 1283 --
13 1284 --              PROCEDURE TDocument.RequestFileName (itsCmdNumber:  CmdNumber;
13 1285 --                                                   makingCopy:    BOOLEAN;
13 1286 --                                                   VAR filename:  Str255;
```

```
13 1287 --                                                  VAR volRefNum: INTEGER);
13 1288 --                  { Asks the user for a filename for the document.  Calls
13 1289 --                      TDocument.SFPutParms and then SFPPutFile.  Fails if the user
13 1290 --                      clicks the Cancel button to end the put file dialog. }
13 1291 --
13 1292 --              PROCEDURE TDocument.Save(itsCmdNumber: CmdNumber;
13 1293 --                      askForFilename, makingCopy: BOOLEAN);
13 1294 --                  { Try to save the document to disk;  If askForFilename, put up a Std
13 1295 --                      File box;  If makingCopy, then this is for the Save a Copy In
13 1296 --                      command, so we do not rename the document.
13 1297 --
13 1298 --                      This method calls FlushVol (which takes care of the requirements
13 1299 --                      of MakeNewCopy, SaveInPlace, and SaveViaTemp. }
13 1300 --
13 1301 --              PROCEDURE TDocument.SaveAgain(itsCmdNumber: CmdNumber;
13 1302 --                      makingCopy: BOOLEAN; savingDoc: TDocument);
13 1303 --                  { Called when the user tries to Save As… or Save a Copy In…
13 1304 --                      and specifies the name of a document that is already opened.
13 1305 --                      The default case calls Failure(errSaveAgain, msgSaveFailed) to
13 1306 --                      display an error.  (The goal is to prevent the user from
13 1307 --                      having 2 copies of the same document open, in which case s/he
13 1308 --                      might make incompatible changes to both copies.)
13 1309 --
13 1310 --                      savingDoc is the document being saved }
13 1311 --
13 1312 --              PROCEDURE TDocument.SavedOn(VAR fileName: Str255; volRefnum: INTEGER);
13 1313 --                  { Called when a save successfully completes and we are switching
13 1314 --                      to the new file (ie., not a Save a Copy In…). }
13 1315 --
13 1316 --              PROCEDURE TDocument.SaveInPlace(itsCmdNumber: CmdNumber;
13 1317 --                      makingCopy, copyFInfo: BOOLEAN; VAR fileName: Str255;
13 1318 --                      volRefnum: INTEGER);
13 1319 --                  { Called to save a document in place.  Default is to delete the
13 1320 --                      existing file if NOT (fDataOpen OR fRsrcOpen).  If you have
13 1321 --                      disk based document, you have to decide whether to allow saving
13 1322 --                      in place.
13 1323 --
13 1324 --                      makingCopy is TRUE if this save is making a copy of the document
13 1325 --                          (vs. a regular Save or Save As…)
13 1326 --
13 1327 --                      copyFInfo is TRUE if the file information should be copied to the
13 1328 --                          new file
13 1329 --
13 1330 --                      Caller is responsible for calling FlushVol after this. }
13 1331 --
13 1332 --              PROCEDURE TDocument.SaveViaTemp(itsCmdNumber: CmdNumber;
13 1333 --                      makingCopy, copyFInfo: BOOLEAN; VAR fileName: Str255;
13 1334 --                      volRefnum: INTEGER);
13 1335 --                  { Called to save the document by creating a temporary copy of the
13 1336 --                      file and renaming it.  MacApp uses this method except when
13 1337 --                      there is not enough disk space available.
13 1338 --
13 1339 --                      makingCopy is TRUE if this save is making a copy of the document
13 1340 --                          (vs. a regular Save or Save As…)
13 1341 --
13 1342 --                      copyFInfo is TRUE if the file information should be copied to the
13 1343 --                          new file
13 1344 --
13 1345 --                      Caller is responsible for calling FlushVol after this. }
13 1346 --
13 1347 --              PROCEDURE TDocument.SFPutParms(itsCmdNumber: CmdNumber;
13 1348 --                      VAR dlgID: INTEGER; VAR where: Point;
13 1349 --                      VAR defaultName, prompt: Str255;
13 1350 --                      VAR dlgHook, filterProc: ProcPtr);
13 1351 --                  { Called to return all the parameters that should be passed to
13 1352 --                      SFPutFile.
13 1353 --                      Defaults to
13 1354 --                          dlgID = putDlgID;
13 1355 --                          where = (100, 100);
13 1356 --                          defaultName not changed;
13 1357 --                          prompt gotten from resource file;
13 1358 --                          dlgHook = NIL;
13 1359 --                          filterProc = NIL. }
13 1360 --
13 1361 --
13 1362 --          { Closing Documents }
13 1363 --            PROCEDURE TDocument.Close;
13 1364 --                  { Close a document and free it. Signals Failure with error = noErr
13 1365 --                      and message = msgCancelled if user cancels. }
13 1366 --                  { ??? NOTE: Must never be called for a document related to a view
13 1367 --                      in the Clipboard. }
13 1368 --
13 1369 --
13 1370 --
13 1371 --          { Revert }
13 1372 --            PROCEDURE TDocument.Revert;
13 1373 --            PROCEDURE TDocument.ShowReverted;
13 1374 --                  { Called after Revert to show brand new document; default tells each
13 1375 --                      view to ShowReverted }
13 1376 --
13 1377 --
13 1378 --          { Miscellaneous }
13 1379 --            PROCEDURE TDocument.CheckDiskFile(rsrcId, rsrcIndex: INTEGER;
13 1380 --                      reverting: BOOLEAN);
13 1381 --                  { Check to see if the disk file has changed.  If so put up alert
13 1382 --                      phFileChanged after setting ^0 to fTitle and ^1 to the string
13 1383 --                      specified by the rsrcId and rsrcIndex.  If user cancels, signal
13 1384 --                      Failure(noErr, msgCancelled).
13 1385 --
```

```
13 1386 --                  If reverting is TRUE, then I/O errors and nonmatching file types
13 1387 --                      will cause a Failure.
13 1388 --                  Otherwise, this is a Save and any I/O errors and nonmatching
13 1389 --                      file type will be ignored. }
13 1390 --
13 1391 --              FUNCTION  TDocument.DiskFileChanged(checkType: BOOLEAN): OSErr;
13 1392 --                  { Returns noErr if fSaveExists is FALSE or if the file's
13 1393 --                      modification date matches fModDate.  If checkType is TRUE then
13 1394 --                      returns errFTypeChanged if the file type has changed.  If the
13 1395 --                      file exists but with a different modification date, returns
13 1396 --                      errFileChanged. }
13 1397 --
13 1398 --              PROCEDURE TDocument.SetTitle(aTitle: Str255);
13 1399 --                  { Sets SELF.fTitle to aTitle and calls SetTitleForDoc for each
13 1400 --                      window of the document. }
13 1401 --                          .
13 1402 --
13 1403 --            { Command Handlers }
13 1404 --              FUNCTION  TDocument.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
13 1405 --              PROCEDURE TDocument.DoSetupMenus; OVERRIDE;
13 1406 --
13 1407 --
13 1408 --            { Hierarchy }
13 1409 --              PROCEDURE TDocument.AddView(aView: TView);
13 1410 --                  { Adds the view to the end of the document's view list. }
13 1411 --
13 1412 --              PROCEDURE TDocument.AddWindow(aWindow: TWindow);
13 1413 --                  { Adds the window to tne end of the document's window list. }
13 1414 --
13 1415 --              PROCEDURE TDocument.DeleteView(viewToDelete: TView);
13 1416 --                  { Deletes the view from the document's view list. }
13 1417 --
13 1418 --              PROCEDURE TDocument.DeleteWindow(windowToDelete: TWindow);
13 1419 --                  { Deletes the window from the document's window list. }
13 1420 --
13 1421 --              PROCEDURE TDocument.ForAllViewsDo(PROCEDURE DoToView(aView: TView));
13 1422 --                  { Performs DoToView on every view in every frame of every window
13 1423 --                      belonging to the document.  If you want to deal with all views
13 1424 --                      belonging to the document use fViewList.Each. }
13 1425 --
13 1426 --              PROCEDURE TDocument.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
13 1427 --                  { Perform the supplied proc to each window belonging to the document }
13 1428 --
13 1429 --              END;
13 1430 --
13 1431 --
13 1432 --          {-----------------------------------+
13 1433 --          |   TView                           |
13 1434 --          +-----------------------------------}
13 1435 --          ViewTemplatePtr = ^ViewTemplate;
13 1436 --          ViewTemplate =  PACKED RECORD
13 1437 --                              itsParentID:    IDType;
13 1438 --                              thisViewID:     IDType;
13 1439 --                              itsLocation:    VPoint;
13 1440 --                              itsSize:        VPoint;
13 1441 --                              itsVSizeDet:    Integer8;   { Coerced into a SizeDeterminer         }
13 1442 --                              itsHSizeDet:    Integer8;   { Coerced into a SizeDeterminer         }
13 1443 --                              isShown:        BOOLEAN;
13 1444 --                              isEnabled:      BOOLEAN;
13 1445 --                              filler:         Byte;
13 1446 --                              itsSignature:   IDType;     { Resource signature                    }
13 1447 --                              CASE INTEGER OF
13 1448 --                                  0: (itsType:        Str255);    { Actually variable length              }
13 1449 --                                  1: (includeRsrcID:  INTEGER);
13 1450 --                          END;
13 1451 --
13 1452 --          ViewRsrcHndl =  ^ViewRsrcPtr;
13 1453 --          ViewRsrcPtr =   ^ViewResource;
13 1454 --          ViewResource =  RECORD
13 1455 --                              numViews:       INTEGER;
13 1456 --                              theViews:       ViewTemplate;   { Actually, open-ended              }
13 1457 --                          END;
13 1458 --
13 1459 --          TView = OBJECT (TEvtHandler)
13 1460 --
13 1461 --              fSuperView:         TView;
13 1462 --              fSubViews:          TList;
13 1463 --              fDocument:          TDocument;
13 1464 --              fLocation:          VPoint;
13 1465 --              fSize:              VPoint;
13 1466 --              fSizeDeterminer:    ARRAY [VHSelect] OF SizeDeterminer;
13 1467 --              fHLDesired:         HLState;
13 1468 --              fIdentifier:        IDType;
13 1469 --              fShown:             BOOLEAN;
13 1470 --              fViewEnabled:       BOOLEAN;
13 1471 --              fPrintHandler:      TPrintHandler;
13 1472 --
13 1473 --            { Creation/Destruction Methods }
13 1474 --
13 1475 --              PROCEDURE TView.IView (itsDocument: TDocument;
13 1476 --                                     itsSuperview: TView;
13 1477 --                                     itsLocation: VPoint;
13 1478 --                                     itsSize: VPoint;
13 1479 --                                     itsHSizeDet,
13 1480 --                                     itsVSizeDet: SizeDeterminer);
13 1481 --
13 1482 --              PROCEDURE TView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
13 1483 --
13 1484 --              PROCEDURE TView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr);
```

```
13 1485 --
13 1486 --              PROCEDURE TView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr);
13 1487 --
13 1488 --              PROCEDURE TView.Free; OVERRIDE;
13 1489 --
13 1490 --              PROCEDURE TView.FreeFromClipboard;
13 1491 --
13 1492 --            { Coordinate Conversion Methods }
13 1493 --
13 1494 --              PROCEDURE TView.QDToViewPt (qdPoint: Point; VAR viewPt: VPoint);
13 1495 --
13 1496 --              PROCEDURE TView.QDToViewRect (qdRect: Rect; VAR viewRect: VRect);
13 1497 --
13 1498 --              FUNCTION  TView.ViewToQDPt (viewPt: VPoint): Point;
13 1499 --
13 1500 --              PROCEDURE TView.ViewToQDRect (viewRect: VRect; VAR qdRect: Rect);
13 1501 --
13 1502 --              PROCEDURE TView.LocalToSuper (VAR thePoint: VPoint);
13 1503 --                  { Given a point in local view coordinates, returns the same
13 1504 --                    point in its superview's coordinate system.}
13 1505 --
13 1506 --              PROCEDURE TView.SuperToLocal (VAR thePoint: VPoint);
13 1507 --                  { Given a point in superview coordinates, returns the same point
13 1508 --                    in the view's own coordinate system. }
13 1509 --
13 1510 --              PROCEDURE TView.LocalToWindow (VAR thePoint: VPoint);
13 1511 --                  { Given a point in local view coordinates, returns the same
13 1512 --                    point in its window's (or most super superview's) coordinate system.}
13 1513 --
13 1514 --              PROCEDURE TView.WindowToLocal (VAR thePoint: VPoint);
13 1515 --                  { Given a point in window (or most super superview) coordinates, returns
13 1516 --                    the same point in the view's own coordinate system. }
13 1517 --
13 1518 --            { Subview Management Methods }
13 1519 --
13 1520 --              PROCEDURE TView.EachSubview (PROCEDURE DoToSubview (theSubview: TView));
13 1521 --
13 1522 --              FUNCTION  TView.FirstSubviewThat (
13 1523 --                      FUNCTION TestSubview (theSubview: TView): BOOLEAN): TView;
13 1524 --
13 1525 --              FUNCTION  TView.LastSubViewThat (
13 1526 --                      FUNCTION TestSubView (theSubView: TView): BOOLEAN): TView;
13 1527 --
13 1528 --              PROCEDURE TView.AddSubview (theSubview: TView);
13 1529 --
13 1530 --              PROCEDURE TView.RemoveSubview (theSubview: TView);
13 1531 --
13 1532 --              PROCEDURE TView.MakeFirstSubView (theSubView: TView);
13 1533 --
13 1534 --              PROCEDURE TView.MakeLastSubView (theSubView: TView);
13 1535 --
13 1536 --              FUNCTION  TView.CountSubViews: INTEGER;
13 1537 --
13 1538 --              FUNCTION  TView.FindSubView (itsIdentifier: IDType): TView;
13 1539 --
13 1540 --            { Open/Close/Activate Methods }
13 1541 --
13 1542 --              PROCEDURE TView.Activate (entering: BOOLEAN);
13 1543 --
13 1544 --              PROCEDURE TView.Open;
13 1545 --
13 1546 --              PROCEDURE TView.Close;
13 1547 --
13 1548 --              PROCEDURE TView.BeInScroller (itsScroller: TScroller);
13 1549 --
13 1550 --              PROCEDURE TView.BeInPort (itsPort: GrafPtr);
13 1551 --
13 1552 --              PROCEDURE TView.ShowReverted;
13 1553 --
13 1554 --            { Choice Handling Methods }
13 1555 --
13 1556 --              PROCEDURE TView.DoChoice (origView: TView; itsChoice: INTEGER);
13 1557 --
13 1558 --            { Size Methods }
13 1559 --
13 1560 --              PROCEDURE TView.AdjustSize;
13 1561 --
13 1562 --              PROCEDURE TView.CalcMinSize (VAR minSize: VPoint);
13 1563 --
13 1564 --              PROCEDURE TView.ComputeSize (VAR newSize: VPoint);
13 1565 --
13 1566 --              PROCEDURE TView.Resize (width, height: VCoordinate; invalidate: BOOLEAN);
13 1567 --
13 1568 --              PROCEDURE TView.SuperViewChangedSize (delta: VPoint; invalidate: BOOLEAN);
13 1569 --
13 1570 --              PROCEDURE TView.SubViewChangedSize (theSubView: TView; delta: VPoint);
13 1571 --
13 1572 --            { Location Methods }
13 1573 --
13 1574 --              PROCEDURE TView.Locate (h, v: VCoordinate; invalidate: BOOLEAN);
13 1575 --
13 1576 --              PROCEDURE TView.SubViewMoved (theSubView: TView);
13 1577 --
13 1578 --              PROCEDURE TView.SuperViewMoved (invalidate: BOOLEAN);
13 1579 --
13 1580 --            { Focusing Methods }
13 1581 --
13 1582 --              FUNCTION  TView.Focus: BOOLEAN;
13 1583 --
```

```
13 1584 --              FUNCTION  TView.FocusOnSuperView: BOOLEAN;
13 1585 --
13 1586 --              PROCEDURE TView.ClipFurtherTo(r: Rect; hDeltaOrg, vDeltaOrg: INTEGER);
13 1587 --
13 1588 --            { Drawing Methods }
13 1589 --
13 1590 --              PROCEDURE TView.DrawContents;
13 1591 --
13 1592 --              PROCEDURE TView.Draw (area: Rect);
13 1593 --
13 1594 --              PROCEDURE TView.DoHighlightSelection (fromHL, toHL: HLState);
13 1595 --
13 1596 --              PROCEDURE TView.Adorn (area: Rect; itsPenSize: Point; itsAdornment: CntlAdornment);
13 1597 --
13 1598 --              PROCEDURE TView.Update;
13 1599 --
13 1600 --            { Validation/Invalidation Methods }
13 1601 --
13 1602 --              PROCEDURE TView.ForceRedraw;
13 1603 --
13 1604 --              PROCEDURE TView.InvalidVRect (viewRect: VRect);
13 1605 --
13 1606 --              PROCEDURE TView.ValidVRect (viewRect: VRect);
13 1607 --
13 1608 --              PROCEDURE TView.InvalidRect (r: Rect);
13 1609 --
13 1610 --              PROCEDURE TView.RevealRect (rectToReveal: VRect; minToSee: Point;
13 1611 --                                          redraw: BOOLEAN);
13 1612 --
13 1613 --              PROCEDURE TView.RevealTop (redraw: BOOLEAN);
13 1614 --
13 1615 --              PROCEDURE TView.RevealBottom (redraw: BOOLEAN);
13 1616 --
13 1617 --            { Mouse Handling Methods }
13 1618 --
13 1619 --              FUNCTION  TView.ContainsMouse (theMouse: VPoint): BOOLEAN;
13 1620 --
13 1621 --              FUNCTION  TView.HandleMouseDown (theMouse: VPoint; VAR info: EventInfo;
13 1622 --                                               VAR hysteresis: Point;
13 1623 --                                               VAR theCommand: TCommand): BOOLEAN;
13 1624 --
13 1625 --              FUNCTION  TView.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
13 1626 --                                              VAR hysteresis: Point): TCommand;
13 1627 --
13 1628 --            { Cursor Handling Methods }
13 1629 --
13 1630 --              FUNCTION  TView.HandleCursor (theMouse: VPoint; cursorRgn: RgnHandle): TView;
13 1631 --
13 1632 --              FUNCTION  TView.DoSetCursor (localPoint: Point;
13 1633 --                                           cursorRgn: RgnHandle): BOOLEAN;
13 1634 --
13 1635 --            { Miscellaneous Methods }
13 1636 --
13 1637 --              FUNCTION  TView.GetDialogView: TView; { actually, TDialogView_  }
13 1638 --
13 1639 --              FUNCTION  TView.GetGrafPort: GrafPtr;
13 1640 --
13 1641 --              FUNCTION  TView.GetScroller (immediateSuperView: BOOLEAN): TScroller;
13 1642 --
13 1643 --              FUNCTION  TView.GetWindow: TWindow;
13 1644 --
13 1645 --              PROCEDURE TView.GetExtent (VAR itsExtent: VRect);
13 1646 --
13 1647 --              PROCEDURE TView.GetQDExtent (VAR qdExtent: Rect);
13 1648 --
13 1649 --              PROCEDURE TView.GetFrame (VAR itsFrame: VRect);
13 1650 --
13 1651 --              PROCEDURE TView.GetVisibleRect (VAR visQDRect: Rect);
13 1652 --
13 1653 --              PROCEDURE TView.ViewEnable (state, redraw: BOOLEAN);
13 1654 --
13 1655 --              FUNCTION  TView.IsViewEnabled: BOOLEAN;
13 1656 --
13 1657 --              PROCEDURE TView.Show (state, redraw: BOOLEAN);
13 1658 --
13 1659 --              FUNCTION  TView.IsShown: BOOLEAN;
13 1660 --
13 1661 --            { Clipboard Handling }
13 1662 --
13 1663 --              FUNCTION  TView.ContainsClipType (aType: ResType): BOOLEAN;
13 1664 --
13 1665 --              FUNCTION  TView.GivePasteData (aDataHandle: Handle; dataType: ResType): LONGINT;
13 1666 --
13 1667 --              PROCEDURE TView.WriteToDeskScrap;
13 1668 --
13 1669 --            { PrintingMethods }
13 1670 --
13 1671 --              PROCEDURE TView.AttachPrintHandler(itsPrintHandler: TPrintHandler);
13 1672 --                  { Called by the PrintHandler during its initialization, NOT directly
13 1673 --                     by you }
13 1674 --
13 1675 --              FUNCTION  TView.DoBreakFollowing(vhs: VHSelect; prevBreak: VCoordinate;
13 1676 --                      VAR automatic: BOOLEAN): VCoordinate;
13 1677 --                  { Returns the location of the page break which follows the page break
13 1678 --                     located at 'prevBreak', in direction vhs; returns automatic =
13 1679 --                     TRUE if the page-break is thought of as an 'automatic' rather
13 1680 --                     than a 'manual' (user-specified) one.  Note that page-breaks in
13 1681 --                     dimension 'v' are drawn as vertical lines, those in dimension
13 1682 --                     'h' as horizontal lines }
```

```
13 1683 --
13 1684 --              PROCEDURE TView.DoCalcPageStrips (VAR pageStrips: Point);
13 1685 --                  { Computes the number of horizontal and vertical page strips. }
13 1686 --
13 1687 --              PROCEDURE TView.DoCalcViewPerPage (VAR viewPerPage: VPoint);
13 1688 --                  { Determine how much of the view normally goes into each printed
13 1689 --                      page, in each dimension. }
13 1690 --
13 1691 --              PROCEDURE TView.DoCheckPrinter;
13 1692 --                  { Check whether printer (if relevant) has changed, and if so, take
13 1693 --                      appropriate action }
13 1694 --
13 1695 --              PROCEDURE TView.DoDrawPrintFeedback(area: Rect);
13 1696 --                  { Draw page-breaks, page-numbers, view-borders, rulers, etc. }
13 1697 --
13 1698 --              PROCEDURE TView.DoDrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
13 1699 --                                               loc: VCoordinate; automatic: BOOLEAN);
13 1700 --                  { Draw one page break. }
13 1701 --
13 1702 --              FUNCTION  TView.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
13 1703 --                  { Enables my fPrintHandler to handle commands }
13 1704 --
13 1705 --              PROCEDURE TView.DoPagination;
13 1706 --                  { Recompute the dividing lines between areas of the view which will
13 1707 --                      be mapped into different printed pages }
13 1708 --
13 1709 --              PROCEDURE TView.DoPrinterChanged;
13 1710 --                  { The metrics relating to printer use have changed; absorb the
13 1711 --                      information and react }
13 1712 --
13 1713 --              PROCEDURE TView.DoSetPageOffset(coord: VPoint);
13 1714 --                  { Only reimplement in very unusual circumstances involving
13 1715 --                      Tall-UnAdjusted text printing etc. }
13 1716 --
13 1717 --              PROCEDURE TView.DoSetupMenus; OVERRIDE;
13 1718 --                  { tells my PrintHandler to set up its menus }
13 1719 --
13 1720 --              PROCEDURE TView.GetPrintExtent (VAR printExtent: VRect);
13 1721 --                  { Returns the part of the view that is to be printed.  The default is
13 1722 --                      to return the view's extent. }
13 1723 --
13 1724 --              PROCEDURE TView.PageInteriorChanged(newInterior: Rect);
13 1725 --                  { Allows a view which is printable to react to the changing of the
13 1726 --                      page-interior rectangle }
13 1727 --
13 1728 --            { Debugging Methods }
13 1729 --
13 1730 --          {$IFC qDebug}
13 1731 --              PROCEDURE TView.Fields (PROCEDURE DoToField (fieldName: Str255;
13 1732 --                                                           fieldAddr: Ptr;
13 1733 --                                                           fieldType: INTEGER)); OVERRIDE;
13 1734 --
13 1735 --              PROCEDURE TView.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13 1736 --          {$ENDC}
13 1737 --
13 1738 --              END { TView };
13 1739 --
13 1740 --          {-----------------------------------+
13 1741 --          |   TScroller                       |
13 1742 --          +-----------------------------------}
13 1743 --          ScrollerTemplatePtr =   ^ScrollerTemplate;
13 1744 --          ScrollerTemplate =  RECORD
13 1745 --                              wantVSBar:      BOOLEAN;
13 1746 --                              wantHSBar:      BOOLEAN;
13 1747 --                              vertMax:        LONGINT;
13 1748 --                              horzMax:        LONGINT;
13 1749 --                              vScrollUnits:   INTEGER;
13 1750 --                              hScrollUnits:   INTEGER;
13 1751 --                              vConstrain:     BOOLEAN;
13 1752 --                              hConstrain:     BOOLEAN;
13 1753 --                              sBarOffsets:    Rect;
13 1754 --                              END;
13 1755 --
13 1756 --          TScroller =     OBJECT (TView)
13 1757 --
13 1758 --              fTranslation:       VPoint;
13 1759 --              fScrollLimit:       VPoint;
13 1760 --              fMaxTranslation:    VPoint;
13 1761 --              fScrollBars:        ARRAY [VHSelect] OF TSScrollBar;
13 1762 --              fScrollUnit:        Point;
13 1763 --              fConstrain:         ARRAY [VHSelect] OF BOOLEAN;
13 1764 --              fSBarOffsets:       VRect;
13 1765 --
13 1766 --            { Creation/Destruction Methods }
13 1767 --
13 1768 --          {$IFC qProceduralViews}
13 1769 --              PROCEDURE TScroller.IScroller (itsSuperView: TView; itsLocation, itsSize: VPoint;
13 1770 --                                             itsHSizeDet, itsVSizeDet: SizeDeterminer;
13 1771 --                                             itsHorzMax, itsVertMax: VCoordinate;
13 1772 --                                             wantHorzSBar, wantVertSBar: BOOLEAN);
13 1773 --          {$ENDC}
13 1774 --
13 1775 --          {$IFC qTemplateViews}
13 1776 --              PROCEDURE TScroller.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 1777 --          {$ENDC}
13 1778 --
13 1779 --          {$IFC qWriteTemplates}
13 1780 --              PROCEDURE TScroller.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 1781 --
```

```
13 1782 --              PROCEDURE TScroller.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 1783 --          {$ENDC}
13 1784 --
13 1785 --              PROCEDURE TScroller.Free; OVERRIDE;
13 1786 --
13 1787 --            { Subview Management Methods }
13 1788 --
13 1789 --              PROCEDURE TScroller.AddSubView (theSubView: TView); OVERRIDE;
13 1790 --
13 1791 --              PROCEDURE TScroller.RemoveSubView (theSubView: TView); OVERRIDE;
13 1792 --
13 1793 --            { Focusing Methods }
13 1794 --
13 1795 --              FUNCTION TScroller.Focus: BOOLEAN; OVERRIDE;
13 1796 --
13 1797 --            { Miscellaneous Methods }
13 1798 --
13 1799 --              PROCEDURE TScroller.LocalToSuper (VAR thePoint: VPoint); OVERRIDE;
13 1800 --
13 1801 --              PROCEDURE TScroller.SuperToLocal (VAR thePoint: VPoint); OVERRIDE;
13 1802 --                  { Given a point in superview coordinates, returns the same point
13 1803 --                      in the view's own coordinate system. }
13 1804 --
13 1805 --              PROCEDURE TScroller.GetExtent (VAR itsExtent: VRect); OVERRIDE;
13 1806 --
13 1807 --            { Size Methods }
13 1808 --
13 1809 --              PROCEDURE TScroller.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 1810 --
13 1811 --            { Location Methods }
13 1812 --
13 1813 --              PROCEDURE TScroller.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 1814 --
13 1815 --              PROCEDURE TScroller.AdjustScrollBars;
13 1816 --
13 1817 --              PROCEDURE TScroller.SubViewChangedSize (theSubView: TView; delta: VPoint); OVERRIDE;
13 1818 --
13 1819 --              PROCEDURE TScroller.SetScrollLimits (scrollLimit: VPoint; drawScrollBars: BOOLEAN);
13 1820 --
13 1821 --              PROCEDURE TScroller.SetScrollParameters (horzUnits, vertUnits: VCoordinate;
13 1822 --                                                       horzConstraint, vertConstraint: BOOLEAN);
13 1823 --
13 1824 --            { Scrolling Methods }
13 1825 --
13 1826 --              PROCEDURE TScroller.CreateScrollBars (wantHorzSBar, wantVertSBar: BOOLEAN);
13 1827 --
13 1828 --              PROCEDURE TScroller.HaveScrollBar (theScrollBar: TSScrollBar;
13 1829 --                                                 direction: VHSelect);
13 1830 --
13 1831 --              PROCEDURE TScroller.ScrollDraw (delta: VPoint);
13 1832 --
13 1833 --              PROCEDURE TScroller.DoScroll (delta: VPoint; redraw: BOOLEAN);
13 1834 --
13 1835 --              FUNCTION  TScroller.ScrollStep (vhs: VHSelect; partCode: INTEGER): VCoordinate;
13 1836 --
13 1837 --              FUNCTION  TScroller.ScrollRelative (vhs: VHSelect; sBarValue: VCoordinate): VCoordinate;
13 1838 --
13 1839 --              PROCEDURE TScroller.ScrollBy (deltaH, deltaV: VCoordinate; redraw: BOOLEAN);
13 1840 --
13 1841 --              PROCEDURE TScroller.ScrollTo (h, v: VCoordinate; redraw: BOOLEAN);
13 1842 --
13 1843 --              PROCEDURE TScroller.AutoScroll (viewPt: VPoint; VAR delta: VPoint);
13 1844 --
13 1845 --              PROCEDURE TScroller.RevealRect (rectToReveal: VRect; minToSee: Point;
13 1846 --                                              redraw: BOOLEAN); OVERRIDE;
13 1847 --
13 1848 --            { Miscellaneous Methods }
13 1849 --
13 1850 --              FUNCTION  TScroller.GetScroller (immediateSuperView: BOOLEAN): TScroller; OVERRIDE;
13 1851 --
13 1852 --            { Debugging Methods }
13 1853 --
13 1854 --          {$IFC qDebug}
13 1855 --              PROCEDURE TScroller.Fields (PROCEDURE DoToField (fieldName: Str255;
13 1856 --                                                               fieldAddr: Ptr;
13 1857 --                                                               fieldType: INTEGER)); OVERRIDE;
13 1858 --          {$ENDC}
13 1859 --
13 1860 --              END { TScroller };
13 1861 --
13 1862 --          {-----------------------------------+
13 1863 --          |   TWindow                         |
13 1864 --          +-----------------------------------}
13 1865 --          WindowTemplatePtr = ^WindowTemplate;
13 1866 --          WindowTemplate =    PACKED RECORD
13 1867 --                                  procID:             INTEGER;
13 1868 --                                  hasGoAway:          BOOLEAN;
13 1869 --                                  resizable:          BOOLEAN;
13 1870 --                                  isModal:            BOOLEAN;
13 1871 --                                  doFirstClick:       BOOLEAN;
13 1872 --                                  freeOnClosing:      BOOLEAN;
13 1873 --                                  disposeOnFree:      BOOLEAN;
13 1874 --                                  closesDocument:     BOOLEAN;
13 1875 --                                  openInitially:      BOOLEAN;
13 1876 --                                  mustAdaptToScreen:  BOOLEAN;
13 1877 --                                  stagger:            BOOLEAN;
13 1878 --                                  mustForceOnScreen:  BOOLEAN;
13 1879 --                                  vertCenter:         BOOLEAN;
13 1880 --                                  horzCenter:         BOOLEAN;
```

```
13 1881 --                              filler:             0..7;
13 1882 --                              targetID:           IDType;
13 1883 --                              title:              Str255; {Actually variable length}
13 1884 --                          END;
13 1885 --
13 1886 --      WindowFlags =       PACKED RECORD
13 1887 --                              adapted:            BOOLEAN;
13 1888 --                              hCentered:          BOOLEAN;
13 1889 --                              vCentered:          BOOLEAN;
13 1890 --                              staggered:          BOOLEAN;
13 1891 --                              forced:             BOOLEAN;
13 1892 --                              isActive:           BOOLEAN;
13 1893 --                              isResizable:        BOOLEAN;
13 1894 --                              isClosable:         BOOLEAN;
13 1895 --                              freeOnClosing:      BOOLEAN;
13 1896 --                              disposeOnFree:      BOOLEAN;
13 1897 --                              closesDocument:     BOOLEAN;
13 1898 --                              openInitially:      BOOLEAN;
13 1899 --                              isModal:            BOOLEAN;
13 1900 --                              doFirstClick:       BOOLEAN;
13 1901 --                          END;
13 1902 --
13 1903 --      TWindow =       OBJECT (TView)
13 1904 --
13 1905 --          fWMgrWindow:        WindowPtr;
13 1906 --          fProcID:            INTEGER;
13 1907 --          fMoveBounds:        Rect;
13 1908 --          fResizeLimits:      Rect;
13 1909 --          fTarget:            TEvtHandler;
13 1910 --          fTargetID:          IDType;
13 1911 --          fPreDocname:        INTEGER;
13 1912 --          fConstTitle:        INTEGER;
13 1913 --          fIsActive:          BOOLEAN;
13 1914 --          fIsResizable:       BOOLEAN;
13 1915 --          fIsClosable:        BOOLEAN;
13 1916 --          fFreeOnClosing:     BOOLEAN;
13 1917 --          fDisposeOnFree:     BOOLEAN;
13 1918 --          fClosesDocument:    BOOLEAN;
13 1919 --          fOpenInitially:     BOOLEAN;
13 1920 --          fIsModal:           BOOLEAN;
13 1921 --          fDoFirstClick:      BOOLEAN;
13 1922 --          fAdapted:           BOOLEAN;
13 1923 --          fHorzCentered:      BOOLEAN;
13 1924 --          fVertCentered:      BOOLEAN;
13 1925 --          fStaggered:         BOOLEAN;
13 1926 --          fForcedOnScreen:    BOOLEAN;
13 1927 --
13 1928 --        { Creation/Destruction Methods }
13 1929 --
13 1930 --      {$IFC qProceduralViews}
13 1931 --          PROCEDURE TWindow.IWindow (itsDocument: TDocument; itsWmgrWindow: WindowPtr;
13 1932 --                                     canResize, canClose, disposeOnFree: BOOLEAN);
13 1933 --      {$ENDC}
13 1934 --
13 1935 --      {$IFC qTemplateViews}
13 1936 --          PROCEDURE TWindow.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 1937 --      {$ENDC}
13 1938 --
13 1939 --      {$IFC qWriteTemplates}
13 1940 --          PROCEDURE TWindow.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 1941 --
13 1942 --          PROCEDURE TWindow.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 1943 --      {$ENDC}
13 1944 --
13 1945 --          PROCEDURE TWindow.Free; OVERRIDE;
13 1946 --
13 1947 --        { Open/Close Methods }
13 1948 --
13 1949 --          PROCEDURE TWindow.Open; OVERRIDE;
13 1950 --
13 1951 --          PROCEDURE TWindow.Close; OVERRIDE;
13 1952 --
13 1953 --          PROCEDURE TWindow.CloseByUser;
13 1954 --
13 1955 --        { Activation Methods }
13 1956 --
13 1957 --          PROCEDURE TWindow.Activate (entering: BOOLEAN); OVERRIDE;
13 1958 --
13 1959 --        { Size/Location Methods }
13 1960 --
13 1961 --          PROCEDURE TWindow.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 1962 --
13 1963 --          PROCEDURE TWindow.MoveByUser (globalMouse: Point);
13 1964 --
13 1965 --          PROCEDURE TWindow.ResizeByUser (globalMouse: Point);
13 1966 --
13 1967 --          PROCEDURE TWindow.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 1968 --
13 1969 --          PROCEDURE TWindow.Zoom (partCode: INTEGER);
13 1970 --
13 1971 --        { Focusing Methods }
13 1972 --
13 1973 --          FUNCTION  TWindow.Focus: BOOLEAN; OVERRIDE;
13 1974 --
13 1975 --          FUNCTION  TWindow.FocusOnSuperView: BOOLEAN; OVERRIDE;
13 1976 --
13 1977 --        { Drawing Methods }
13 1978 --
13 1979 --          PROCEDURE TWindow.DrawContents; OVERRIDE;
```

```
13 1980 --
13 1981 --              PROCEDURE TWindow.Draw (area: Rect); OVERRIDE;
13 1982 --
13 1983 --              PROCEDURE TWindow.DrawResizeIcon;
13 1984 --
13 1985 --            { Mouse Handling Methods }
13 1986 --
13 1987 --              FUNCTION TWindow.HandleMouseDown (theMouse:         VPoint;
13 1988 --                                                VAR info:         EventInfo;
13 1989 --                                                VAR hysteresis:   Point;
13 1990 --                                                VAR theCommand:   TCommand): BOOLEAN; OVERRIDE;
13 1991 --
13 1992 --            { Menu Handling Methods}
13 1993 --
13 1994 --              FUNCTION  TWindow.AllowsMenuAccess: BOOLEAN;
13 1995 --
13 1996 --              PROCEDURE TWindow.DoSetupMenus; OVERRIDE;
13 1997 --
13 1998 --            { Misc. Methods }
13 1999 --
13 2000 --              PROCEDURE TWindow.Show (state, redraw: BOOLEAN); OVERRIDE;
13 2001 --
13 2002 --              FUNCTION  TWindow.IsShown: BOOLEAN; OVERRIDE;
13 2003 --
13 2004 --              PROCEDURE TWindow.Select;
13 2005 --
13 2006 --              PROCEDURE TWindow.SetTitleForDoc (newDocTitle: Str255);
13 2007 --
13 2008 --              PROCEDURE TWindow.AdaptToScreen;
13 2009 --
13 2010 --              PROCEDURE TWindow.ForceOnScreen;
13 2011 --
13 2012 --              PROCEDURE TWindow.Center (horizontally, vertically: BOOLEAN);
13 2013 --
13 2014 --              PROCEDURE TWindow.SimpleStagger (dh, dv: INTEGER; VAR counter: INTEGER);
13 2015 --
13 2016 --              FUNCTION  TWindow.GetGrafPort: GrafPtr; OVERRIDE;
13 2017 --
13 2018 --              PROCEDURE TWindow.GetGlobalBounds (VAR globalBounds: Rect);
13 2019 --
13 2020 --              FUNCTION  TWindow.GetWindow: TWindow; OVERRIDE;
13 2021 --
13 2022 --              PROCEDURE TWindow.SetResizeLimits (minSize, maxSize: Point);
13 2023 --
13 2024 --              PROCEDURE TWindow.SetTarget (newTarget: TEvtHandler);
13 2025 --
13 2026 --              PROCEDURE TWindow.InstallDocument (itsDocument: TDocument);
13 2027 --
13 2028 --            { Debugging Methods }
13 2029 --
13 2030 --          {$IFC qDebug}
13 2031 --              PROCEDURE TWindow.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2032 --                                                             fieldAddr: Ptr;
13 2033 --                                                             fieldType: INTEGER)); OVERRIDE;
13 2034 --
13 2035 --              PROCEDURE TWindow.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13 2036 --          {$ENDC}
13 2037 --
13 2038 --              END { TWindow };
13 2039 --
13 2040 --          {-----------------------------------+
13 2041 --          |   TControl                        |
13 2042 --          +-----------------------------------}
13 2043 --          ControlTemplatePtr = ^ControlTemplate;
13 2044 --          ControlTemplate =   PACKED RECORD
13 2045 --                              itsAdornment:   Integer256;     { Cast to CntlAdornment }
13 2046 --                              filler1:        0..255;
13 2047 --                              itsPenSize:     Point;
13 2048 --                              isSizable:      BOOLEAN;
13 2049 --                              isDimmed:       BOOLEAN;
13 2050 --                              isHilited:      BOOLEAN;
13 2051 --                              canDismiss:     BOOLEAN;
13 2052 --                              filler2:        0..4095;
13 2053 --                              itsInset:       Rect;
13 2054 --                              itsTextFace:    Style;
13 2055 --                              itsTextSize:    INTEGER;
13 2056 --                              itsTextColor:   RGBColor;
13 2057 --                              itsFontName:    Str255;         {Actually a variable length P-String}
13 2058 --                              END;
13 2059 --
13 2060 --          TControl =      OBJECT (TView)
13 2061 --
13 2062 --              fDefChoice:         INTEGER;
13 2063 --              fHilite:            BOOLEAN;
13 2064 --              fDimmed:            BOOLEAN;
13 2065 --              fSizeable:          BOOLEAN;
13 2066 --              fDismissesDialog:   BOOLEAN;
13 2067 --              fAdornment:         CntlAdornment;
13 2068 --              fPenSize:           Point;
13 2069 --              fInset:             Rect;
13 2070 --              fTextStyle:         TextStyle;
13 2071 --
13 2072 --          {$IFC qProceduralViews}
13 2073 --              PROCEDURE TControl.IControl (itsSuperView: TView; itsLocation, itsSize: VPoint;
13 2074 --                                           itsHSizeDet, itsVSizeDet: SizeDeterminer);
13 2075 --          {$ENDC}
13 2076 --
13 2077 --          {$IFC qTemplateViews}
13 2078 --              PROCEDURE TControl.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
```

```
13 2079 --          {$ENDC}
13 2080 --
13 2081 --          {$IFC qWriteTemplates}
13 2082 --              PROCEDURE TControl.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2083 --
13 2084 --              PROCEDURE TControl.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2085 --          {$ENDC}
13 2086 --
13 2087 --              PROCEDURE TControl.Activate (entering: BOOLEAN); OVERRIDE;
13 2088 --
13 2089 --              PROCEDURE TControl.ComputeSize (VAR newSize: VPoint); OVERRIDE;
13 2090 --
13 2091 --              FUNCTION  TControl.ContainsMouse (theMouse: VPoint): BOOLEAN; OVERRIDE;
13 2092 --
13 2093 --              PROCEDURE TControl.ControlArea (VAR theArea: Rect);
13 2094 --
13 2095 --              PROCEDURE TControl.Dim;
13 2096 --
13 2097 --              PROCEDURE TControl.DimState (state, redraw: BOOLEAN);
13 2098 --
13 2099 --              FUNCTION  TControl.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
13 2100 --                                                 VAR hysteresis: Point): TCommand; OVERRIDE;
13 2101 --
13 2102 --              PROCEDURE TControl.Draw (area: Rect); OVERRIDE;
13 2103 --
13 2104 --              FUNCTION  TControl.Focus: BOOLEAN; OVERRIDE;
13 2105 --
13 2106 --              PROCEDURE TControl.Hilite;
13 2107 --
13 2108 --              PROCEDURE TControl.HiliteState (state, redraw: BOOLEAN);
13 2109 --
13 2110 --              PROCEDURE TControl.Inset (dh, dv: INTEGER; redraw: BOOLEAN);
13 2111 --
13 2112 --              FUNCTION  TControl.IsDimmed: BOOLEAN;
13 2113 --
13 2114 --              PROCEDURE TControl.SetInset (newInset: Rect; redraw: BOOLEAN);
13 2115 --
13 2116 --              PROCEDURE TControl.InstallColor (theColor: RGBColor; redraw: BOOLEAN);
13 2117 --
13 2118 --              PROCEDURE TControl.InstallTextStyle (theTextStyle: TextStyle; redraw: BOOLEAN);
13 2119 --
13 2120 --              PROCEDURE TControl.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 2121 --
13 2122 --              PROCEDURE TControl.TrackConstrain(anchorPoint, previousPoint: VPoint;
13 2123 --                      VAR nextPoint: VPoint);
13 2124 --                  { Override this if you want to constrain the mouse point to a grid,
13 2125 --                      force drawing a square, etc.  This is called only if
13 2126 --                      fConstrainsMouse is TRUE. }
13 2127 --
13 2128 --              PROCEDURE TControl.TrackFeedback(anchorPoint, nextPoint: VPoint;
13 2129 --                      turnItOn, mouseDidMove: BOOLEAN);
13 2130 --
13 2131 --              PROCEDURE TControl.TrackMouse(aTrackPhase: TrackPhase;
13 2132 --                      VAR anchorPoint, previousPoint, nextPoint: VPoint;
13 2133 --                      mouseDidMove: BOOLEAN);
13 2134 --
13 2135 --              FUNCTION  TControl.Validate: BOOLEAN;
13 2136 --
13 2137 --          {$IFC qDebug}
13 2138 --              PROCEDURE TControl.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2139 --                                                              fieldAddr: Ptr;
13 2140 --                                                              fieldType: INTEGER)); OVERRIDE;
13 2141 --          {$ENDC}
13 2142 --
13 2143 --              END { TControl };
13 2144 --
13 2145 --          {-----------------------------------+
13 2146 --          |   TCtlMgr                         |
13 2147 --          +-----------------------------------}
13 2148 --          TCtlMgr =       OBJECT (TControl)
13 2149 --
13 2150 --              fCMgrControl:       ControlHandle;
13 2151 --
13 2152 --          {$IFC qProceduralViews}
13 2153 --              PROCEDURE TCtlMgr.ICtlMgr (itsSuperView: TView; itsLocation, itsSize: VPoint;
13 2154 --                                         itsHSizeDet, itsVSizeDet: SizeDeterminer;
13 2155 --                                         itsTitle: Str255; itsVal, itsMin, itsMax, itsProcID: INTEGER);
13 2156 --          {$ENDC}
13 2157 --
13 2158 --          {$IFC qTemplateViews}
13 2159 --              PROCEDURE TCtlMgr.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 2160 --
13 2161 --              PROCEDURE TCtlMgr.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2162 --
13 2163 --              PROCEDURE TCtlMgr.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2164 --          {$ENDC}
13 2165 --
13 2166 --              PROCEDURE TCtlMgr.Free; OVERRIDE;
13 2167 --
13 2168 --              PROCEDURE TCtlMgr.BeInPort (itsPort: GrafPtr); OVERRIDE;
13 2169 --
13 2170 --              PROCEDURE TCtlMgr.CreateCMgrControl (itsBounds: Rect; itsTitle: Str255;
13 2171 --                                                   itsValue, itsMin, itsMax, itsProcID: INTEGER);
13 2172 --
13 2173 --              PROCEDURE TCtlMgr.DimState (state, redraw: BOOLEAN); OVERRIDE;
13 2174 --
13 2175 --              FUNCTION  TCtlMgr.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
13 2176 --                                                VAR hysteresis: Point): TCommand; OVERRIDE;
13 2177 --
```

```
13 2178 --              PROCEDURE TCtlMgr.Draw (area: Rect); OVERRIDE;
13 2179 --
13 2180 --              FUNCTION  TCtlMgr.GetMax: INTEGER;
13 2181 --
13 2182 --              FUNCTION  TCtlMgr.GetMin: INTEGER;
13 2183 --
13 2184 --              PROCEDURE TCtlMgr.GetText (VAR theText: Str255);
13 2185 --
13 2186 --              FUNCTION  TCtlMgr.GetVal: INTEGER;
13 2187 --
13 2188 --              PROCEDURE TCtlMgr.HiliteState (state, redraw: BOOLEAN); OVERRIDE;
13 2189 --
13 2190 --              PROCEDURE TCtlMgr.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
13 2191 --
13 2192 --              PROCEDURE TCtlMgr.SetMax (itsMax: INTEGER; redraw: BOOLEAN);
13 2193 --
13 2194 --              PROCEDURE TCtlMgr.SetMin (itsMin: INTEGER; redraw: BOOLEAN);
13 2195 --
13 2196 --              PROCEDURE TCtlMgr.SetText (itsText: Str255; redraw: BOOLEAN);
13 2197 --
13 2198 --              PROCEDURE TCtlMgr.SetVal (newVal: INTEGER; redraw: BOOLEAN);
13 2199 --
13 2200 --              PROCEDURE TCtlMgr.SetValues (itsVal, itsMin, itsMax: INTEGER; redraw: BOOLEAN);
13 2201 --
13 2202 --              PROCEDURE TCtlMgr.WhileFocused (PROCEDURE DoToControl; redraw: BOOLEAN);
13 2203 --
13 2204 --          {$IFC qDebug}
13 2205 --              PROCEDURE TCtlMgr.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2206 --                                                             fieldAddr: Ptr;
13 2207 --                                                             fieldType: INTEGER)); OVERRIDE;
13 2208 --          {$ENDC}
13 2209 --              END { TCtlMgr };
13 2210 --
13 2211 --          {-----------------------------------+
13 2212 --          |   TScrollBar                      |
13 2213 --          +-----------------------------------}
13 2214 --          ScrollBarTemplatePtr =  ^ScrollBarTemplate;
13 2215 --          ScrollBarTemplate =     RECORD
13 2216 --                                  value:      LONGINT;
13 2217 --                                  minimum:    LONGINT;
13 2218 --                                  maximum:    LONGINT;
13 2219 --                                  END;
13 2220 --
13 2221 --          TScrollBar =    OBJECT (TCtlMgr)
13 2222 --
13 2223 --              fBitsToShift:       INTEGER;    { # bits shifted to convert between
13 2224 --                                                short and long value. }
13 2225 --              fDirection:         VHSelect;
13 2226 --              fLongVal:           VCoordinate;
13 2227 --              fLongMin:           VCoordinate;
13 2228 --              fLongMax:           VCoordinate;
13 2229 --
13 2230 --          {$IFC qProceduralViews}
13 2231 --              PROCEDURE TScrollBar.IScrollBar (itsSuperView: TView; itsLocation, itsSize: VPoint;
13 2232 --                                               itsHSizeDet, itsVSizeDet: SizeDeterminer;
13 2233 --                                               itsDirection: VHSelect; itsVal, itsMin, itsMax: LONGINT);
13 2234 --          {$ENDC}
13 2235 --
13 2236 --          {$IFC qTemplateViews}
13 2237 --              PROCEDURE TScrollBar.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 2238 --          {$ENDC}
13 2239 --
13 2240 --          {$IFC qWriteTemplates}
13 2241 --              PROCEDURE TScrollBar.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2242 --
13 2243 --              PROCEDURE TScrollBar.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2244 --          {$ENDC}
13 2245 --
13 2246 --              PROCEDURE TScrollBar.DeltaValue (delta: VCoordinate);
13 2247 --
13 2248 --              FUNCTION  TScrollBar.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
13 2249 --                                                   VAR hysteresis: Point): TCommand; OVERRIDE;
13 2250 --
13 2251 --              FUNCTION  TScrollBar.GetLongMax: VCoordinate;
13 2252 --
13 2253 --              FUNCTION  TScrollBar.GetLongMin: VCoordinate;
13 2254 --
13 2255 --              FUNCTION  TScrollBar.GetLongVal: VCoordinate;
13 2256 --
13 2257 --              PROCEDURE TScrollBar.SetLongMax (itsMax: VCoordinate; redraw: BOOLEAN);
13 2258 --
13 2259 --              PROCEDURE TScrollBar.SetLongMin (itsMin: VCoordinate; redraw: BOOLEAN);
13 2260 --
13 2261 --              PROCEDURE TScrollBar.SetLongVal (itsVal: VCoordinate; redraw: BOOLEAN);
13 2262 --
13 2263 --              PROCEDURE TScrollBar.SetLongValues (itsVal, itsMin, itsMax: VCoordinate; redraw: BOOLEAN);
13 2264 --
13 2265 --              PROCEDURE TScrollBar.TrackScrollBar (partCode: INTEGER);
13 2266 --
13 2267 --            { Debugging Methods }
13 2268 --
13 2269 --          {$IFC qDebug}
13 2270 --              PROCEDURE TScrollBar.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2271 --                                                                fieldAddr: Ptr;
13 2272 --                                                                fieldType: INTEGER)); OVERRIDE;
13 2273 --          {$ENDC}
13 2274 --              END { TScrollBar };
13 2275 --
13 2276 --          {-----------------------------------+
```

```
13 2277 --          |   TSScrollBar                     |
13 2278 --          +-----------------------------------}
13 2279 --          TSScrollBar =   OBJECT (TScrollBar)
13 2280 --
13 2281 --              fScrollers:         TList;
13 2282 --
13 2283 --          {$IFC qProceduralViews}
13 2284 --              PROCEDURE TSScrollBar.ISScrollBar (itsSuperView: TView; itsLocation, itsSize: VPoint;
13 2285 --                                      itsHSizeDet, itsVSizeDet: SizeDeterminer;
13 2286 --                                      itsDirection: VHSelect; itsMax: LONGINT;
13 2287 --                                      itsScroller: TScroller);
13 2288 --          {$ENDC}
13 2289 --
13 2290 --          {$IFC qTemplateViews}
13 2291 --              PROCEDURE TSScrollBar.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 2292 --          {$ENDC}
13 2293 --
13 2294 --          {$IFC qWriteTemplates}
13 2295 --              PROCEDURE TSScrollBar.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2296 --
13 2297 --              PROCEDURE TSScrollBar.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
13 2298 --          {$ENDC}
13 2299 --
13 2300 --              PROCEDURE TSScrollBar.Free; OVERRIDE;
13 2301 --
13 2302 --              PROCEDURE TSScrollBar.Activate (entering: BOOLEAN); OVERRIDE;
13 2303 --
13 2304 --              FUNCTION  TSScrollBar.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
13 2305 --                                      VAR hysteresis: Point): TCommand; OVERRIDE;
13 2306 --
13 2307 --              PROCEDURE TSScrollBar.Draw (area: Rect); OVERRIDE;
13 2308 --
13 2309 --              PROCEDURE TSScrollBar.TrackScrollBar (partCode: INTEGER); OVERRIDE;
13 2310 --
13 2311 --            { Debugging Methods }
13 2312 --
13 2313 --          {$IFC qDebug}
13 2314 --              PROCEDURE TSScrollBar.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2315 --                                                      fieldAddr: Ptr;
13 2316 --                                                      fieldType: INTEGER)); OVERRIDE;
13 2317 --          {$ENDC}
13 2318 --              END { TSScrollBar };
13 2319 --
13 2320 --
13 2321 --          {-----------------------------------+
13 2322 --          |   TPrintHandler                   |
13 2323 --          +-----------------------------------}
13 2324 --          TPrintHandler = OBJECT(TEvtHandler)
13 2325 --
13 2326 --              fView:                  TView;      { The view whose printing is handled }
13 2327 --              fDeviceRes:             Point;      { formal printer resolution, spots
13 2328 --                                                    per inch }
13 2329 --              fEffectiveDeviceRes:    Point;      { true effective spots per inch of
13 2330 --                                      printer; larger than the fDeviceRes if
13 2331 --                                      REDUCTION is in effect, smaller if
13 2332 --                                      ENLARGEMENT is, same as fDeviceRes if
13 2333 --                                      neither is in effect. }
13 2334 --              fViewPerPage:       VPoint;     { The amount of the view that is visible on a page }
13 2335 --              fFocusedPage:       INTEGER;    { The page number currently focused }
13 2336 --
13 2337 --          { Initialization and termination }
13 2338 --
13 2339 --              PROCEDURE TPrintHandler.IPrintHandler(itsView: TView);
13 2340 --                  { Initialize the printHandler, and associate it and 'itsView' with
13 2341 --                    each other }
13 2342 --
13 2343 --          {$IFC qDebug}
13 2344 --              PROCEDURE TPrintHandler.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13 2345 --
13 2346 --              PROCEDURE TPrintHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2347 --                                                      fieldAddr: Ptr;
13 2348 --                                                      fieldType: INTEGER)); OVERRIDE;
13 2349 --          {$ENDC}
13 2350 --
13 2351 --              PROCEDURE TPrintHandler.FocusOnInterior;
13 2352 --
13 2353 --          { Printing-related reformatting and computations }
13 2354 --
13 2355 --              FUNCTION  TPrintHandler.BreakFollowing(vhs: VHSelect;
13 2356 --                      prevBreak: VCoordinate; VAR automatic: BOOLEAN): VCoordinate;
13 2357 --                  { Returns the location of the page break which follows the page break
13 2358 --                    located at 'prevBreak', in direction vhs; returns automatic =
13 2359 --                    TRUE if the page-break is thought of as an 'automatic' rather
13 2360 --                    than a 'manual' (user-specified) one.  Note that page-breaks in
13 2361 --                    dimension 'v' are drawn as vertical lines, those in dimension
13 2362 --                    'h' as horizontal lines }
13 2363 --
13 2364 --              PROCEDURE TPrintHandler.CalcPageStrips (VAR pageStrips: Point);
13 2365 --                  { Recalculate the number of strips of pages in each dimension }
13 2366 --
13 2367 --              PROCEDURE TPrintHandler.CalcViewPerPage(VAR amtPerPage: VPoint);
13 2368 --                  { Computes the amount of view, in each dimension, to be allocated to
13 2369 --                    a printed page }
13 2370 --
13 2371 --              PROCEDURE TPrintHandler.CheckPrinter;
13 2372 --                  { See if there are changed printer-configuration parameters which
13 2373 --                    need to be absorbed, and if so, absorb them }
13 2374 --
13 2375 --              PROCEDURE TPrintHandler.LocatePageInterior(pageNumber: INTEGER;
```

```
13 2376 --                      VAR loc: Point);
13 2377 --                  { Decide where the top-left-most point of the the interior of the
13 2378 --                      page should be }
13 2379 --
13 2380 --              PROCEDURE TPrintHandler.PrinterChanged;
13 2381 --                  { The metrics relating to printer use have changed; absorb the
13 2382 --                      information and react }
13 2383 --
13 2384 --              PROCEDURE TPrintHandler.RedoPageBreaks;
13 2385 --                  { Recompute the dividing lines between areas of the view which will
13 2386 --                      be mapped into different printed pages }
13 2387 --
13 2388 --              PROCEDURE TPrintHandler.Reset;
13 2389 --                  { Resets the print handler to the default values }
13 2390 --
13 2391 --          { Printing-related commands }
13 2392 --
13 2393 --              FUNCTION TPrintHandler.Print(itsCmdNumber: CmdNumber; VAR proceed: BOOLEAN): TCommand;  {+}
13 2394 --
13 2395 --              FUNCTION TPrintHandler.SetupForFinder: BOOLEAN;
13 2396 --                  { Intended to set up the print handler for finder printing. }
13 2397 --
13 2398 --
13 2399 --           { Printing-related screen feedback }
13 2400 --
13 2401 --              PROCEDURE TPrintHandler.DrawPrintFeedback(area: Rect);
13 2402 --                  { Draw page-breaks, page-numbers, view-borders, rulers, etc. }
13 2403 --
13 2404 --              PROCEDURE TPrintHandler.DrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
13 2405 --                                                     loc: VCoordinate; automatic: BOOLEAN);
13 2406 --
13 2407 --           { Actual printing }
13 2408 --
13 2409 --              FUNCTION  TPrintHandler.MaxPageNumber: INTEGER;
13 2410 --                  { Returns the largest page number which could be reasonably printed,
13 2411 --                      given the properties of the view; in cases like a SpreadSheet,
13 2412 --                      this is not necessarily as large as the product (row strips) x
13 2413 --                      (column strips) }
13 2414 --
13 2415 --              PROCEDURE TPrintHandler.SetPageInterior(pageNumber: INTEGER);
13 2416 --                  { Set up the pad-space Interior rectangle for the given page number }
13 2417 --
13 2418 --              PROCEDURE TPrintHandler.SetPageOffset(coord: VPoint);
13 2419 --                  { Given the view coordinates of the top-left-most point of the view
13 2420 --                      which is being mapped into the current page (in coord), this
13 2421 --                      method's job is to compute the correct fRelOrigin field for the
13 2422 --                      current page }
13 2423 --
13 2424 --              END;
13 2425 --
13 2426 --
13 2427 --          {-----------------------------------+
13 2428 --          |   TCommand                        |
13 2429 --          +-----------------------------------}
13 2430 --          TCommand = OBJECT(TObject)
13 2431 --
13 2432 --              fCmdNumber:          CmdNumber;      { Can be a many-to-one mapping of
13 2433 --                                                     fCmdNumber to TCommand }
13 2434 --              fChangedDocument:    TDocument;      { the document changed by this
13 2435 --                                                     command; defaults to gDocument
13 2436 --                                                     global variable }
13 2437 --              fTarget:             TEvtHandler;    { The target tp set before calling
13 2438 --                                                     UndoIt or RedoIt }
13 2439 --              fCmdDone:            BOOLEAN;        { TRUE if the last "execution"
13 2440 --                                                     message sent to the command was
13 2441 --                                                     DoIt or RedoIt }
13 2442 --              fCanUndo:            BOOLEAN;        { Defaults to TRUE }
13 2443 --              fCausesChange:       BOOLEAN;        { Defaults to TRUE; Marks document
13 2444 --                                                     changed when command is done }
13 2445 --              fChangesClipboard:   BOOLEAN;        { Defaults to FALSE.  Set to true for
13 2446 --                                                     command subclasses representing
13 2447 --                                                     CUT and/ or COPY commands which
13 2448 --                                                     change the Clipboard. }
13 2449 --              { The remaining fields are for command objects that track the mouse }
13 2450 --              fView:               TView;          { The view in which tracking takes place }
13 2451 --              fConstrainsMouse:    BOOLEAN;        { Defaults to FALSE; if you set to
13 2452 --                                                     TRUE then TrackConstrain will be
13 2453 --                                                     called to constrain the mouse. }
13 2454 --              fViewConstrain:      BOOLEAN;        { Defaults to TRUE, which means
13 2455 --                                                     constrain mouse to view limits }
13 2456 --              fTrackNonMovement:   BOOLEAN;        { Defaults to FALSE, which means don't
13 2457 --                                                     track mouse unless it moves. }
13 2458 --              fScroller:           TScroller;      { The scroller that handles auto-scrolling }
13 2459 --
13 2460 --            { Init & Free }
13 2461 --              PROCEDURE TCommand.ICommand(itsCmdNumber: CmdNumber; itsDocument: TDocument;
13 2462 --                                          itsView: TView; itsScroller: TScroller);
13 2463 --
13 2464 --          {$IFC qDebug}
13 2465 --            { Debugging }
13 2466 --              PROCEDURE TCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2467 --                                                              fieldAddr: Ptr;
13 2468 --                                                              fieldType: INTEGER)); OVERRIDE;
13 2469 --          {$ENDC}
13 2470 --
13 2471 --              { command processing - usually overridden }
13 2472 --
13 2473 --              PROCEDURE TCommand.Commit;
13 2474 --              PROCEDURE TCommand.DoIt;
```

```
13 2475 --              PROCEDURE TCommand.RedoIt;
13 2476 --              PROCEDURE TCommand.UndoIt;
13 2477 --
13 2478 --              { Mouse Tracking }
13 2479 --
13 2480 --              PROCEDURE TCommand.TrackConstrain(anchorPoint, previousPoint: VPoint;
13 2481 --                      VAR nextPoint: VPoint);
13 2482 --                  { Override this if you want to constrain the mouse point to a grid,
13 2483 --                      force drawing a square, etc.  This is called only if
13 2484 --                      fConstrainsMouse is TRUE. }
13 2485 --
13 2486 --              PROCEDURE TCommand.TrackFeedback(anchorPoint, nextPoint: VPoint;
13 2487 --                      turnItOn, mouseDidMove: BOOLEAN);
13 2488 --                  { Default is: IF NOT mouseDidMove THEN FrameRect(<rect formed by
13 2489 --                      anchorPoint and nextPoint>).  Before this is called the pen
13 2490 --                      is set to PenNormal and the PenMode to PatXOR. }
13 2491 --
13 2492 --              FUNCTION  TCommand.TrackMouse(aTrackPhase: TrackPhase;
13 2493 --                      VAR anchorPoint, previousPoint, nextPoint: VPoint;
13 2494 --                      mouseDidMove: BOOLEAN): TCommand;
13 2495 --                  { This does the mouse tracking.
13 2496 --
13 2497 --                  The default returns SELF.  Override it to force gridding, to return
13 2498 --                  a different type of command, or to perform the command on mouse
13 2499 --                  release instead of using the usual DoIt approach.
13 2500 --
13 2501 --                  All points are in local coordinates.
13 2502 --
13 2503 --                  If aTrackPhase = trackPress then all 3 points will be the same.
13 2504 --                  If aTrackPhase = trackRelease then nextPoint will be the coordinate
13 2505 --                      in the mouseUp event (if a mouseUp event is found), otherwise
13 2506 --                      the same point as previousPoint.  Generally, you will ignore
13 2507 --                      nextPoint and just look at previousPoint.
13 2508 --
13 2509 --                  mouseDidMove will be TRUE if aTrackPhase = trackPress or
13 2510 --                      trackRelease; otherwise, it will indicate if nextPoint =
13 2511 --                      previousPoint.
13 2512 --
13 2513 --                  If you change anchorPoint, the new value will be passed to you next
13 2514 --                      time.  The value of nextPoint at the time the routine exits will
13 2515 --                      be passed to you as previousPoint the next time the routine is
13 2516 --                      called.  You can change nextPoint if you wish.
13 2517 --                  Usually, however, you will do gridding in the TrackConstrain method. }
13 2518 --
13 2519 --              PROCEDURE TCommand.AutoScroll (deltaH, deltaV: VCoordinate);
13 2520 --                  { Implements autoscrolling by calling fScroller.ScrollBy. }
13 2521 --
13 2522 --              END;
13 2523 --
13 2524 --
13 2525 --          {-----------------------------------+
13 2526 --          |   TControlTracker                 |
13 2527 --          +-----------------------------------}
13 2528 --          TControlTracker =   OBJECT (TCommand)
13 2529 --
13 2530 --              fControl:           TControl;
13 2531 --
13 2532 --              PROCEDURE TControlTracker.IControlTracker (theControl: TControl);
13 2533 --
13 2534 --              FUNCTION  TControlTracker.TrackMouse(aTrackPhase: TrackPhase;
13 2535 --                                          VAR anchorPoint, previousPoint, nextPoint: VPoint;
13 2536 --                                          mouseDidMove: BOOLEAN): TCommand; OVERRIDE;
13 2537 --
13 2538 --              PROCEDURE TControlTracker.TrackFeedback(anchorPoint, nextPoint: VPoint;
13 2539 --                                              turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
13 2540 --
13 2541 --              PROCEDURE TControlTracker.TrackConstrain (anchorPoint, previousPoint: VPoint;
13 2542 --                                          VAR nextPoint: VPoint); OVERRIDE;
13 2543 --
13 2544 --              END { TControlTracker };
13 2545 --
13 2546 --
13 2547 --          {-----------------------------------+
13 2548 --          |   TDeskScrapView                  |
13 2549 --          +-----------------------------------}
13 2550 --          TDeskScrapView = OBJECT(TView)
13 2551 --
13 2552 --              fHavePicture:   BOOLEAN;
13 2553 --              fHaveText:      BOOLEAN;
13 2554 --              fScrapCount:    INTEGER;    { scrapCount of the scrap represented by
13 2555 --                                              this view.}
13 2556 --
13 2557 --              fDataHandle:    Handle;     { if non-NIL, will be either a PicHandle to
13 2558 --                                              my picture or a Handle to my Text }
13 2559 --
13 2560 --          {$IFC qProceduralViews}
13 2561 --              PROCEDURE TDeskScrapView.IDeskScrapView;
13 2562 --          {$ENDC}
13 2563 --
13 2564 --          {$IFC qTemplateViews}
13 2565 --              PROCEDURE TDeskScrapView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
13 2566 --          {$ENDC}
13 2567 --
13 2568 --              PROCEDURE TDeskScrapView.Free; OVERRIDE;
13 2569 --
13 2570 --              PROCEDURE TDeskScrapView.CheckScrapContents;
13 2571 --              PROCEDURE TDeskScrapView.CalcMinSize (VAR minSize: VPoint); OVERRIDE;
13 2572 --              PROCEDURE TDeskScrapView.Draw(area: Rect); OVERRIDE;
13 2573 --              PROCEDURE TDeskScrapView.SuperViewChangedSize (delta: VPoint; invalidate: BOOLEAN); OVERRIDE;
```

```
13 2574 --              PROCEDURE TDeskScrapView.WriteToDeskScrap; OVERRIDE;
13 2575 --
13 2576 --          {$IFC qDebug}
13 2577 --            { Debugging }
13 2578 --              PROCEDURE TDeskScrapView.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
13 2579 --
13 2580 --              PROCEDURE TDeskScrapView.Fields (PROCEDURE DoToField (fieldName: Str255;
13 2581 --                                                                    fieldAddr: Ptr;
13 2582 --                                                                    fieldType: INTEGER)); OVERRIDE;
13 2583 --          {$ENDC}
13 2584 --
13 2585 --              END;
13 2586 --
13 2587 --
13 2588 --      {-----------------------------------+
13 2589 --      |   Global Variables                |
13 2590 --      +-----------------------------------}
13 2591 --      VAR
13 2592 --          gAppDone:           BOOLEAN;            { set this to TRUE when you want the
13 2593 --                                                    application to terminate }
13 2594 --          gApplication:       TApplication;       { the application object }
13 2595 --      {$IFC qDebug}
13 2596 --          gBusyTempRgn:   BOOLEAN;        { Is gTempRgn in use? }
13 2597 --      {$ENDC}
13 2598 --          gChooserOK:         BOOLEAN;            { if FALSE, user will not be allowed
13 2599 --                                                to change the printer selection using
13 2600 --                                                'Choose Printer' DA while the application is
13 2601 --                                                alive; if TRUE (the default set in
13 2602 --                                                InitToolBox), then printer-change will only
13 2603 --                                                be disabled during background printing.  If
13 2604 --                                                you want it FALSE in your application, the
13 2605 --                                                right time to set it is after calling
13 2606 --                                                InitToolBox (where it is initialized to
13 2607 --                                                TRUE) and before calling InitPrinting
13 2608 --                                                (where, if it's found to be FALSE, the low-
13 2609 --                                                memory location governing is option is poked) }
13 2610 --          gClickCount:    INTEGER;        { number of 'saved up' mouse clicks; handled in
13 2611 --                                                TApplication.DispatchEvent, ObeyEvent &
13 2612 --                                                ObeyMouseDown.
13 2613 --                                            If a mouseDown, the value is that returned by
13 2614 --                                                the call to TApplication.CountClicks.
13 2615 --                                            If a mouseUp, the value is left alone.
13 2616 --                                                    Otherwise, set to 0.
13 2617 --                                            Also set to 0 is the 2 clicks were in different
13 2618 --                                                parts ofthe window. }
13 2619 --          gClipClaimed:   BOOLEAN;        { Used by PerformCommand & ClaimClipboard to determine, if
13 2620 --                                            DoIt of a cut/copy cmd fails, whether the Clipboard had
13 2621 --                                            already been claimed by the new command or not }
13 2622 --          gClipOrphanage:     TDeskScrapView;     { A view to represent the Clipboard
13 2623 --                                                    when it contains TEXT or PICT
13 2624 --                                                    data }
13 2625 --
13 2626 --          gClipUndoView:      TView;          { The view previously installed in
13 2627 --                                                the Clipboard }
13 2628 --
13 2629 --          gClipView:          TView;          { The view currently installed in the
13 2630 --                                                Clipboard }
13 2631 --          gClipWindow:        TWindow;            { The window holding the Clipboard
13 2632 --                                                    display }
13 2633 --          gClipWrittenToDeskScrap: BOOLEAN;       { True if the clipboard view has been
13 2634 --                                                    written to the desk scrap. }
13 2635 --          gCouldPrint:        BOOLEAN;            { whether Printer code is accessible
13 2636 --                                                    to the application }
13 2637 --          gCurrPrintHandler:  TPrintHandler;      { If printing, this is set to the print
13 2638 --                                                    handler.  Nil if not printing. }
13 2639 --          gCursorRgn:         RgnHandle;          { the current cursor region }
13 2640 --      {$IFC qDebug}
13 2641 --          gDebugPrinting:     BOOLEAN;    { simple toggle for debugging printing }
13 2642 --      {$ENDC}
13 2643 --          gDocument:          TDocument;          { the current document }
13 2644 --          gDocList:           TList;              { list of documents }
13 2645 --          gDrawingPictScrap:  BOOLEAN;            { TRUE if a view's Draw routine is
13 2646 --                                                being called in order to create PICT data in
13 2647 --                                                the Desk Scrap; your View.Draw can check
13 2648 --                                                this routine, and insert PictComments as
13 2649 --                                                appropriate if it wishes to have them in
13 2650 --                                                the Picture in the Desk Scrap }
13 2651 --          gErrorParm3:        Str255;             { This is used as the last argument
13 2652 --                                                to ParamText in ErrorAlert, if ErrorAlert
13 2653 --                                                displays one of the standard alerts.  (This
13 2654 --                                                string will replace ^3 in those alerts.)
13 2655 --                                                ErrorAlert also sets this to '' when called.
13 2656 --                                                You can use this to parameterize the
13 2657 --                                                automatic alerts that MacApp displays.  For
13 2658 --                                                example, TDocument.ReadFromFile sets this to
13 2659 --                                                the document name. }
13 2660 --          gEventLevel:        INTEGER;        { A count of the number of nested calls to
13 2661 --                                                PollEvent. }
13 2662 --      {$IFC qDebug}
13 2663 --          gExperimenting:     BOOLEAN;    { simple toggle for enabling/disabling
13 2664 --                                                experimental features }
13 2665 --      {$ENDC}
13 2666 --          gFakeWindow:        WindowRecord;   { Initialized with OpenPort, and by
13 2667 --                                                setting its ControlList to NIL.
13 2668 --                                                This is NOT a real window, just
13 2669 --                                                a port. }
13 2670 --          gFileCount:     INTEGER;        { # files to open/print from finder; set in Init2 }
13 2671 --          gFinderPrinting:    BOOLEAN;            { TRUE if the Finder started the
13 2672 --                                                    App up just for printing docs }
```

```
13 2673 --          gFocusedView:       TView;              { the view that is currently focused }
13 2674 --          gFreeWindowList:    TList;              { list of free-standing
13 2675 --                                                      (documentless) windows }
13 2676 --          gFrontWindow:       TWindow;            { If non-NIL the TWindow object of
13 2677 --                                                      the FrontWindow. }
13 2678 --          gGotClipType:       BOOLEAN;            { True if we can paste the data in the clipboard }
13 2679 --          gHeadCohandler:     TEvtHandler;        { head of linked list of global
13 2680 --                                                      co-handlers }
13 2681 --          gIdleFreq:          LONGINT;            { number of ticks needed to call DoIdle's
13 2682 --                                                      since the last DoIdle's. }
13 2683 --          gIdlePhase:         IdlePhase;          { The current idle phase }
13 2684 --          gInBackground:      BOOLEAN;            { True if app is currently in background }
13 2685 --          gInitialized:       BOOLEAN;            { Set to TRUE at the end of
13 2686 --                                                      IApplication }
13 2687 --      {$IFC qDebug}
13 2688 --          gIntenseDebugging:  BOOLEAN;            { debugging toggle for intensive debugging }
13 2689 --      {$ENDC}
13 2690 --          gLastClickPart:     INTEGER;            { the list window part clicked in-used for double
13 2691 --                                                      click detection }
13 2692 --          gLastCommand:       TCommand;           { the last command done or undone by
13 2693 --                                                      the user }
13 2694 --          gLastDeskAcc:       LONGINT;            { time of the most recent possible
13 2695 --                                                      excursion to a a Desk Accessory }
13 2696 --          gLastIdle:          LONGINT;            { time in ticks of the last time the
13 2697 --                                                      event handlers were idled. }
13 2698 --          gLastMsePt:         Point;              { coordinates of mouse in last event
13 2699 --                                                      passed to TApplication.CountClicks }
13 2700 --          gLastUpTime:        LONGINT;            { time of last mouse up event passed
13 2701 --                                                      to TApplication.ObeyEvent }
13 2702 --          gLongOffset:        VPoint;
13 2703 --          gLowSpaceInterval:  LONGINT;            { If >= 0, the frequency (in Ticks)
13 2704 --                                                      with which MacApp displays a low
13 2705 --                                                      space alert.  (Defaults to
13 2706 --                                                      kLowSpaceInterval.)  If < 0,
13 2707 --                                                      MacApp doesn't display an alert. }
13 2708 --          gMainEventMask:     INTEGER;            { Event mask used in main event loop.
13 2709 --                                                      Initialized by InitToolbox. }
13 2710 --          gMainFileType:      OSType;             { principal file type opened/printed
13 2711 --                                                  by application; set in
13 2712 --                                                  TApplication.IApplication; by default,
13 2713 --                                                  TApplication.SFGetFilters returns a list of
13 2714 --                                                  just this }
13 2715 --          gMBarDisplayed:     INTEGER;        { menus that are read in and installed in
13 2716 --                                                  menu bar }
13 2717 --          gMBarHeight:        INTEGER;        { Height of the menu bar in pixels }
13 2718 --          gMBarHierarchical:  INTEGER;        { menus that pop up when a menu item is
13 2719 --                                                  choosen }
13 2720 --          gMBarNotDisplayed:  INTEGER;        { menus that are read in but not installed }
13 2721 --          gNewScrapStuff:     ScrapStuff;
13 2722 --          gNextSpaceMsg:      LONGINT;        { time when next low space message should
13 2723 --                                                  be displayed }
13 2724 --          gNoChanges:         TCommand;           { special TCommand object to return
13 2725 --                                                  if the command does not change the document.
13 2726 --                                                  You should carry out the command in the
13 2727 --                                                  DoMenuCommand procedure. }
13 2728 --          gNullPrintHandler:  TPrintHandler;      { handles printing-relating messages
13 2729 --                                                      for views which don't print }
13 2730 --          gNumUntitled:       INTEGER;            { The number to assign to the next
13 2731 --                                                  Untitled document (assuming that the app
13 2732 --                                                  provides a template for filling in the
13 2733 --                                                  number.  You must change STR# resource with
13 2734 --                                                  ID 0 / index 3 to read (for example)
13 2735 --                                                  Untitled-<<<#>>>. }
13 2736 --          gOldChooserFlag:    BOOLEAN;        { The state of the chooser alert flag when
13 2737 --                                                  the application started up. }
13 2738 --          gOldScrapStuff:     ScrapStuff;
13 2739 --          gOrthogonal:        ARRAY[VHSelect] OF VHSelect;
13 2740 --          gPageOffset:        VPoint;             { offset in view of page being
13 2741 --                                                      printed }
13 2742 --          gPrefClipType:      ResType;
13 2743 --          gPrintHandler:      TPrintHandler;      { a global print-handler object for
13 2744 --                                                  use in some standard printing-related
13 2745 --                                                  activities; this is initialized to be just
13 2746 --                                                  a reference to gNullPrintHandler, but if
13 2747 --                                                  you call InitPrinting, that will install a
13 2748 --                                                  non-trivial print-handler here… }
13 2749 --          gPrinting:          BOOLEAN;            { true if currently Printing }
13 2750 --      {$IFC qTemplateViews}
13 2751 --          gProtoList:         TAssociation;   { List of object prototypes for cloning}
13 2752 --      {$ENDC}
13 2753 --      {$IFC qDebug}
13 2754 --          gReportEvt:         BOOLEAN;    { debugging toggle for reporting events }
13 2755 --          gReportMenuChoices: BOOLEAN;    { debugging toggle for tracing cmds }
13 2756 --          gRsrcCheck:         INTEGER;    { debugging toggle for checking resource
13 2757 --                                              usage }
13 2758 --      {$ENDC}
13 2759 --
13 2760 --          { Set by GetFocus; used by SetFocus.  GetFocus/SetFocus are used to speed up
13 2761 --              scrolling (see TScroller). }
13 2762 --          gSaveClip:          RgnHandle;
13 2763 --          gSaveFocusedView:   TView;
13 2764 --          gSaveOrg:           Point;
13 2765 --          gSaveLongOffset:    VPoint;
13 2766 --          gSavePort:          WindowPtr;
13 2767 --
13 2768 --      {$IFC qTemplateViews}
13 2769 --          gSignatures:        ARRAY [1..kMaxSignatures] OF IDType;        { Standard view signatures }
13 2770 --          gSignatureViews:    ARRAY [1..kMaxSignatures] OF TObject;
13 2771 --          gSignatureCount:    INTEGER;
```

```
13 2772 --      {$ENDC}
13 2773 --
13 2774 --          gStdHysteresis:     Point;      { standard hysteresis used in TCommand.ICommand }
13 2775 --          gStdStaggerCount:   INTEGER;    { Used to stagger windows created from templates }
13 2776 --          gStdWMoveBounds:    Rect;       { standard boundsRect to pass to DragWindow
13 2777 --                                              Toolbox routine }
13 2778 --          gStdWSizeRect:      Rect;       { standard sizeRect to pass to GrowWindow
13 2779 --                                              Toolbox routine }
13 2780 --          gStdWScreenRect:    Rect;       { Used to force a window onto the screen.
13 2781 --                                              If the window is not in the GrayRgn,
13 2782 --                                              then at least one corner of the window
13 2783 --                                              must lie within gStdWScreenRect. }
13 2784 --
13 2785 --          gSysWindowActive:   BOOLEAN;    { TRUE if the front window is a system window }
13 2786 --          gSystemStyle:       TextStyle;      { System's default text style }
13 2787 --          gTarget:            TEvtHandler;        { the TEvtHandler that gets the first
13 2788 --                                          chance at DoCommand, DoSetupMenu,
13 2789 --                                          DoKeyCommand, Idle; should never be NIL --
13 2790 --                                          If you do not want your own target set this
13 2791 --                                          to the application object }
13 2792 --          { gTempRgn is a temporary RgnHandle created in InitToolbox.
13 2793 --              When debugging: before using gTempRgn, call UseTempRgn and when
13 2794 --              done call DoneWithTempRgn; this ensures that 2 routines do not try to
13 2795 --              use gTempRgn and the same time }
13 2796 --          gTempRgn:           RgnHandle;
13 2797 --      {$IFC qTrace}
13 2798 --          gTraceIdle:         BOOLEAN;
13 2799 --      {$ENDC}
13 2800 --          gUndoState:         BOOLEAN;
13 2801 --          gUndoCmd:           CmdNumber;
13 2802 --      {$IFC qDebug}
13 2803 --          gUsedBy:        Str255;         { The routine using gTempRgn }
13 2804 --      {$ENDC}
13 2805 --          gVarClipPicSize:    BOOLEAN;            { if TRUE, Pictures in the Clipboard
13 2806 --                                          are treated as variable size, depending on
13 2807 --                                          the window size; if FALSE (default), then
13 2808 --                                          pictures in the Clipboard are drawn and
13 2809 --                                          pasted ??? actual size }
13 2810 --          gWorkPort:          GrafPtr;        { Pointer to a graf port that is created
13 2811 --                                              during MacApp initialization }
13 2812 --          gWResSignature:     IDType;         { Signature used by TView.WRes }
13 2813 --          gWResType:          Str255;         { Type used by TView.WRes }
13 2814 --          gZeroRect:          Rect;           { SetRect(gZeroRect, 0, 0, 0, 0); }
13 2815 --          gZeroVPt:           VPoint;         { SetVPt(gZeroVPt, 0, 0); }
13 2816 --          gZeroVRect:         VRect;          { SetVRect(gZeroVRect, 0, 0, 0, 0); }
13 2817 --
13 2818 --
13 2819 --      {-----------------------------------+
13 2820 --      |   Global Procedures               |
13 2821 --      +-----------------------------------}
13 2822 --
13 2823 --      { D E B U G G I N G }
13 2824 --
13 2825 --      {$IFC qDebug}
13 2826 --      PROCEDURE DoneWithTempRgn;
13 2827 --          { Indicates that gTempRgn is no longer in use.  Call this only if qDebug
13 2828 --              is true. }
13 2829 --
13 2830 --      PROCEDURE EntDebugger(entering: BOOLEAN);
13 2831 --
13 2832 --      FUNCTION  LookupSymbol(VAR sym: String8): LONGINT;
13 2833 --
13 2834 --      PROCEDURE NotYetImplemented(where: Str255);
13 2835 --          { Put up an alert saying Not Yet Implemented and WriteLn where. }
13 2836 --
13 2837 --      PROCEDURE UseTempRgn(byWhom: Str255);
13 2838 --          { Call this when you are about to use gTempRgn and qDebug is true.  Used
13 2839 --              with DoneWithTempRgn will prevent you from trying to use gTempRgn
13 2840 --              from two places at the same time. }
13 2841 --      {$ENDC qDebug}
13 2842 --
13 2843 --
13 2844 --
13 2845 --      { W I N D O W S }
13 2846 --
13 2847 --      PROCEDURE FreeWmgrWindow(w: WindowPtr; dispose: BOOLEAN);
13 2848 --          { Calls DisposeWindow if dispose is true, else calls CloseWindow. }
13 2849 --
13 2850 --      {$IFC qProceduralViews}
13 2851 --      FUNCTION  NewPaletteWindow(itsRsrcID: INTEGER;
13 2852 --              wantHScrollBar, wantVScrollBar: BOOLEAN; itsDocument: TDocument;
13 2853 --              itsMainView: TView;
13 2854 --              itsPaletteView: TView; sizePalette: INTEGER; whichWay: VHSelect): TWindow;
13 2855 --          { Utility for creating MacDraw-line windows with 1 non-scrolling palette
13 2856 --              along the left edge (whichWay = h), or a non-scrolling status area at
13 2857 --              the top of the window (whichWay = v), and a main view that may or may
13 2858 --              not scroll. Signals Failure if the window could not be created. }
13 2859 --
13 2860 --      FUNCTION  NewSimpleWindow(itsRsrcID: INTEGER; wantHScrollBar, wantVScrollBar: BOOLEAN;
13 2861 --                              itsDocument: TDocument; itsView: TView): TWindow;
13 2862 --          { Utility for creating simple windows that contain 1 view and may or may not
13 2863 --              scroll.  Signals Failure if the window could not be created. }
13 2864 --
13 2865 --      FUNCTION  NewTWindow(itsRsrcID: INTEGER; itsDocument: TDocument): TWindow;
13 2866 --          { Utility for creating a window object.  Used within NewPaletteWindow &
13 2867 --              NewSimpleWindow.  SIgnals Failure if the window could not be created. }
13 2868 --      {$ENDC}
13 2869 --
13 2870 --      {$IFC qTemplateViews}
```

```
13 2871 --      FUNCTION  NewTemplateWindow (viewRsrcID: INTEGER; itsDocument: TDocument): TWindow;
13 2872 --          { Creates a window using a 'WIND' resource and a 'view' template. }
13 2873 --      {$ENDC}
13 2874 --
13 2875 --      FUNCTION  ParseTitleTemplate(VAR template: Str255;
13 2876 --              VAR preDocname, constTitle: INTEGER): BOOLEAN;
13 2877 --          { Used in TWindow.IWindow to parse the template for window titles.  Returns
13 2878 --              TRUE if it changed the template.  preDocname is start of document name
13 2879 --              in string; constChars is total # characters of window title that are
13 2880 --              constants.  (These values are fixed regardless of the document name.)
13 2881 --              If preDocname is 0, then the entire title is constant. }
13 2882 --
13 2883 --      FUNCTION  SubstituteInTitle(VAR title: Str255; newStuff: Str255;
13 2884 --              preDocname, constTitle: INTEGER): BOOLEAN;
13 2885 --          { Substitutes newStuff into the template as parsed by ParseTitleTemplate;
13 2886 --              returns TRUE if a substitution was made. }
13 2887 --
13 2888 --
13 2889 --      { C L I P B O A R D }
13 2890 --
13 2891 --      PROCEDURE CanPaste(aClipType: ResType);
13 2892 --          { Call this in your SetUpMenus code to register an ability to paste a
13 2893 --              particular type of Clipboard data }
13 2894 --
13 2895 --      FUNCTION PutDeskScrapData(aResType: ResType; aDataHandle: Handle): OSErr;
13 2896 --          { Call this from your TCommand method 'WriteToDeskScrap' (for a Command which
13 2897 --              changes the Clipboard) to write out data to the actual Desk Scrap.  The
13 2898 --              return code from the Scrap Manager is returned as the function value --
13 2899 --              will be noErr unless something went wrong.  This procedure leaves
13 2900 --              aDataHandle UNLOCKED.  Rather than calling this, you can call the
13 2901 --              ToolBox routine PutScrap yourself }
13 2902 --
13 2903 --
13 2904 --      { E R R O R S }
13 2905 --      PROCEDURE ErrorAlert(err: OSErr; message: LONGINT);
13 2906 --          { Displays an error alert box.  The message displayed in the alert is
13 2907 --              constructed base upon err and message, as described in the "Failure
13 2908 --              Handling" section of the "Cookbook" chapter of the MacApp manual. }
13 2909 --
13 2910 --      FUNCTION  LookupErrString(value: INTEGER; resourceID: INTEGER;
13 2911 --                                      VAR str: Str255): BOOLEAN;
13 2912 --          { Looks up an error value in an 'errs' resource and returns TRUE
13 2913 --              if found.  resourceID should be that of a MacApp errs resource;
13 2914 --              we do the lookup first in errAppTable+resourceID, which applications
13 2915 --              can use to override or extend the MacApp table.  If the value is not
13 2916 --              found, str will be empty. }
13 2917 --
13 2918 --      FUNCTION  MacAppAlert(alertID: INTEGER; filterProc: ProcPtr): INTEGER;
13 2919 --          { Does SetCursor(arrow) before calling Alert. }
13 2920 --
13 2921 --      PROCEDURE StdAlert(alertId: INTEGER);
13 2922 --          { no filterProc, reply ignored }
13 2923 --
13 2924 --
13 2925 --      { V I E W   R E S O U R C E   U T I L I T I E S }
13 2926 --
13 2927 --      FUNCTION  NewViewRsrc (VAR p: UNIV Ptr): ViewRsrcHndl;
13 2928 --          { Convenience routine to create a new view resource template handle.  Passes back
13 2929 --              in "p" the pointer to the first template entry. }
13 2930 --
13 2931 --      PROCEDURE DoneViewRsrc (viewRsrc: UNIV Handle; lastPtr: UNIV LONGINT);
13 2932 --          { Cuts back the handle to the proper length, based on lastPtr which should be the
13 2933 --              address of the next template entry. }
13 2934 --
13 2935 --      FUNCTION  ExpandPtr (viewRsrc: UNIV Handle; VAR p: UNIV LONGINT;
13 2936 --                                      offset: LONGINT): Ptr;
13 2937 --          { Bumps size of view resource if necessary to add a new template.  Amount to bump is set
13 2938 --              to at least kViewRsrcExpandAmt over the current size.  "p" is offset to point to the
13 2939 --              next available position for a new template.
13 2940 --            Returns the old value of p before it was updated. }
13 2941 --
13 2942 --      FUNCTION  ExpandPtrWStr (viewRsrc: UNIV Handle; VAR p: UNIV LONGINT;
13 2943 --                                      offset, len: LONGINT): Ptr;
13 2944 --          { Bumps size of view resource if necessary to add a new template which ends with a
13 2945 --              variable length string, the length of which is passed in len.
13 2946 --            Returns the old value of p before it was updated. }
13 2947 --
13 2948 --      PROCEDURE OffsetPtr (VAR p: UNIV LONGINT; offset: LONGINT);
13 2949 --          { Adds offset to p, forcing word alignment.}
13 2950 --
13 2951 --      PROCEDURE OffsetPtrWStr (VAR p: UNIV LONGINT; offset: LONGINT);
13 2952 --          { Adds offset to p, taking length of trailing variable length string into account,
13 2953 --              forcing word alignment.}
13 2954 --
13 2955 --              ,
13 2956 --      { M I S C E L L A N E O U S }
13 2957 --
13 2958 --      PROCEDURE ExitMacApp;
13 2959 --          { Call this if for some reason you want to immediately exit the application.  It calls
13 2960 --              gApplication.Terminate, cleans up some other internal stuff, and then calls ExitToShell.
13 2961 --              (Normally, you would not call this, because MacApp takes care of terminating
13 2962 --              the application.)
13 2963 --
13 2964 --              You must have first called InitToolbox before calling this. }
13 2965 --
13 2966 --      PROCEDURE InitToolbox(callsToMoreMasters: INTEGER);
13 2967 --          { Call this the very first thing in your main program.  Does the essential
13 2968 --              Toolbox initialization;  If you also use the printing unit UPrinting,
13 2969 --              call InitPrinting just after you call InitToolbox }
```

```
13 2970 --
13 2971 --      FUNCTION MakeNewRgn: RgnHandle;
13 2972 --          { Calls NewRgn, then FailNIL}
13 2973 --
13 2974 --      (*  Removed because it has bugs and may not be the right solution
13 2975 --      PROCEDURE PrepareForDialog;
13 2976 --          { Deactivates the Front Window, in preparation for a coming Modal Dialog }
13 2977 --      *)
13 2978 --
13 2979 --      FUNCTION  RectIsVisible(r: Rect): BOOLEAN;
13 2980 --          { Determine if a rect is visible in the current grafport. }
13 2981 --
13 2982 --      PROCEDURE SetHLPenState(fromHL, toHL: HLState);
13 2983 --          { Set the pen state for highlighting by XOR, given the highlight transition. You can use
13 2984 --              this if you use Paint<X> or Frame<X> or line drawing for your highlighting. }
13 2985 --          { ??? Add a pattern parameter or make a helper routine that has a pattern parameter ??? }
13 2986 --
13 2987 --      {$IFC qTemplateViews}
13 2988 --      PROCEDURE RegisterType (typeName: Str255; protoObj: TObject);
13 2989 --          { Register the given class name with an object of that class. }
13 2990 --
13 2991 --      PROCEDURE RegisterStdType (typeName: Str255; signature: IDType; protoObj: TObject);
13 2992 --          { Register the given class name and 'view' signature with an object of that class. }
13 2993 --
13 2994 --      FUNCTION FindStdView (signature: IDType): TView;
13 2995 --          { Returns the prototype view for the given signature. }
13 2996 --      {$ENDC}
13 2997 --
13 2998 --
13 2999 --      IMPLEMENTATION
13 3000 --
13 3001 --      {$I UMacApp.Globals.p}
```

```
14    1 --      {$P}
14    2 --      { UMacApp.Globals.p }
14    3 --      { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
14    4 --
14    5 --      {-----------------------------------+
14    6 --      |   Local Constants                 |
14    7 --      +-----------------------------------}
14    8 --      CONST
14    9 --
14   10 --          kMaxIdleTime =  MAXLONGINT;             { Default tick count sent to WaitNextEvent }
14   11 --
14   12 --          { Numbers to pass to SystemEdit }
14   13 --              kSysUndo      =   0;
14   14 --              kSysCut       =   2;
14   15 --              kSysCopy      =   3;
14   16 --              kSysPaste     =   4;
14   17 --              kSysClear     =   5;
14   18 --      {$IFC qWWNeeded}
14   19 --              kDbgWWHeight = 100;
14   20 --      {$ENDC}
14   21 --
14   22 --      {$IFC qDebug}
14   23 --              kRsrcCheckInterval = 100;
14   24 --      {$ENDC}
14   25 --
14   26 --
14   27 --      {-----------------------------------+
14   28 --      |   Local Types                     |
14   29 --      +-----------------------------------}
14   30 --      TYPE
14   31 --          dbugParams =    RECORD                  { Format of 'dbug' resource                 }
14   32 --                              boundsRect:     Rect;       { Rect of debugging window          }
14   33 --                              fontNumber:     INTEGER;    { Font rsrc ID                      }
14   34 --                              fontSize:       INTEGER;    { Font size                         }
14   35 --                              numLines:       INTEGER;    { Number of lines                   }
14   36 --                              lineWidth:      INTEGER;    { Line width                        }
14   37 --                              title:          Str255;     { Actually, variable length         }
14   38 --                          END;
14   39 --          dbugParamsPtr =     ^dbugParams;
14   40 --          dbugParamsHndl =    ^dbugParamsPtr;
14   41 --
14   42 --      {-----------------------------------+
14   43 --      |   Local Variables                 |
14   44 --      +-----------------------------------}
14   45 --      VAR
14   46 --              { MacApp private global variables, in alphabetical order}
14   47 --          gCopyright:         StringHandle;
14   48 --          gETSPatch:          TrapPatch;      { patch for ExitToShell }
14   49 --
14   50 --
14   51 --      (*****************************************************************************************)
14   52 --      (*      External and Forward declared procedures                                         *)
14   53 --      (*****************************************************************************************)
14   54 -- A    PROCEDURE Init2; FORWARD;
14   55 -- A    PROCEDURE Init3; FORWARD;
14   56 --
14   57 --      {$IFC qInspector}
14   58 -- A    PROCEDURE InitUInspector; EXTERNAL;
14   59 -- A    PROCEDURE MakeInspector; EXTERNAL;
14   60 -- A    PROCEDURE MakeInspectorWindow; EXTERNAL;
14   61 -- A    PROCEDURE AddObjectToInspector (theObject: TObject); EXTERNAL;
14   62 --      {$ENDC}
14   63 --
14   64 --
14   65 --      (*****************************************************************************************)
14   66 --      (*      Regular procedures (in alphabetical order)                                       *)
14   67 --      (*****************************************************************************************)
14   68 --
14   69 --      {$S MARes}
14   70 --      {-----------------------------------+
14   71 --      |   CanPaste                        |
14   72 --      +-----------------------------------}
14   73 -- A    PROCEDURE CanPaste(aClipType: ResType);
14   74 0- A   BEGIN
14   75 --          IF gClipView <> NIL THEN
14   76 --              IF gClipView.ContainsClipType(aClipType) THEN
14   77 1-                BEGIN
14   78 --                gGotClipType := TRUE;
14   79 --                gPrefClipType := aClipType;
14   80 -1                END;
14   81 -0 A   END;
14   82 --
14   83 --      {$S Main}
14   84 --      {-----------------------------------+
14   85 --      |   CleanupMacApp                   |
14   86 --      +-----------------------------------}
14   87 -- A    PROCEDURE CleanupMacApp;
14   88 --      VAR
14   89 --          dontCare:       BOOLEAN;
14   90 0- A   BEGIN
14   91 --          SetupA5;        { ***** Called from trap patches *****}
14   92 --
14   93 --          UnpatchTrap(gETSPatch);
14   94 --
14   95 --          IF gApplication <> NIL THEN
14   96 --              gApplication.Terminate;
14   97 --
14   98 --          BusyRemove;
```

```
14   99 --
14  100 --          UnpatchAll;
14  101 --
14  102 --          {$IFC qWWNeeded}
14  103 --          IF WWRedirect(0, '') <> noErr THEN { ??? }; { close redirect file }
14  104 --          {$ENDC}
14  105 --
14  106 --          {$IFC qTrace}
14  107 --          TRCTerminate;
14  108 --          {$ENDC}
14  109 --
14  110 --          SetResLoad(TRUE);
14  111 --
14  112 --          dontCare := SetChooserAlert(gOldChooserFlag);
14  113 --
14  114 --          RestoreA5;
14  115 -0 A    END;
14  116 --
14  117 --      {$IFC qDebug}
14  118 --      {$S MADebug}
14  119 --          { See comment for UseTempRgn above. }
14  120 --      {$IFC qTrace}{$D+}{$ENDC}
14  121 --      {----------------------------------+
14  122 --      |   DoneWithTempRgn                 |
14  123 --      +----------------------------------}
14  124 -- A    PROCEDURE DoneWithTempRgn;
14  125 --          { Indicates that gTempRgn is no longer in use.  Call this only if qDebug
14  126 --              is true. }
14  127 --
14  128 0- A    BEGIN
14  129 --          IF NOT gBusyTempRgn THEN
14  130 --              ProgramBreak('DoneWithTempRgn called, but gTempRgn is not locked');
14  131 --          gBusyTempRgn := FALSE;
14  132 --          gUsedBy := '';
14  133 --          SetEmptyRgn(gTempRgn);
14  134 -0 A    END { DoneWithTempRgn };
14  135 --      {$IFC qTrace}{$D++}{$ENDC}
14  136 --      {$ENDC}
14  137 --
14  138 --      {$IFC qDebug}
14  139 --      {$S MADebug}
14  140 --      {$IFC qTrace}{$D+}{$ENDC}
14  141 --      {----------------------------------+
14  142 --      |   EntDebugger                     |
14  143 --      +----------------------------------}
14  144 -- A    PROCEDURE EntDebugger(entering: BOOLEAN);
14  145 0- A    BEGIN
14  146 --          BusyActivate(NOT entering);
14  147 -0 A    END;
14  148 --      {$IFC qTrace}{$D++}{$ENDC}
14  149 --      {$ENDC}
14  150 --
14  151 --      {$S MAError}
14  152 --      {----------------------------------+
14  153 --      |   ErrorAlert                      |
14  154 --      +----------------------------------}
14  155 -- A    PROCEDURE ErrorAlert(err: OSErr; message: LONGINT);
14  156 --          CONST
14  157 --              kMsgCmdErr      =       msgCmdErr DIV $10000;
14  158 --              kMsgAlert       =       msgAlert DIV $10000;
14  159 --              kMsgLookup      =       msgLookup DIV $10000;
14  160 --              kMsgAltRecov    =       msgAltRecovery DIV $10000;
14  161 --
14  162 --          TYPE
14  163 --              Converter = RECORD
14  164 --                              CASE BOOLEAN OF
14  165 --                                  TRUE:   (message: LONGINT);
14  166 --                                  FALSE:  (hiWd, loWd: INTEGER);
14  167 --                                  END;
14  168 --
14  169 --          VAR c:              Converter;
14  170 --              alertID:        INTEGER;
14  171 --              genericAlert:   BOOLEAN;
14  172 --              opString:       Str255;
14  173 --              errStr:         Str255;
14  174 --              recovErr:       OSErr;
14  175 --              recovery:       Str255;
14  176 --              x:              BOOLEAN;
14  177 --
14  178 0- A    BEGIN
14  179 --          c.message := message;
14  180 --
14  181 --          alertID := phGenError;    { the default alert }
14  182 --          genericAlert := TRUE;
14  183 --          opString := '';
14  184 --
14  185 1-          CASE c.hiWd OF
14  186 --              kMsgCmdErr:
14  187 2-                  BEGIN
14  188 --                  alertID := phCmdErr;
14  189 --                  CmdToName(c.loWd, opString);
14  190 -2                  END;
14  191 --              kMsgAlert:
14  192 2-                  BEGIN
14  193 --                  alertID := c.loWd;
14  194 --                  genericAlert := FALSE;
14  195 -2                  END;
14  196 --              kMsgLookup, kMsgAltRecov:
14  197 2-                  BEGIN
```

```
14  198 --                x := LookupErrString(c.loWd, errOperationsID, opString);
14  199 -2               END;
14  200 --           OTHERWISE
14  201 2-               BEGIN
14  202 --               GetIndString(opString, c.hiWd, c.loWd);
14  203 -2               END;
14  204 -1           END;
14  205 --
14  206 --       IF genericAlert THEN
14  207 1-           BEGIN
14  208 --           x := LookupErrString(err, errReasonID, errStr);
14  209 --
14  210 --           IF c.hiWd = kMsgAltRecov THEN
14  211 --               recovErr := c.loWd
14  212 --           ELSE
14  213 --               recovErr := err;
14  214 --
14  215 --           x := LookupErrString(recovErr, errRecoveryID, recovery);
14  216 --
14  217 --           ParamText(errStr, recovery, opString, gErrorParm3);
14  218 --
14  219 --           IF opString = '' THEN
14  220 --               alertID := phUnknownErr;
14  221 -1           END;
14  222 --
14  223 --       StdAlert(alertID);
14  224 --
14  225 --       IF genericAlert THEN
14  226 --           ResetAlrtStage;
14  227 -0 A   END;
14  228 --
14  229 --    {$S MATerminate}
14  230 --    {-----------------------------------+
14  231 --    |   ExitMacApp                      |
14  232 --    +-----------------------------------}
14  233 -- A   PROCEDURE ExitMacApp;
14  234 0- A   BEGIN
14  235 --       CleanupMacApp;
14  236 --       ExitToShell;
14  237 -0 A   END;
14  238 --
14  239 --    {$S MAFinder}
14  240 --    { This is a dummy procedure to allow us to find the Finder segment }
14  241 --    {-----------------------------------+
14  242 --    |   FinderSegProc                   |
14  243 --    +-----------------------------------}
14  244 -- A   PROCEDURE FinderSegProc;
14  245 0- A   BEGIN
14  246 -0 A   END;
14  247 --
14  248 --    {$IFC qTemplateViews}
14  249 --    {$S MAOpen}
14  250 --    {-----------------------------------+
14  251 --    |   FindStdView                     |
14  252 --    +-----------------------------------}
14  253 -- A   FUNCTION FindStdView (signature: IDType): TView;
14  254 --
14  255 --    VAR
14  256 --        i:          INTEGER;
14  257 --
14  258 0- A   BEGIN
14  259 --        FOR i := 1 TO gSignatureCount DO
14  260 --            IF LONGINT(gSignatures[i]) = LONGINT(signature) THEN
14  261 1-                BEGIN
14  262 --                FindStdView := TView(gSignatureViews[i]);
14  263 --                EXIT(FindStdView);
14  264 -1                END;
14  265 --        FindStdView := NIL;
14  266 -0 A   END;
14  267 --    {$ENDC}
14  268 --
14  269 --    {$S MAMain}
14  270 --    {-----------------------------------+
14  271 --    |   FreeWmgrWindow                  |
14  272 --    +-----------------------------------}
14  273 -- A   PROCEDURE FreeWmgrWindow(w: WindowPtr; dispose: BOOLEAN);
14  274 --        VAR theItemList:    Handle;
14  275 0- A   BEGIN
14  276 --        IF dispose THEN
14  277 --            DisposeWindow(w)
14  278 --        ELSE
14  279 --            CloseWindow(w);
14  280 -0 A   END;
14  281 --
14  282 --    {$S MARes}
14  283 --    {-----------------------------------+
14  284 --    |   GetFocus                        |
14  285 --    +-----------------------------------}
14  286 -- A   PROCEDURE GetFocus;
14  287 --
14  288 0- A   BEGIN
14  289 --        GetPort(gSavePort);
14  290 --        gSaveOrg := thePort^.portRect.topLeft;
14  291 --        gSaveLongOffset := gLongOffset;
14  292 --        GetClip(gSaveClip);
14  293 --        gSaveFocusedView := gFocusedView;
14  294 -0 A   END { GetFocus };
14  295 --
14  296 --    {$S Main} .     { Must be in a resident segment to ensure that UnloadAllSegments doesn't unload it. }
```

```
14  297 --      {-----------------------------------+
14  298 --      |   InitFailed                      |
14  299 --      +-----------------------------------}
14  300 -- A    PROCEDURE InitFailed(error: OSErr; message: LONGINT);
14  301 0- A    BEGIN
14  302 --          UnloadAllSegments;
14  303 --
14  304 --          IF error <> noErr THEN { check to see if an alert has already been displayed }
14  305 1-              BEGIN
14  306 --              IF message = 0 THEN
14  307 --                  message := msgInitFailed; { if no message specified, use our own }
14  308 --
14  309 --              ErrorAlert(error, message);
14  310 --
14  311 --              ExitToShell;
14  312 -1              END;
14  313 -0 A    END;
14  314 --
14  315 --      {$%+}
14  316 -- A        PROCEDURE %_InitObj; EXTERNAL;
14  317 --      {$%-}
14  318 --
14  319 --      {$S Main} { Always put this in the Main segment.}
14  320 --          { Essential Toolbox initialization & other one-time initialization }
14  321 --      {$IFC qTrace}{$D+}{$ENDC}
14  322 --      {-----------------------------------+
14  323 --      |   InitToolbox                     |
14  324 --      +-----------------------------------}
14  325 -- A    PROCEDURE InitToolbox(callsToMoreMasters: INTEGER);
14  326 --
14  327 --      CONST
14  328 --          nameSelProcs = '%_SelProcs';        { Name of Selector procs segment (CASE SENSITIVE) }
14  329 --
14  330 --      VAR
14  331 --          h:              Handle;
14  332 --          i:              INTEGER;
14  333 --          theType:        ResType;
14  334 --          name:           Str255;
14  335 --          initSeg:        INTEGER;
14  336 --          applZone:       THz;
14  337 --          oldMoreMast:    INTEGER;
14  338 --
14  339 0- A    BEGIN
14  340 --          Init2;
14  341 --              { initalize Toolbox routines & WritelnWindow; do necessary set up for
14  342 --              calling InitFailed; compute required memory and stack size; set up
14  343 --              segment loader patch & expand heap }
14  344 --
14  345 --          IF NOT InitUMemory THEN
14  346 --              InitFailed(memFullErr, 0);
14  347 --
14  348 --          UnloadAllSegments;
14  349 --          LoadResidentSegments;
14  350 --
14  351 --      {$%+}
14  352 --          { Must call %_InitObj to load method table segment and initialize stuff }
14  353 --           %_InitObj;
14  354 --      {$%-}
14  355 --
14  356 --          { Force the init segment to be memory resident, so we can call UnloadAllSegs during init }
14  357 --          initSeg := GetSegNumber(@Init3);
14  358 --          SetResidentSegment(initSeg, TRUE);
14  359 --
14  360 --      { Here is a trick sugested by Jerome C.--it allocates one large block of master pointers}
14  361 --          applZone := ApplicZone;
14  362 --          oldMoreMast := applZone^.moreMast;
14  363 --          applZone^.moreMast := oldMoreMast * callsToMoreMasters;
14  364 --          MoreMasters;
14  365 --          applZone^.moreMast := oldMoreMast;
14  366 --
14  367 --          Init3; { do rest of initialization }
14  368 --
14  369 --          SetResidentSegment(initSeg, FALSE); { make it non-resident }
14  370 --          UnloadAllSegments;
14  371 -0 A    END;
14  372 --
14  373 --      {$S MAInit} { Must be in MAInit }
14  374 --      {-----------------------------------+
14  375 --      |   Init2                           |
14  376 --      +-----------------------------------}
14  377 -- A    PROCEDURE Init2;
14  378 --      VAR
14  379 --          dParams:        Handle;
14  380 --
14  381 0- A    BEGIN
14  382 --          gInitialized := FALSE; { will be set TRUE in IApplication }
14  383 --
14  384 --          InitGraf(@thePort);
14  385 --          InitFonts;
14  386 --          InitWindows; { creates a non-relocatable for the WM port }
14  387 --          FlushEvents(everyEvent - diskMask, 0);
14  388 --          InitMenus;
14  389 --          TEInit;
14  390 --          InitDialogs(NIL);
14  391 --          InitCursor;
14  392 --
14  393 --      {$IFC qDebug}
14  394 --          gDebugWindowPtr := NIL;                 { Just in case someone tries to use }
14  395 --                                                  { it before its been allocated.     }
```

```
14  396 --          gFieldToStrRtn := @StdFieldToString;    { Get this set up early.           }
14  397 --      {$ENDC}
14  398 --
14  399 --          DefineConfiguration(gConfiguration);
14  400 --      {$IFC qNeedsROM128K}
14  401 --          { If we require 128K ROMs then make sure we are in fact running on them,
14  402 --              and if we are not, display an alert and exit the app }
14  403 --
14  404 --          IF NOT gConfiguration.hasROM128K THEN
14  405 1-              BEGIN
14  406 --              StdAlert(phUnsupportedROMs);
14  407 --              ExitToShell;
14  408 -1              END;
14  409 --      {$ENDC}
14  410 --
14  411 --          InitUPatch;
14  412 --          InitUFailure;
14  413 --
14  414 --          SetRGBColor(gRGBBlack, 0, 0, 0);    { Needs to come before InitUInspector }
14  415 --          SetRGBColor(gRGBWhite, $FFFF, $FFFF, $FFFF);
14  416 --
14  417 --      {$IFC qInspector}
14  418 --          InitUInspector;
14  419 --      {$ENDC}
14  420 --
14  421 --      {$IFC FALSE}
14  422 --          IF gConfiguration.hasROM128K THEN
14  423 --              SetFractEnable(TRUE);
14  424 --      {$ENDC}
14  425 --
14  426 --      {$IFC qDebug}
14  427 --          BuildObjNameTable;
14  428 --          gExperimenting := FALSE;
14  429 --          gDebugPrinting := FALSE;
14  430 --          gReportMenuChoices := FALSE;
14  431 --          gIntenseDebugging := FALSE;
14  432 --          gReportEvt := FALSE;
14  433 --          gMastReport := FALSE;
14  434 --          gRsrcReport := FALSE;
14  435 --          gMemMgtBreak := FALSE;
14  436 --      {$ENDC}
14  437 --
14  438 --      {$IFC qWWNeeded}
14  439 --          dParams := GetResource('dbug', kDebugParamsID);
14  440 --          IF dParams <> NIL THEN
14  441 --              WITH dbugParamsHndl(dParams)^^ DO
14  442 --                  WWInit(numLines, lineWidth)
14  443 --          ELSE
14  444 --              WWInit(52, 80); { 52 lines of 80 characters }
14  445 --      {$ENDC}
14  446 --
14  447 --      {$IFC qDebug}
14  448 --          TRCInit;
14  449 --      {$ENDC}
14  450 --
14  451 --          SetInitHandler(@InitFailed);
14  452 --
14  453 --          { the following set up is necessary to call CleanupMacApp }
14  454 --          gApplication := NIL;
14  455 --
14  456 --          FailOsErr(TailPatch(gETSPatch, $A9F4, @CleanupMacApp));     { ExitToShell }
14  457 --
14  458 -0 A    END;
14  459 --
14  460 --      {$S MAInit} { Must be in the init segment; unloaded at start of event loop }
14  461 --          { Essential Toolbox initialization & other one-time initialization; part 2 }
14  462 --      {$IFC qTrace}{$D+}{$ENDC}
14  463 --      {-----------------------------------+
14  464 --      |   Init3                           |
14  465 --      +-----------------------------------}
14  466 -- A    PROCEDURE Init3;
14  467 --
14  468 --      CONST
14  469 --          MBarHeight =    $BAA;
14  470 --          SysFontSize =   $BA8;
14  471 --
14  472 --      VAR
14  473 --          message:        INTEGER;
14  474 --          aScrollBar:     TSScrollBar;
14  475 --          aScroller:      TScroller;
14  476 --          aWindow:        TWindow;
14  477 --          aDeskScrapView: TDeskScrapView;
14  478 --      {$IFC qWWNeeded}
14  479 --          outerRect:      Rect;  { ??? }
14  480 --          dParams:        Handle;
14  481 --      {$ENDC}
14  482 --      {$IFC qDebug}
14  483 --          gDebugKeyMap:   KeyMap;     { the key state at start-up time }
14  484 --      {$ENDC}
14  485 --          fontSize:       INTEGER;
14  486 --
14  487 0- A    BEGIN
14  488 --
14  489 --          BusyInstall;
14  490 --
14  491 --      {$IFC qTrace}
14  492 --          gTraceSetupMenus := FALSE;
14  493 --          gTraceIdle := FALSE;
14  494 --          TRCInstall(Handle(gCodeSegs), Handle(gNonResidents),
```

```
14  495 --                          @EntDebugger, @InspectObject, @LookupSymbol);
14  496 --
14  497 --          TRCFlag(@gAskAboutAlloc,        'A', 'ask about allocs');
14  498 --          TRCFlag(@gMemMgtBreak,          'B', 'mem mgt. break');
14  499 --          TRCFlag(@gReportMenuChoices,    'C', 'report menu commands');
14  500 --          TRCFlag(@gReportEvt,            'E', 'report events');
14  501 --          TRCFlag(@gAskFailure,           'F', 'ask about failures');
14  502 --          TRCFlag(@gIntenseDebugging,     'I', 'intense debugging');
14  503 --          TRCFlag(@gMastReport,           'M', 'report # masters');
14  504 --          TRCFlag(@gDebugPrinting,        'P', 'printing debug');
14  505 --          TRCFlag(@gRsrcReport,           'R', 'report rsrc usage');
14  506 --          TRCFlag(@gSegReport,            'S', 'report seg load');
14  507 --          TRCFlag(@gUnloadAllSegs,        'U', 'unload segments');
14  508 --          TRCFlag(@gExperimenting,        'X', 'experimenting');
14  509 --      {$ENDC qTrace}
14  510 --
14  511 --          gMainEventMask := everyEvent;
14  512 --
14  513 --      {$IFC qWWNeeded}
14  514 --          dParams := GetResource('dbug', kDebugParamsID);
14  515 --          IF dParams <> NIL THEN
14  516 1-              BEGIN
14  517 --              HNoPurge(dParams);
14  518 --              WITH dbugParamsHndl(dParams)^^ DO
14  519 --                  WWNew(boundsRect, title, TRUE, TRUE, fontNumber, fontSize);
14  520 --              HPurge(dParams);
14  521 -1              END
14  522 --          ELSE
14  523 1-              BEGIN
14  524 --              outerRect := screenBits.bounds;
14  525 --              InsetRect(outerRect, 5, 5);
14  526 --              outerRect.top := outerRect.bottom - kDbgWWHeight;
14  527 --              WWNew(outerRect, 'Debug Window', TRUE, TRUE, kDebugFont, kDebugSize);
14  528 -1              END;
14  529 --      {$ENDC}
14  530 --
14  531 --      {$IFC qTrace}
14  532 --          TRCEnable(TRUE);
14  533 --      {$ENDC}
14  534 --
14  535 --          gCopyright := NewString(kCopyright);
14  536 --
14  537 --      {$IFC qDebug}
14  538 --          gBoolString[TRUE] := 'TRUE';
14  539 --          gBoolString[FALSE] := 'FALSE';
14  540 --
14  541 --          gRsrcCheck := kRsrcCheckInterval;
14  542 --      {$ENDC}
14  543 --
14  544 --          InitUMenuSetup;
14  545 --
14  546 --          gOldChooserFlag := SetChooserAlert(FALSE);
14  547 --
14  548 --      {$IFC qDebug}
14  549 --          GetKeys(gDebugKeyMap);
14  550 --          IF gDebugKeyMap[55] AND gDebugKeyMap[58] THEN { clover-option }
14  551 --              ProgramBreak('At start of application');
14  552 --      {$ENDC}
14  553 --
14  554 --      {$IFC qDebug}
14  555 --              IF cUndo -  cEditBase <> kSysUndo THEN
14  556 --                  WriteLn('Invalid UNDO command number');
14  557 --              IF cCut -   cEditBase <> kSysCut THEN
14  558 --                  WriteLn('Invalid CUT command number');
14  559 --              IF cCopy -  cEditBase <> kSysCopy THEN
14  560 --                  WriteLn('Invalid COPY command number');
14  561 --              IF cPaste - cEditBase <> kSysPaste THEN
14  562 --                  WriteLn('Invalid PASTE command number');
14  563 --              IF cClear - cEditBase <> kSysClear THEN
14  564 --                  WriteLn('Invalid CLEAR command number');
14  565 --      {$ENDC}
14  566 --
14  567 --        { Other 1-time initialization }
14  568 --          gTempRgn := MakeNewRgn;
14  569 --          gSaveClip := MakeNewRgn;
14  570 --
14  571 --          gClickCount := 0;
14  572 --          gLastUpTime := TickCount;
14  573 --          gLastClickPart := inDesk;
14  574 --          gIdlePhase := idleEnd;
14  575 --          gInBackground := FALSE;                 { When we start an app, it's in foreground }
14  576 --          gLastDeskAcc := gLastUpTime;
14  577 --          gIdleFreq := 0;
14  578 --          gLastIdle := 0;
14  579 --          gCursorRgn := MakeNewRgn;
14  580 --
14  581 --          { Create a work port for our convenience }
14  582 --          gWorkPort := @gFakeWindow;
14  583 --          IF gConfiguration.hasColorQD THEN
14  584 --              OpenCPort(CGrafPtr(gWorkPort))
14  585 --          ELSE
14  586 --              OpenPort(gWorkPort);
14  587 --          gFakeWindow.ControlList := NIL;
14  588 --
14  589 --          gNextSpaceMsg := gLastUpTime;
14  590 --          gLowSpaceInterval := kLowSpaceInterval;
14  591 --
14  592 --      {$IFC qDebug}
14  593 --          gBusyTempRgn := FALSE;
```

```
14  594 --          gUsedBy := '';
14  595 --      {$ENDC}
14  596 --
14  597 --          gNoChanges := Pointer(-1);
14  598 --
14  599 --          gStdHysteresis := Point($00040004); { ??? any better choice ??? }
14  600 --
14  601 --          gFrontWindow := NIL;
14  602 --
14  603 --          gZeroRect.topLeft := Point(0);
14  604 --          gZeroRect.botRight := Point(0);
14  605 --          SetVPt(gZeroVPt, 0, 0);
14  606 --          SetVRect(gZeroVRect, 0, 0, 0, 0);
14  607 --
14  608 --      {$IFC NOT qNeedsROM128K}
14  609 --          IF NOT gConfiguration.hasROM128K THEN
14  610 --              gMBarHeight := 20
14  611 --          ELSE
14  612 --      {$ENDC}
14  613 --          IF gConfiguration.hasScriptManager THEN
14  614 --              gMBarHeight := GetMBarHeight
14  615 --          ELSE
14  616 --              gMBarHeight := PInteger(MBarHeight)^;
14  617 --
14  618 --          WITH screenBits.bounds DO
14  619 1-             BEGIN
14  620 --              SetRect(gStdWMoveBounds, 4, gMBarHeight + 4, right - 4, bottom - 4);
14  621 --
14  622 --              { arbitrary minimum size; maximum size is screen size minus menu bar,
14  623 --                  and half the title bar }
14  624 --              SetRect(gStdWSizeRect, 80, 80, right, bottom - gMBarHeight - 8);
14  625 --
14  626 --              SetRect(gStdWScreenRect, 16, gMBarHeight + 16, right - 16, bottom - 16);
14  627 -1             END;
14  628 --
14  629 --          gOrthogonal[v] := h;
14  630 --          gOrthogonal[h] := v;
14  631 --
14  632 --          gPrinting := FALSE;
14  633 --          gCurrPrintHandler := NIL;
14  634 --          gDrawingPictScrap := FALSE;
14  635 --
14  636 --          gFinderPrinting := FALSE;
14  637 --          gCouldPrint := FALSE;
14  638 --
14  639 --          CountAppFiles(message, gFileCount);
14  640 --          gFinderPrinting := (message = appPrint);
14  641 --
14  642 --          gHeadCohandler := NIL;
14  643 --          gEventLevel := 0;
14  644 --
14  645 --          New(gNullPrintHandler);
14  646 --          FailNil(gNullPrintHandler);
14  647 --          gNullPrintHandler.IPrintHandler(NIL);
14  648 --
14  649 --          gPrintHandler := gNullPrintHandler;
14  650 --
14  651 --          gFreeWindowList := NewList;
14  652 --
14  653 --      {$IFC qDebug}
14  654 --          gFreeWindowList.SetEltType('TWindow');
14  655 --      {$ENDC}
14  656 --
14  657 --          gChooserOK := TRUE;
14  658 --
14  659 --          gClipWindow := NIL;
14  660 --
14  661 --          gGotClipType := FALSE;
14  662 --
14  663 --          gClipView := NIL;
14  664 --          gClipUndoView := NIL;
14  665 --
14  666 --          gNumUntitled := 1; { call the first document Untitled-1 }
14  667 --
14  668 --          gUndoState := kShowUndo;
14  669 --          gUndoCmd := cNoCommand;
14  670 --          gErrorParm3 := '';
14  671 --          gFocusedView := NIL;
14  672 --          gStdStaggerCount := 0;
14  673 --
14  674 --          gMBarDisplayed := kMBarDisplayed;
14  675 --          gMBarNotDisplayed := kMBarNotDisplayed;
14  676 --          gMBarHierarchical := kMBarHierarchical;
14  677 --
14  678 --          IF gConfiguration.hasScriptManager THEN
14  679 --              fontSize := GetDefFontSize
14  680 --          ELSE
14  681 --      {$IFC NOT qNeedsROM128K}
14  682 --              IF NOT gConfiguration.hasROM128K THEN
14  683 --                  fontSize := 12
14  684 --              ELSE
14  685 --      {$ENDC}
14  686 --                  fontSize := PInteger(SysFontSize)^;
14  687 --          SetTextStyle(gSystemStyle, systemFont, [], fontSize, gRGBBlack);
14  688 --
14  689 --      {$IFC qTemplateViews}
14  690 --          gSignatureCount := 0;
14  691 --
14  692 --          NEW(gProtoList);    FailNIL(gProtoList);    gProtoList.IAssociation;
```

```
14  693 --          NEW(aWindow);       FailNIL(aWindow);       RegisterStdType('TWindow', 'wind', aWindow);
14  694 --          NEW(aScrollBar);    FailNIL(aScrollBar);    RegisterStdType('TSScrollBar', 'sbar', aScrollBar);
14  695 --          NEW(aScroller);     FailNIL(aScroller);     RegisterStdType('TScroller', 'scrl', aScroller);
14  696 --          NEW(aDeskScrapView); FailNIL(aDeskScrapView); RegisterType('TDeskScrapView', aDeskScrapView);
14  697 --      {$ENDC}
14  698 --
14  699 --      {$IFC qDebug}
14  700 --          IF gConfiguration.hasROM128K THEN Write('ROM128K  ');
14  701 --          IF gConfiguration.hasHFS THEN Write('HFS Installed  ');
14  702 --          IF gConfiguration.hasColorQD THEN Write('Has Color  ');
14  703 --          IF gConfiguration.hasWaitNextEvent THEN Write('WaitNextEvent  ');
14  704 --          Writeln;
14  705 --      {$ENDC}
14  706 --
14  707 -0 A    END;
14  708 --      {$IFC qTrace}{$D++}{$ENDC}
14  709 --
14  710 --      {$S MAError}
14  711 --      {-----------------------------------+
14  712 --      |   LookupErrString                 |
14  713 --      +-----------------------------------}
14  714 -- A    FUNCTION  LookupErrString(value: INTEGER; resourceID: INTEGER;
14  715 --                                                          VAR str: Str255): BOOLEAN;
14  716 --
14  717 --          {-------------------------------+
14  718 --          |   SearchTable                 |
14  719 --          +-------------------------------}
14  720 -- B        FUNCTION  SearchTable(value: INTEGER; resourceID: INTEGER;
14  721 --                                                      VAR str: Str255): BOOLEAN;
14  722 --              LABEL 1;
14  723 --
14  724 --              TYPE    PErrRecord = ^ErrRecord;
14  725 --                      ErrRecord = RECORD
14  726 --                          lowErr, highErr, index: INTEGER;
14  727 --                          END;
14  728 --
14  729 --              VAR table:          Handle;
14  730 --                  pEntry:         PErrRecord;
14  731 --                  tableOffset:    LONGINT;
14  732 --                  lenTab:         INTEGER;
14  733 --                  strID:          INTEGER;
14  734 --                  i:              INTEGER;
14  735 0- B          BEGIN
14  736 --              SearchTable := FALSE;
14  737 --              str := '';
14  738 --
14  739 --              table := GetResource('errs', resourceID);
14  740 --              IF table <> NIL THEN
14  741 1-                BEGIN
14  742 --                lenTab := GetHandleSize(Handle(table)) DIV SIZEOF(ErrRecord);
14  743 --
14  744 --                strID := 0;
14  745 --                tableOffset := 0;
14  746 --
14  747 --                FOR i := 1 TO lenTab DO
14  748 2-                  BEGIN
14  749 --                  pEntry := PErrRecord(Ord4(table^) + tableOffset);
14  750 --
14  751 --                  WITH pEntry^ DO
14  752 3-                    BEGIN
14  753 --                    IF lowErr = 0 THEN
14  754 --                      strID := index
14  755 --                    ELSE IF (lowErr <= value) & (value <= highErr) THEN
14  756 4-                      BEGIN
14  757 --                      IF index > 0 THEN
14  758 --                        GetIndString(str, strID, index);
14  759 --                      SearchTable := TRUE;
14  760 --                      GOTO 1; { exit the loop }
14  761 -4                     END;
14  762 -3                   END;
14  763 --
14  764 --                  tableOffset := tableOffset + SIZEOF(ErrRecord);
14  765 -2                  END;
14  766 --          1:
14  767 -1                END;
14  768 -0 B        END;
14  769 --
14  770 0- A    BEGIN
14  771 --          IF SearchTable(value, errAppTable+resourceId, str) THEN
14  772 --              LookupErrString := TRUE
14  773 --          ELSE
14  774 --              LookupErrString := SearchTable(value, resourceId, str);
14  775 -0 A    END;
14  776 --
14  777 --      {$IFC qDebug}
14  778 --      {$S MADebug}
14  779 --      {$IFC qTrace}{$D+}{$ENDC}
14  780 --      {-----------------------------------+
14  781 --      |   LookupSymbol                    |
14  782 --      +-----------------------------------}
14  783 -- A    FUNCTION  LookupSymbol(VAR sym: String8): LONGINT;
14  784 0- A    BEGIN
14  785 --          IF gInitialized THEN
14  786 --              LookupSymbol := gTarget.LookupSymbol(sym)
14  787 --          ELSE
14  788 --              LookupSymbol := -1;
14  789 -0 A    END;
14  790 --      {$IFC qTrace}{$D++}{$ENDC}
14  791 --      {$ENDC}
```

```
14  792 --
14  793 --      {$S MAMain}
14  794 --      {-----------------------------------+
14  795 --      |   MacAppAlert                     |
14  796 --      +-----------------------------------}
14  797 -- A    FUNCTION  MacAppAlert(alertID: INTEGER; filterProc: ProcPtr): INTEGER;
14  798 0- A    BEGIN
14  799 --          {$IFC qDebug}
14  800 --          gRsrcCheck := 0;                        { force immediate check.          }
14  801 --          {$ENDC}
14  802 --
14  803 --          SetCursor(arrow);
14  804 --          MacAppAlert := Alert(alertID, filterProc);
14  805 --          SetEmptyRgn(gCursorRgn);                { Make sure cursor get reset.     }
14  806 -0 A    END;
14  807 --
14  808 --
14  809 --      {$S MAMain}
14  810 --      {$IFC qTrace}{$D+}{$ENDC}
14  811 --      {-----------------------------------+
14  812 --      |   MakeNewRgn                      |
14  813 --      +-----------------------------------}
14  814 -- A    FUNCTION MakeNewRgn: RgnHandle;
14  815 --      VAR
14  816 --          aRgn:       RgnHandle;
14  817 0- A    BEGIN
14  818 --          aRgn := NewRgn;
14  819 --          FailNIL(aRgn);
14  820 --          MakeNewRgn := aRgn;
14  821 -0 A    END;
14  822 --      {$IFC qTrace}{$D++}{$ENDC}
14  823 --
14  824 --
14  825 --      {$IFC qProceduralViews}
14  826 --      {$S MAOpen}
14  827 --      {-----------------------------------+
14  828 --      |   NewPaletteWindow                |
14  829 --      +-----------------------------------}
14  830 -- A    FUNCTION  NewPaletteWindow(itsRsrcID: INTEGER; wantHScrollBar, wantVScrollBar: BOOLEAN;
14  831 --                                itsDocument: TDocument; itsMainView: TView; itsPaletteView: TView;
14  832 --                                sizePalette: INTEGER; whichWay: VHSelect): TWindow;
14  833 --          VAR aWindow:            TWindow;
14  834 --              aScroller:          TScroller;
14  835 --              fi:                 FailInfo;
14  836 --              itsSize:            VPoint;
14  837 --              itsLocation:        VPoint;
14  838 --              wSize:              Point;
14  839 --
14  840 --          {-------------------------------+
14  841 --          |   HdlNPWindow                 |
14  842 --          +-------------------------------}
14  843 -- B        PROCEDURE HdlNPWindow(error: INTEGER; message: LONGINT);
14  844 0- B        BEGIN
14  845 --              aWindow.Free;
14  846 -0 B        END;
14  847 --
14  848 0- A    BEGIN
14  849 --          aWindow := NewTWindow(itsRsrcID, itsDocument);
14  850 --
14  851 --          WITH aWindow.fResizeLimits.topLeft DO
14  852 --              vh[whichWay] := vh[whichWay] + sizePalette;
14  853 --
14  854 --          CatchFailures(fi, HdlNPWindow);
14  855 --
14  856 --          aWindow.AddSubView(itsPaletteView);
14  857 --
14  858 --          itsLocation := gZeroVPt;
14  859 --          itsLocation.vh[whichWay] := sizePalette;
14  860 --          IF wantHScrollBar OR wantVScrollBar THEN
14  861 1-              BEGIN
14  862 --              SetVPt(itsSize, aWindow.fSize.h - kSBarSizeMinus1,
14  863 --                              aWindow.fSize.v - kSBarSizeMinus1);
14  864 --              itsSize.vh[whichWay] := itsSize.vh[whichWay] - sizePalette;
14  865 --              New(aScroller);
14  866 --              FailNIL(aScroller);
14  867 --              aScroller.IScroller(aWindow, itsLocation, itsSize, sizeRelSuperView,
14  868 --                                  sizeRelSuperView, 0, 0, wantHScrollBar, wantVScrollBar);
14  869 --              aScroller.AddSubView(itsMainView);
14  870 -1              END
14  871 --          ELSE
14  872 --              aWindow.AddSubView(itsMainView);
14  873 --
14  874 --          aWindow.fTarget := itsMainView;
14  875 --
14  876 --          { make frames be the right size }
14  877 --          WITH aWindow.fWmgrWindow^.portRect DO
14  878 1-              BEGIN
14  879 --              wSize := botRight;
14  880 --              {$H-}
14  881 --              SubPt(topLeft, wSize);
14  882 --              {$H+}
14  883 -1              END;
14  884 --          aWindow.Resize(wSize.h, wSize.v, FALSE);
14  885 --
14  886 --          NewPaletteWindow := aWindow;
14  887 --
14  888 --          Success(fi);
14  889 -0 A    END;
14  890 --      {$ENDC}
```

```
14  891 --
14  892 --
14  893 --      {$IFC qProceduralViews}
14  894 --      {$S MAOpen}
14  895 --      {-----------------------------------+
14  896 --      |   NewSimpleWindow                 |
14  897 --      +-----------------------------------}
14  898 -- A    FUNCTION  NewSimpleWindow(itsRsrcID: INTEGER; wantHScrollBar, wantVScrollBar: BOOLEAN;
14  899 --                                itsDocument: TDocument; itsView: TView): TWindow;
14  900 --          VAR aWindow:            TWindow;
14  901 --              aScroller:          TScroller;
14  902 --              fi:                 FailInfo;
14  903 --              itsSize:            VPoint;
14  904 --              wSize:              Point;
14  905 --              sBarOffsets:        VRect;
14  906 --
14  907 --          {------------------------------+
14  908 --          |   HdlNSWindow                |
14  909 --          +------------------------------}
14  910 -- B        PROCEDURE HdlNSWindow(error: INTEGER; message: LONGINT);
14  911 0- B       BEGIN
14  912 --              aWindow.Free;
14  913 -0 B       END;
14  914 --
14  915 0- A    BEGIN
14  916 --          aWindow := NewTWindow(itsRsrcID, itsDocument);
14  917 --
14  918 --          aScroller := NIL;
14  919 --
14  920 --          CatchFailures(fi, HdlNSWindow);
14  921 --
14  922 --          IF wantHScrollBar | wantVScrollBar THEN
14  923 1-              BEGIN
14  924 --              sBarOffsets := gZeroVRect;
14  925 --              itsSize := aWindow.fSize;
14  926 --              IF wantHScrollBar THEN
14  927 2-                  BEGIN
14  928 --                  itsSize.v := itsSize.v - kSBarSizeMinus1;
14  929 --                  IF NOT wantVScrollBar THEN
14  930 --                      sBarOffsets.right := -kSBarSizeMinus1;
14  931 -2                  END;
14  932 --              IF wantVScrollBar THEN
14  933 2-                  BEGIN
14  934 --                  itsSize.h := itsSize.h - kSBarSizeMinus1;
14  935 --                  IF NOT wantHScrollBar THEN
14  936 --                      sBarOffsets.bottom := -kSBarSizeMinus1;
14  937 -2                  END;
14  938 --              New(aScroller);
14  939 --              FailNIL(aScroller);
14  940 --              aScroller.IScroller(aWindow, gZeroVPt, itsSize, sizeRelSuperView,
14  941 --                                  sizeRelSuperView, 0, 0, wantHScrollBar, wantVScrollBar);
14  942 --              aScroller.fSBarOffsets := sBarOffsets;
14  943 --              IF itsView <> NIL THEN
14  944 --                  aScroller.AddSubView(itsView);
14  945 -1              END
14  946 --          ELSE
14  947 --              IF itsView <> NIL THEN
14  948 --                  aWindow.AddSubView(itsView);
14  949 --
14  950 --          IF itsView <> NIL THEN
14  951 --              aWindow.fTarget := itsView;
14  952 --
14  953 --          { make sure window and subviews are the right size }
14  954 --          WITH aWindow.fWmgrWindow^.portRect DO
14  955 1-              BEGIN
14  956 --              wSize := botRight;
14  957 --              {$H-}
14  958 --              SubPt(topLeft, wSize);
14  959 --              {$H+}
14  960 -1              END;
14  961 --          aWindow.Resize(wSize.h, wSize.v, FALSE);
14  962 --
14  963 --          NewSimpleWindow := aWindow;
14  964 --
14  965 --          Success(fi);
14  966 -0 A    END;
14  967 --      {$ENDC}
14  968 --
14  969 --
14  970 --      {$IFC qProceduralViews}
14  971 --      {$S MAOpen}
14  972 --      {-----------------------------------+
14  973 --      |   NewTWindow                      |
14  974 --      +-----------------------------------}
14  975 -- A    FUNCTION  NewTWindow(itsRsrcID: INTEGER; itsDocument: TDocument): TWindow;
14  976 --          VAR aWmgrWindow:        WindowPtr;
14  977 --              aWindow:            TWindow;
14  978 --              canResize:          BOOLEAN;
14  979 --              canClose:           BOOLEAN;
14  980 --              fi:                 FailInfo;
14  981 --
14  982 --          {------------------------------+
14  983 --          |   HdlNewWObj                 |
14  984 --          +------------------------------}
14  985 -- B        PROCEDURE HdlNewWObj(error: INTEGER; message: LONGINT);
14  986 0- B       BEGIN
14  987 --              { the wmgrWindow is known to exist }
14  988 --              { Since aWindow didn't get created, the wmgrWindow won't be
14  989 --                  freed unless we do it here. }
```

```
14  990 --
14  991 --              FreeWmgrWindow(aWmgrWindow, TRUE);
14  992 --
14  993 -0 B         END;
14  994 --
14  995 0- A    BEGIN
14  996 --          aWmgrWindow := gApplication.GetRsrcWindow(NIL, itsRsrcID, canResize, canClose);
14  997 --              { GetRsrcWindow signals Failure }
14  998 --
14  999 --          CatchFailures(fi, HdlNewWObj);
14 1000 --
14 1001 --          aWindow := NIL;
14 1002 --
14 1003 --          New(aWindow);
14 1004 --          FailNIL(aWindow);
14 1005 --          Success(fi);
14 1006 --
14 1007 --          aWindow.IWindow(itsDocument, aWmgrWindow, canResize, canClose, TRUE);   { TRUE means can dispose wmgr window }
14 1008 --
14 1009 --          NewTWindow := aWindow;
14 1010 --
14 1011 -0 A    END;
14 1012 --      {$ENDC}
14 1013 --
14 1014 --
14 1015 --      {$IFC qTemplateViews}
14 1016 --      {$S MAOpen}
14 1017 --      {-----------------------------------+
14 1018 --      |   NewTemplateWindow               |
14 1019 --      +-----------------------------------}
14 1020 -- A    FUNCTION  NewTemplateWindow (viewRsrcID: INTEGER; itsDocument: TDocument): TWindow;
14 1021 --
14 1022 --      VAR
14 1023 --          theWindow:      TWindow;
14 1024 --          theTarget:      TView;
14 1025 --
14 1026 0- A    BEGIN
14 1027 --          theWindow := TWindow(gTarget.DoCreateViews(itsDocument, NIL, viewRsrcID));
14 1028 --          IF theWindow <> NIL THEN
14 1029 1-              BEGIN
14 1030 --              IF theWindow.fWMgrWindow <> NIL THEN
14 1031 --                  WITH theWindow.fWMgrWindow^.portRect DO
14 1032 --                      theWindow.Resize(right - left, bottom - top, FALSE);
14 1033 --              IF theWindow.fTargetID <> kNoIdentifier THEN
14 1034 2-                  BEGIN
14 1035 --                  theTarget := theWindow.FindSubView(theWindow.fTargetID);
14 1036 --                  IF theTarget <> NIL THEN
14 1037 --                      theWindow.fTarget := theTarget;
14 1038 -2                  END;
14 1039 -1              END;
14 1040 --          NewTemplateWindow := theWindow;
14 1041 -0 A    END;
14 1042 --      {$ENDC}
14 1043 --
14 1044 --
14 1045 --      {$IFC qDebug}
14 1046 --      {$S MADebug}
14 1047 --      {$IFC qTrace}{$D+}{$ENDC}
14 1048 --      {-----------------------------------+
14 1049 --      |   NotYetImplemented               |
14 1050 --      +-----------------------------------}
14 1051 -- A    PROCEDURE NotYetImplemented(where: Str255);
14 1052 0- A    BEGIN
14 1053 --          StdAlert(phUnimplemented);
14 1054 --          WriteLn('Feature Not Yet Implemented: ', where);
14 1055 -0 A    END;
14 1056 --      {$IFC qTrace}{$D++}{$ENDC}
14 1057 --      {$ENDC}
14 1058 --
14 1059 --      {$S MARes}
14 1060 --      {-----------------------------------+
14 1061 --      |   ParseTitleTemplate              |
14 1062 --      +-----------------------------------}
14 1063 -- A    FUNCTION  ParseTitleTemplate(VAR template: Str255; VAR preDocname, constTitle: INTEGER): BOOLEAN;
14 1064 --      CONST
14 1065 --          kPreDocname = '<<<';
14 1066 --          kPreSize = 3;
14 1067 --          kPostDocname = '>>>';
14 1068 --          kPostSize = 3;
14 1069 --
14 1070 --      VAR
14 1071 --          x:  INTEGER;
14 1072 --
14 1073 -- B        FUNCTION FindPos (pattern: Str255; VAR source: Str255): INTEGER;
14 1074 --          VAR
14 1075 --              position:   INTEGER;
14 1076 0- B        BEGIN
14 1077 --              IF gConfiguration.hasScriptManager THEN
14 1078 1-                  REPEAT
14 1079 --                      position := POS(pattern, source);
14 1080 -1                  UNTIL (position = 0) OR (CharByte(@source, position) = 0)
14 1081 --              ELSE
14 1082 --                  position := POS(pattern, source);
14 1083 --              FindPos := position;
14 1084 -0 B        END;
14 1085 --
14 1086 0- A    BEGIN
14 1087 --          IF template = '' THEN
14 1088 1-              BEGIN
```

```
14 1089 --              preDocname := 1;
14 1090 --              constTitle := 0;
14 1091 -1              END
14 1092 --          ELSE
14 1093 1-              BEGIN
14 1094 --              preDocname := FindPos(kPreDocname, template);
14 1095 --              IF preDocname > 0 THEN
14 1096 2-                  BEGIN
14 1097 --                  Delete(template, preDocname, kPreSize);
14 1098 --
14 1099 --                  x := FindPos(kPostDocname, template);
14 1100 --                  IF x = 0 THEN
14 1101 --                      constTitle := preDocname - 1
14 1102 --                  ELSE
14 1103 3-                      BEGIN
14 1104 --                      Delete(template, x, kPostSize);
14 1105 --                      constTitle := Length(template) - x + preDocname;
14 1106 -3                      END;
14 1107 -2                  END;
14 1108 -1              END;
14 1109 --
14 1110 --          ParseTitleTemplate := preDocname > 0;
14 1111 -0 A   END;
14 1112 --
14 1113 --      {$S MAActivate}
14 1114 --      {-----------------------------------+
14 1115 --      |   PutDeskScrapData                |
14 1116 --      +-----------------------------------}
14 1117 -- A   FUNCTION  PutDeskScrapData(aResType: ResType; aDataHandle: Handle): OSErr;
14 1118 --              VAR err:    LONGINT;
14 1119 0- A       BEGIN
14 1120 --              MoveHHi(aDataHandle);
14 1121 --              HLock(aDataHandle);
14 1122 --              err := PutScrap(GetHandleSize(aDataHandle), aResType, aDataHandle^);
14 1123 --              HUnlock(aDataHandle);
14 1124 --      {$IFC qDebug}
14 1125 --              IF err <> noErr THEN
14 1126 --                  WriteLn('Error from PutScrap is: ', err:1);
14 1127 --      {$ENDC}
14 1128 --              PutDeskScrapData := err;
14 1129 -0 A       END;
14 1130 --
14 1131 --      {$S MARes}
14 1132 --      {-----------------------------------+
14 1133 --      |   RectIsVisible                   |
14 1134 --      +-----------------------------------}
14 1135 -- A   FUNCTION  RectIsVisible(r: Rect): BOOLEAN;
14 1136 0- A   BEGIN
14 1137 --      {$IFC FALSE}        { ??? Eventually get rid of this in MacApp 2.0? }
14 1138 --          IF gPrinting THEN
14 1139 --              RectIsVisible := gCurrPrintHandler.WouldPrintRect(r)
14 1140 --          ELSE
14 1141 --      {$ENDC}
14 1142 --          IF gDrawingPictScrap THEN
14 1143 --              RectIsVisible := TRUE
14 1144 --          ELSE
14 1145 --              RectIsVisible := RectInRgn(r, thePort^.visRgn);
14 1146 --                  { ??? need to check the clipping also ??? }
14 1147 -0 A   END;
14 1148 --
14 1149 --
14 1150 --      {$IFC qTemplateViews}
14 1151 --      {$S MAMain}
14 1152 --      {-----------------------------------+
14 1153 --      |   RegisterType                    |
14 1154 --      +-----------------------------------}
14 1155 -- A   PROCEDURE RegisterType (typeName: Str255; protoObj: TObject);
14 1156 --      VAR
14 1157 --          anEntry:        TEntry;
14 1158 0- A   BEGIN
14 1159 --          gProtoList.InsertEntry(typeName, '');
14 1160 --          anEntry := gProtoList.EntryWithKey(typeName);
14 1161 --          anEntry.fValue := StringHandle(protoObj);
14 1162 -0 A   END;
14 1163 --      {$ENDC}
14 1164 --
14 1165 --
14 1166 --      {$IFC qTemplateViews}
14 1167 --      {$S MAMain}
14 1168 --      {-----------------------------------+
14 1169 --      |   RegisterStdType                 |
14 1170 --      +-----------------------------------}
14 1171 -- A   PROCEDURE RegisterStdType (typeName: Str255; signature: IDType; protoObj: TObject);
14 1172 0- A   BEGIN
14 1173 --          gSignatureCount := gSignatureCount + 1;
14 1174 --      {$IFC qDebug}
14 1175 --          IF gSignatureCount >= kMaxSignatures THEN
14 1176 --              ProgramBreak('Maximum number of signature views exceeded.');
14 1177 --      {$ENDC}
14 1178 --          gSignatures[gSignatureCount] := signature;
14 1179 --          gSignatureViews[gSignatureCount] := protoObj;
14 1180 --
14 1181 --          IF LENGTH(typeName) <> 0 THEN
14 1182 --              RegisterType(typeName, protoObj);
14 1183 -0 A   END;
14 1184 --      {$ENDC}
14 1185 --
14 1186 --
14 1187 --      {$S MARes}
```

```
14 1188 --      {-----------------------------------+
14 1189 --      |   SetFocus                        |
14 1190 --      +-----------------------------------}
14 1191 -- A    PROCEDURE SetFocus;
14 1192 --
14 1193 0- A    BEGIN
14 1194 --          SetPort(gSavePort);
14 1195 --          SetOrigin(gSaveOrg.h, gSaveOrg.v);
14 1196 --          SetClip(gSaveClip);
14 1197 --          gLongOffset := gSaveLongOffset;
14 1198 --          gFocusedView := gSaveFocusedView;
14 1199 -0 A    END { SetFocus };
14 1200 --
14 1201 --      {$S MARes}
14 1202 --      {-----------------------------------+
14 1203 --      |   SetHLPenState                   |
14 1204 --      +-----------------------------------}
14 1205 -- A    PROCEDURE SetHLPenState(fromHL, toHL: HLState);
14 1206 --          VAR pPat:   ^Pattern;
14 1207 --              mode:   INTEGER;
14 1208 0- A    BEGIN
14 1209 --          mode := patXOR;         { every transition except hlOn <-> hlDim uses patXOR }
14 1210 --
14 1211 --
14 1212 --          IF fromHL = toHL THEN
14 1213 --              pPat := @white
14 1214 --
14 1215 --          ELSE IF fromHL + toHL = hlOffOn THEN
14 1216 --              pPat := @black
14 1217 --
14 1218 --          ELSE
14 1219 --              pPat := @gray; { ??? make this pattern a parameter ??? }
14 1220 --
14 1221 --          IF fromHL + toHL = hlDimOn THEN
14 1222 --              mode := NOTpatXOR;
14 1223 --
14 1224 --          PenMode(mode);
14 1225 --          PenPat(pPat^);
14 1226 -0 A    END;
14 1227 --
14 1228 --      {$S MAMain}
14 1229 --      {-----------------------------------+
14 1230 --      |   StdAlert                        |
14 1231 --      +-----------------------------------}
14 1232 -- A    PROCEDURE StdAlert(alertId: INTEGER);
14 1233 --          VAR reply: INTEGER;
14 1234 0- A    BEGIN
14 1235 --          reply := MacAppAlert(alertId, NIL);
14 1236 -0 A    END;
14 1237 --
14 1238 --      {$S MARes}
14 1239 --      {-----------------------------------+
14 1240 --      |   SubstituteInTitle               |
14 1241 --      +-----------------------------------}
14 1242 -- A    FUNCTION  SubstituteInTitle(VAR title: Str255; newStuff: Str255;
14 1243 --                                      preDocname, constTitle: INTEGER): BOOLEAN;
14 1244 0- A    BEGIN
14 1245 --          IF preDocname > 0 THEN
14 1246 1-              BEGIN
14 1247 --              IF constTitle = 0 THEN
14 1248 --                  title := newStuff
14 1249 --              ELSE
14 1250 2-                  BEGIN
14 1251 --                  Delete(title, preDocname, Length(title) - constTitle);
14 1252 --                  Insert(newStuff, title, preDocname);
14 1253 -2                  END;
14 1254 --              SubstituteInTitle := TRUE;
14 1255 -1              END
14 1256 --          ELSE
14 1257 --              SubstituteInTitle := FALSE;
14 1258 -0 A    END;
14 1259 --
14 1260 --      {$IFC qDebug}
14 1261 --      {$S MADebug}
14 1262 --      {$IFC qTrace}{$D+}{$ENDC}
14 1263 --      {-----------------------------------+
14 1264 --      |   UseTempRgn                      |
14 1265 --      +-----------------------------------}
14 1266 -- A    PROCEDURE UseTempRgn(byWhom: Str255);
14 1267 --          { Call this when you are about to use gTempRgn and qDebug is true.  Used
14 1268 --              with DoneWithTempRgn will prevent you from trying to use gTempRgn
14 1269 --              from two places at the same time. }
14 1270 --
14 1271 0- A    BEGIN
14 1272 --          IF gBusyTempRgn THEN
14 1273 1-              BEGIN
14 1274 --              WriteLn('"', byWhom, '" is trying to lock gTempRgn,');
14 1275 --              WriteLn('but it is already locked by "', gUsedBy, '"');
14 1276 --              ProgramBreak('Error in UseTempRgn');
14 1277 -1              END
14 1278 --          ELSE
14 1279 1-              BEGIN
14 1280 --              gBusyTempRgn := TRUE;
14 1281 --              gUsedBy := byWhom;
14 1282 -1              END;
14 1283 -0 A    END;
14 1284 --      {$IFC qTrace}{$D++}{$ENDC}
14 1285 --      {$ENDC qDebug}
13 3002 --      {$I UMacApp.TEvtHandler.p}
```

```
15     1 --      {$P}
15     2 --      { UMacApp.TEvtHandler.p }
15     3 --      { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
15     4 --
15     5 --      (*****************************************************************************************)
15     6 --      (*      T E v t H a n d l e r                                                            *)
15     7 --      (*****************************************************************************************)
15     8 --      {$S MARes}
15     9 --      {-----------------------------------+
15    10 --      |   IEvtHandler                     |
15    11 --      +-----------------------------------}
15    12 -- A    PROCEDURE TEvtHandler.IEvtHandler(itsNextHandler: TEvtHandler);
15    13 0- A    BEGIN
15    14 --          fNextHandler := itsNextHandler;
15    15 --          fIdleFreq := kMaxIdleTime;              { Assume it never wants idle time }
15    16 --          fLastIdle := 0;
15    17 -0 A    END;
15    18 --
15    19 --      {$S MAClose}
15    20 --      {-----------------------------------+
15    21 --      |   Free                            |
15    22 --      +-----------------------------------}
15    23 -- A    PROCEDURE TEvtHandler.Free; OVERRIDE;
15    24 0- A    BEGIN
15    25 --          {??? Should this call SetTarget to set the target?}
15    26 --          IF gTarget = SELF THEN
15    27 --              IF fNextHandler = NIL THEN
15    28 --                  gTarget := gApplication
15    29 --              ELSE
15    30 --                  gTarget := fNextHandler;
15    31 --          INHERITED Free;
15    32 -0 A    END;
15    33 --
15    34 --      {$IFC qDebug}
15    35 --      {$S MAFields}
15    36 --      {-----------------------------------+
15    37 --      |   Fields                          |
15    38 --      +-----------------------------------}
15    39 -- A    PROCEDURE TEvtHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
15    40 --                                                         fieldAddr: Ptr;
15    41 --                                                         fieldType: INTEGER));
15    42 0- A    BEGIN
15    43 --          DoToField('TEvtHandler', NIL, bClass);
15    44 --          DoToField('fNextHandler', @fNextHandler, bObject);
15    45 --          DoToField('fIdleFreq', @fIdleFreq, bLongInt);
15    46 --          DoToField('fLastIdle', @fLastIdle, bLongInt);
15    47 --          INHERITED Fields(DoToField);
15    48 -0 A    END {TEvtHandler.Fields};
15    49 --      {$ENDC}
15    50 --
15    51 --
15    52 --      {$IFC qTemplateViews}
15    53 --      {$S MAOpen}
15    54 --      {-----------------------------------+
15    55 --      |   CreateAView                     |
15    56 --      +-----------------------------------}
15    57 -- A    FUNCTION  TEvtHandler.CreateAView ( itsDocument: TDocument; itsSuperView: TView;VAR itsParams: Ptr): TView;
15    58 --      VAR
15    59 --          anEntry:    TEntry;
15    60 --          aView:      TView;
15    61 --          newView:    TView;
15    62 --
15    63 0- A    BEGIN
15    64 --          IF fNextHandler <> NIL THEN
15    65 --              CreateAView := fNextHandler.CreateAView(itsDocument, itsSuperView, itsParams)
15    66 --          ELSE
15    67 1-              BEGIN
15    68 --              WITH ViewTemplatePtr(itsParams)^ DO
15    69 --                  IF LENGTH(itsType) = 0 THEN
15    70 --                      aView := FindStdView(itsSignature)
15    71 --                  ELSE
15    72 2-                      BEGIN
15    73 --                      anEntry := gProtoList.EntryWithKey(itsType);
15    74 --                      IF anEntry <> NIL THEN
15    75 --                          aView := TView(anEntry.fValue)
15    76 --                      ELSE
15    77 --                          aView := NIL;
15    78 -2                      END;
15    79 --
15    80 --              IF aView <> NIL THEN
15    81 2-                  BEGIN
15    82 --                  newView := TView(aView.ShallowClone);
15    83 --                  FailNIL(newView);
15    84 --                  newView.IRes(itsDocument, itsSuperView, itsParams);
15    85 --                  CreateAView := newView;
15    86 -2                  END
15    87 --              ELSE
15    88 2-                  BEGIN
15    89 --          {$IFC qDebug}
15    90 --                  ProgramBreak(CONCAT('Class ''', ViewTemplatePtr(itsParams)^.itsType,
15    91 --                                      ''' is not registered.'));
15    92 --          {$ENDC}
15    93 --                  CreateAView := NIL;
15    94 -2                  END;
15    95 -1              END;
15    96 -0 A    END;
15    97 --      {$ENDC}
15    98 --
```

```
15   99 --
15  100 --      {$IFC qTemplateViews}
15  101 --      {$S MAOpen}
15  102 --      {-----------------------------------+
15  103 --      |   DoCreateViews                   |
15  104 --      +-----------------------------------}
15  105 -- A    FUNCTION  TEvtHandler.DoCreateViews (itsDocument: TDocument; parentView: TView; itsRsrcID: INTEGER): TView;
15  106 --      VAR
15  107 --          i:              INTEGER;
15  108 --          numViews:       INTEGER;
15  109 --          aView:          TView;
15  110 --          aHandle:        ViewRsrcHndl;
15  111 --          theViewInfo:    ViewTemplatePtr;
15  112 --          lastParentID:   IDType;
15  113 --          lastParent:     TView;
15  114 --          firstView:      TView;
15  115 --          fi:             FailInfo;
15  116 --
15  117 -- B        PROCEDURE HdlDoCreateViews (error: OSErr; message: LONGINT);
15  118 0- B       BEGIN
15  119 --              FreeObject(firstView);
15  120 -0 B       END;
15  121 --
15  122 0- A    BEGIN
15  123 --          IF fNextHandler <> NIL THEN
15  124 --              DoCreateViews := fNextHandler.DoCreateViews(itsDocument, parentView, itsRsrcID)
15  125 --          ELSE
15  126 1-              BEGIN
15  127 --              firstView := NIL;                   { Assume the worst.                         }
15  128 --
15  129 --              aHandle := ViewRsrcHndl(GetResource('view', itsRsrcID));
15  130 --              FailNIL(Handle(aHandle));
15  131 --              MoveHHi(Handle(aHandle));
15  132 --              HLock(Handle(aHandle));
15  133 --
15  134 --              CatchFailures(fi, HdlDoCreateViews);
15  135 --
15  136 --              numViews := aHandle^^.numViews;
15  137 --              theViewInfo := @aHandle^^.theViews;
15  138 --              lastParentID := kNoIdentifier;
15  139 --              lastParent := NIL;
15  140 --
15  141 --              FOR i := 1 TO numViews DO
15  142 --                  WITH theViewInfo^ DO
15  143 2-                     BEGIN
15  144 --      {$IFC qDebug}
15  145 --                      IF gIntenseDebugging THEN
15  146 3-                         BEGIN
15  147 --                          WrLblSig('signature', itsSignature);
15  148 --                          WRITELN;
15  149 --                          WrLblSig('itsParentID', itsParentID);
15  150 --                            WrLblSig(', thisViewID', thisViewID);
15  151 --                            WriteLn;
15  152 --                          WrLblVPt('itsLocation', itsLocation);
15  153 --                            WrLblVPt(', itsSize', itsSize);
15  154 --                          Write('itsHSizeDet = ', ORD(itsHSizeDet):3);
15  155 --                            WriteLn(', itsVSizeDet = ', ORD(itsVSizeDet):3);
15  156 --                            WrLblBoolean(', isEnabled ', isEnabled);
15  157 --                            WriteLn;
15  158 --                          WriteLn('----------  end of view  ----------');
15  159 -3                         END;
15  160 --      {$ENDC}
15  161 --
15  162 --                      IF LONGINT(itsParentID) = LONGINT(kNoIdentifier) THEN
15  163 --                          lastParent := parentView
15  164 --                      ELSE IF (LONGINT(itsParentID) <> LONGINT(lastParentID)) & (parentView <> NIL) THEN
15  165 3-                         BEGIN
15  166 --                          lastParent := parentView.FindSubView(itsParentID);
15  167 --      {$IFC qDebug}
15  168 --                          IF lastParent = NIL THEN
15  169 --                              ProgramBreak('Unable to find parent view for template');
15  170 --      {$ENDC}
15  171 -3                         END;
15  172 --                      lastParentID := itsParentID;
15  173 --
15  174 --                      IF itsSignature = 'incl' THEN
15  175 3-                         BEGIN
15  176 --                          aView := DoCreateViews(itsDocument, lastParent, includeRsrcID);
15  177 --                          OffsetPtr(theViewInfo, SIZEOF(ViewTemplate) - SIZEOF(Str255) + SIZEOF(INTEGER));
15  178 -3                         END
15  179 --                      ELSE
15  180 --                          aView := CreateAView(itsDocument, lastParent, Ptr(theViewInfo));
15  181 --
15  182 --                      IF aView = NIL THEN
15  183 --                          LEAVE;
15  184 --                      IF parentView = NIL THEN
15  185 --                          parentView := aView;
15  186 --                      IF i = 1 THEN
15  187 --                          firstView := aView;
15  188 -2                     END;
15  189 --
15  190 --              HUnlock(Handle(aHandle));
15  191 --              Success(fi);
15  192 --              DoCreateViews := firstView;
15  193 -1              END;
15  194 -0 A    END;
15  195 --      {$ENDC}
15  196 --
15  197 --      {$S MARes}
```

```
15  198 --      {-------------------------------------+
15  199 --      |   DoCommandKey                      |
15  200 --      +-------------------------------------}
15  201 -- A    FUNCTION  TEvtHandler.DoCommandKey (ch: Char; VAR info: EventInfo): TCommand;
15  202 0- A    BEGIN
15  203 --          IF fNextHandler <> NIL THEN
15  204 --              DoCommandKey := fNextHandler.DoCommandKey(ch, info)
15  205 --          ELSE
15  206 --              DoCommandKey := gNoChanges;
15  207 -0 A     END;
15  208 --
15  209 --      {$S MARes}
15  210 --      {-------------------------------------+
15  211 --      |   DoHandleEvent                     |
15  212 --      +-------------------------------------}
15  213 -- A    FUNCTION  TEvtHandler.DoHandleEvent(nextEvent: PEventRecord;
15  214 --                          VAR commandToPerform: TCommand): BOOLEAN;
15  215 0- A    BEGIN
15  216 --          DoHandleEvent := FALSE;
15  217 -0 A     END;
15  218 --
15  219 --      {$IFC qTrace}{$D+}{$ENDC}    {??? No longer want this compile-time stuff here ???}
15  220 --      {$S MARes}
15  221 --      {-------------------------------------+
15  222 --      |   DoIdle                            |
15  223 --      +-------------------------------------}
15  224 -- A    PROCEDURE TEvtHandler.DoIdle(phase: IdlePhase);
15  225 0- A    BEGIN
15  226 -0 A     END;
15  227 --      {$IFC qTrace}{$D++}{$ENDC}
15  228 --
15  229 --      {$S MARes}
15  230 --      {-------------------------------------+
15  231 --      |   DoKeyCommand                      |
15  232 --      +-------------------------------------}
15  233 -- A    FUNCTION  TEvtHandler.DoKeyCommand(ch: Char; aKeyCode: INTEGER; VAR info: EventInfo): TCommand;
15  234 0- A    BEGIN
15  235 --          IF fNextHandler <> NIL THEN
15  236 --              DoKeyCommand := fNextHandler.DoKeyCommand(ch, aKeyCode, info)
15  237 --          ELSE
15  238 --              DoKeyCommand := gNoChanges;  {??? give an error ???}
15  239 -0 A     END;
15  240 --
15  241 --      {$S MASelCommand}
15  242 --      {-------------------------------------+
15  243 --      |   DoMenuCommand                     |
15  244 --      +-------------------------------------}
15  245 -- A    FUNCTION  TEvtHandler.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
15  246 0- A    BEGIN
15  247 --          IF fNextHandler <> NIL THEN
15  248 --              DoMenuCommand := fNextHandler.DoMenuCommand(aCmdNumber)
15  249 --          ELSE
15  250 1-              BEGIN
15  251 --              {$IFC qDebug}
15  252 --              WriteLn('No one handled the command ', aCmdNumber:1);
15  253 --              {$ENDC}
15  254 --              DoMenuCommand := gNoChanges;
15  255 -1              END;
15  256 -0 A     END;
15  257 --
15  258 --      {$S MARes}
15  259 --      {-------------------------------------+
15  260 --      |   DoMultiClick                      |
15  261 --      +-------------------------------------}
15  262 -- A    FUNCTION  TEvtHandler.DoMultiClick(lastDownPt, newDownPt: Point): BOOLEAN;
15  263 0- A    BEGIN
15  264 --          IF fNextHandler <> NIL THEN
15  265 --              DoMultiClick := fNextHandler.DoMultiClick(lastDownPt, newDownPt)
15  266 --          ELSE
15  267 --              DoMultiClick := ABS(lastDownPt.h - newDownPt.h) +
15  268 --                                  ABS(lastDownPt.v - newDownPt.v) <= 5;  {??? arbitrary choice ???}
15  269 -0 A     END;
15  270 --
15  271 --      {$IFC qTrace}{$D+}{$ENDC}
15  272 --      {$S MARes}
15  273 --      {-------------------------------------+
15  274 --      |   DoSetupMenus                      |
15  275 --      +-------------------------------------}
15  276 -- A    PROCEDURE TEvtHandler.DoSetupMenus;
15  277 0- A    BEGIN
15  278 --          IF fNextHandler <> NIL THEN
15  279 --              fNextHandler.DoSetupMenus;
15  280 -0 A     END;
15  281 --      {$IFC qTrace}{$D++}{$ENDC}
15  282 --
15  283 --      {$S MARes}
15  284 --      {-------------------------------------+
15  285 --      |   HandlesPrintingCommands           |
15  286 --      +-------------------------------------}
15  287 -- A    FUNCTION  TEvtHandler.HandlesPrintingCommands: BOOLEAN;
15  288 0- A    BEGIN
15  289 --          IF fNextHandler <> NIL THEN
15  290 --              HandlesPrintingCommands := fNextHandler.HandlesPrintingCommands
15  291 --          ELSE
15  292 --              HandlesPrintingCommands := FALSE
15  293 -0 A     END;
15  294 --
15  295 --      {$IFC qDebug}
15  296 --      {$S MADebug}
```

```
15  297 --      {$IFC qTrace}{$D+}{$ENDC}
15  298 --      {-----------------------------------+
15  299 --      |   IdentifySoftware                |
15  300 --      +-----------------------------------}
15  301 -- A    PROCEDURE TEvtHandler.IdentifySoftware;
15  302 0- A    BEGIN
15  303 --          IF fNextHandler <> NIL THEN
15  304 --              fNextHandler.IdentifySoftware;
15  305 -0 A    END;
15  306 --      {$IFC qTrace}{$D++}{$ENDC}
15  307 --      {$ENDC}
15  308 --
15  309 --      {$S MAActivate}
15  310 --      {-----------------------------------+
15  311 --      |   InstallSelection                |
15  312 --      +-----------------------------------}
15  313 -- A    PROCEDURE TEvtHandler.InstallSelection(wasActive, beActive: BOOLEAN);
15  314 0- A    BEGIN
15  315 -0 A    END;
15  316 --
15  317 --      {$IFC qDebug}
15  318 --      {$S MADebug}
15  319 --      {$IFC qTrace}{$D+}{$ENDC}
15  320 --      {-----------------------------------+
15  321 --      |   LookupSymbol                    |
15  322 --      +-----------------------------------}
15  323 -- A    FUNCTION  TEvtHandler.LookupSymbol(VAR sym: String8): LONGINT;
15  324 0- A    BEGIN
15  325 --          IF fNextHandler <> NIL THEN
15  326 --              LookupSymbol := fNextHandler.LookupSymbol(sym)
15  327 --          ELSE
15  328 --              LookupSymbol := -1;
15  329 -0 A    END;
15  330 --      {$IFC qTrace}{$D++}{$ENDC}
15  331 --      {$ENDC}
15  332 --
15  333 --      {$S MATerminate}
15  334 --      {-----------------------------------+
15  335 --      |   Terminate                       |
15  336 --      +-----------------------------------}
15  337 -- A    PROCEDURE TEvtHandler.Terminate;
15  338 0- A    BEGIN
15  339 -0 A    END;
13 3003 --      {$I UMacApp.TApplication.p}
```

```
16    1 --      {$P}
16    2 --      { UMacApp.TApplication.p }
16    3 --      { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
16    4 --
16    5 --      {$S MAInit}
16    6 --      {-----------------------------------+   This routine preserves the menu color table information
16    7 --      |   MAGetNewMBar                    |       which is rudely disposed of by GetNewMBar
16    8 --      +-----------------------------------}
16    9 -- A    FUNCTION  MAGetNewMBar (menuRsrcID: INTEGER): Handle;
16   10 --
16   11 --      TYPE
16   12 --          MenuCRsrc = RECORD
16   13 --                          numEntries:     INTEGER;
16   14 --                          data:           ARRAY [1..1092] OF MCEntry;
16   15 --                      END;
16   16 --          PMenuCRsrc = ^MenuCRsrc;
16   17 --          HMenuCRsrc = ^PMenuCRsrc;
16   18 --
16   19 --      VAR
16   20 --          theColorTab:        MCTableHandle;
16   21 --          theColorRsrc:       HMenuCRsrc;
16   22 --
16   23 0- A    BEGIN
16   24 --          IF gConfiguration.hasColorQD THEN
16   25 --              theColorTab := GetMCInfo;
16   26 --          MAGetNewMBar := GetNewMBar(menuRsrcID);
16   27 --          IF gConfiguration.hasColorQD & (theColorTab <> NIL) THEN
16   28 1-              BEGIN
16   29 --              HLock(Handle(theColorTab));
16   30 --                  SetMCEntries(GetHandleSize(Handle(theColorTab)) DIV SIZEOF(MCEntry), theColorTab^);
16   31 --              HUnlock(Handle(theColorTab));
16   32 --              DispMCInfo(theColorTab);
16   33 --      {$IFC FALSE}
16   34 --              { Now that the old and new worlds coincide, let's see if an 'mctb' with the same
16   35 --                  number as the menu bar is around, and load it as a convenience to the programmer.
16   36 --                  THIS ISN'T STANDARD ACCEPTED PRACTICE, but the worst that can happen is that an
16   37 --                  errant 'mctb' resource for a single menu might be reloaded. }
16   38 --              theColorRsrc := HMenuCRsrc(GetResource('mctb', menuRsrcID));
16   39 --              IF theColorRsrc <> NIL THEN
16   40 2-                  BEGIN
16   41 --                  HLock(Handle(theColorRsrc));
16   42 --                      WITH theColorRsrc^^ DO
16   43 --                          SetMCEntries(numEntries, @data[1]);
16   44 --                  HUnlock(Handle(theColorRsrc));
16   45 --                  ReleaseResource(Handle(theColorRsrc));
16   46 -2                  END;
16   47 --      {$ENDC}
16   48 -1              END;
16   49 -0 A    END;
16   50 --
16   51 --      (*****************************************************************************************)
16   52 --      (*      T A p p l i c a t i o n                                                          *)
16   53 --      (*****************************************************************************************)
16   54 --      {$S MAInit}
16   55 --      {-----------------------------------+
16   56 --      |   IApplication                    |
16   57 --      +-----------------------------------}
16   58 -- A    PROCEDURE TApplication.IApplication(itsMainFileType: OSType);
16   59 --
16   60 --      TYPE
16   61 --          MenuBarHdl =    ^MenuBarPtr;
16   62 --          MenuBarPtr =    ^MenuBarRecord;
16   63 --          MenuBarRecord = RECORD
16   64 --                          nMenus:         INTEGER;
16   65 --                          menuID:         ARRAY [1..1000] OF INTEGER;
16   66 --                          END;
16   67 --
16   68 --      VAR
16   69 --          menuID:         INTEGER;
16   70 --          aMenu:          MenuHandle;
16   71 --          mbar:           Handle;
16   72 --          hmbar:          MenuBarHdl;
16   73 --          i:              INTEGER;
16   74 --
16   75 0- A    BEGIN
16   76 --          gApplication := SELF;
16   77 --          gAppDone := FALSE;
16   78 --          gSysWindowActive := FALSE;
16   79 --          gTarget := SELF;
16   80 --
16   81 --          IEvtHandler(NIL);
16   82 --
16   83 --      {$IFC qInspector}
16   84 --          MakeInspector;
16   85 --          AddObjectToInspector(SELF);
16   86 --          AddObjectToInspector(gNullPrintHandler);
16   87 --          AddObjectToInspector(gFreeWindowList);
16   88 --      {$ENDC}
16   89 --
16   90 --          gDocList := NewList;
16   91 --      {$IFC qDebug}
16   92 --          gDocList.SetEltType('TDocument');
16   93 --      {$ENDC}
16   94 --
16   95 --          gMainFileType := itsMainFileType;
16   96 --
16   97 --          gDocument := NIL;
16   98 --          gLastCommand := NIL;
```

```
16   99 --
16  100 --          gVarClipPicSize := FALSE;     { temporary }
16  101 --
16  102 --          IF NOT gFinderPrinting THEN
16  103 1-              BEGIN
16  104 --              mbar := MAGetNewMBar(gMBarDisplayed);
16  105 --              IF mbar <> NIL THEN
16  106 2-                  BEGIN
16  107 --                  SetMenuBar(mbar);
16  108 --                  DisposHandle(mbar);
16  109 -2                 END
16  110 --              ELSE
16  111 2-                  BEGIN
16  112 --                  {$IFC qDebug}
16  113 --                  Writeln('The MBAR ', gMBarDisplayed:1, ' resource was not specified.');
16  114 --                  ProgramBreak('You will not have any menus!');
16  115 --                  {$ENDC}
16  116 -2                 END;
16  117 --
16  118 --      {$IFC qDebug}
16  119 --              aMenu := GetMenu(mDebug);
16  120 --              IF aMenu <> NIL THEN
16  121 --                  InsertMenu(aMenu, 0);
16  122 --      {$ENDC}
16  123 --
16  124 --              aMenu := GetResMenu(mApple);
16  125 --              IF aMenu <> NIL THEN
16  126 --                  AddResMenu(aMenu, 'DRVR');
16  127 --
16  128 --              mbar := MAGetNewMBar(gMBarNotDisplayed);
16  129 --              DisposIfHandle(mbar);
16  130 --
16  131 --              IF gConfiguration.hasHierarchicalMenus THEN
16  132 2-                  BEGIN
16  133 --                  hmbar := MenuBarHdl(GetResource('MBAR', gMBarHierarchical));
16  134 --                  IF hmbar <> NIL THEN
16  135 3-                      BEGIN
16  136 --                      FOR i := 1 TO hmbar^^.nMenus DO
16  137 4-                          BEGIN
16  138 --                          aMenu := GetMenu(hmbar^^.menuID[i]);
16  139 --                          IF aMenu <> NIL THEN
16  140 --                              InsertMenu(aMenu, -1);
16  141 -4                         END;
16  142 --                      DisposHandle(Handle(hmbar));
16  143 -3                      END;
16  144 -2                  END;
16  145 --
16  146 --              New(gClipOrphanage);
16  147 --              FailNil(gClipOrphanage);
16  148 --              gClipOrphanage.IDeskScrapView;
16  149 --
16  150 --              gClipWindow := MakeClipboardWindow;
16  151 -1              END;
16  152 --
16  153 --      {$IFC qDebug}
16  154 --              { These must occur after the application is initialized }
16  155 --          TRCGlobalHandle(@gTarget,           'gTarget');
16  156 --          TRCGlobalHandle(@gLastCommand,      'gLastCommand');
16  157 --          TRCGlobalHandle(@gDocument,         'gDocument');
16  158 --          TRCGlobalHandle(@gApplication,      'gApplication');
16  159 --          TRCGlobalHandle(@gDocList,          'gDocList');
16  160 --          TRCGlobalHandle(@gFreeWindowList,   'gFreeWindowList');
16  161 --          TRCGlobalHandle(@gClipView,         'gClipView');
16  162 --          TRCGlobalHandle(@gClipUndoView,     'gClipUndoView');
16  163 --          TRCGlobalHandle(@gPrintHandler,     'gPrintHandler');
16  164 --          TRCGlobalHandle(@gFrontWindow,      'gFrontWindow');
16  165 --          TRCGlobalHandle(@gFocusedView,      'gFocusedView');
16  166 --      {$ENDC}
16  167 -0 A   END;
16  168 --
16  169 --      {$S MAClipboard}
16  170 --      {-----------------------------------+
16  171 --      |   AbandonUndoClipboard            |
16  172 --      +-----------------------------------}
16  173 -- A   PROCEDURE TApplication.AbandonUndoClipboard;
16  174 0- A   BEGIN
16  175 --          IF gClipUndoView <> NIL THEN
16  176 1-              BEGIN
16  177 --      {$IFC qDebug}
16  178 --              IF gClipUndoView = gClipView THEN
16  179 --                  ProgramBreak('About to Free view both in clip and undo Clip!');
16  180 --      {$ENDC}
16  181 --              gClipUndoView.FreeFromClipboard;
16  182 --              gClipUndoView := NIL;
16  183 -1              END;
16  184 -0 A   END;
16  185 --
16  186 --      {$S MAActivate}
16  187 --      {-----------------------------------+
16  188 --      |   AboutToLoseControl              |
16  189 --      +-----------------------------------}
16  190 -- A   PROCEDURE TApplication.AboutToLoseControl(convertClipboard: BOOLEAN);
16  191 --      LABEL 1000;
16  192 --      VAR
16  193 --          err:    LONGINT;
16  194 --          fi:     FailInfo;
16  195 --
16  196 --          {-------------------------------+
16  197 --          |   PublicizeFailed             |
```

```
16  198 --          +--------------------------------}
16  199 -- B       PROCEDURE PublicizeFailed(error: INTEGER; message: LONGINT); { ??? ERROR ??? }
16  200 0- B       BEGIN
16  201 --      {$IFC qDebug}
16  202 --              WRITELN('Can''t use clipboard data outside this app');
16  203 --      {$ENDC}
16  204 --              IF message = 0 THEN
16  205 --                  message := msgExportClipFailed;
16  206 --              ShowError(error, message);
16  207 --              GOTO 1000;
16  208 -0 B       END;
16  209 --
16  210 0- A   BEGIN
16  211 --          { Remember when we last started a desk accessory.  UPrinting uses this
16  212 --            to know whether the Chooser may have been run. }
16  213 --          gLastDeskAcc := TickCount;
16  214 --
16  215 --          BusyActivate(FALSE);        { Don't want busy cursor while in desk acc. }
16  216 --
16  217 --          IF convertClipboard THEN
16  218 1-              BEGIN
16  219 --              IF (gLastCommand <> NIL) & gLastCommand.fChangesClipboard THEN
16  220 --                  CommitLastCommand;
16  221 --
16  222 --              IF gClipView <> NIL THEN
16  223 --                  IF NOT gClipWrittenToDeskScrap THEN
16  224 2-                      BEGIN
16  225 --                      err := ZeroScrap;
16  226 --                      CatchFailures(fi, PublicizeFailed);
16  227 --                      gClipView.WriteToDeskScrap;
16  228 --                      Success(fi);
16  229 --                      gClipWrittenToDeskScrap := TRUE;
16  230 --            1000:     AbsorbScrapStuff;   { ??? correct post-error reentry point? }
16  231 -2                    END;
16  232 -1              END;
16  233 -0 A   END;
16  234 --
16  235 --      {$S MARes}
16  236 --      {-----------------------------------+
16  237 --      |   AbsorbScrapStuff                |
16  238 --      +-----------------------------------}
16  239 -- A   PROCEDURE TApplication.AbsorbScrapStuff;
16  240 --      VAR
16  241 --          aPtr:   PScrapStuff;
16  242 0- A   BEGIN
16  243 --          gOldScrapStuff := gNewScrapStuff;  { stash previous version, for later change-checkage }
16  244 --          aPtr := InfoScrap;
16  245 --          gNewScrapStuff := aPtr^; { Copy over from low memory to our private global record }
16  246 -0 A   END;
16  247 --
16  248 --      {$S MAOpen}
16  249 --      {-----------------------------------+
16  250 --      |   AddDocument                     |
16  251 --      +-----------------------------------}
16  252 -- A   PROCEDURE TApplication.AddDocument(aNewDocument: TDocument);
16  253 0- A   BEGIN
16  254 --          gDocList.InsertLast(aNewDocument);
16  255 -0 A   END;
16  256 --
16  257 --      {$S MAOpen}
16  258 --      {-----------------------------------+
16  259 --      |   AddFreeWindow                   |
16  260 --      +-----------------------------------}
16  261 -- A   PROCEDURE TApplication.AddFreeWindow(aWindow: TWindow);
16  262 0- A   BEGIN
16  263 --          gFreeWindowList.InsertLast(aWindow);
16  264 -0 A   END;
16  265 --
16  266 --      {$S MAFile}
16  267 --      {-----------------------------------+
16  268 --      |   AlreadyOpen                     |
16  269 --      +-----------------------------------}
16  270 -- A   FUNCTION  TApplication.AlreadyOpen(fileName: Str255; volRefnum: INTEGER): TDocument;
16  271 --      CONST
16  272 --          ignoreCase = FALSE;
16  273 --          diacritSens = TRUE;
16  274 --
16  275 --      VAR
16  276 --          parmDirID:      LONGINT;
16  277 --          parmVRefnum:    INTEGER;
16  278 --          result:         TDocument;
16  279 --          err:            OSErr;
16  280 --
16  281 --          {-------------------------------+
16  282 --          |   TestDoc                     |
16  283 --          +-------------------------------}
16  284 -- B       PROCEDURE TestDoc(doc: TDocument);
16  285 --          VAR
16  286 --              err:        OSErr;
16  287 --              docVRefnum: INTEGER;
16  288 --              docDirID:   LONGINT;
16  289 0- B       BEGIN
16  290 --              IF result = NIL THEN
16  291 --                  IF doc.fSaveExists THEN
16  292 1-                      BEGIN
16  293 --                      docVRefnum := doc.fVolRefnum;
16  294 --                      err := GetDirID(docVRefnum, docDirID);
16  295 --                      IF err = noErr THEN
16  296 --                          IF docVRefnum = parmVRefnum THEN
```

```
16  297 --                              IF docDirID = parmDirID THEN
16  298 --                                  IF EqualString(fileName, doc.fTitle,
16  299 --                                                          ignoreCase, diacritSens) THEN
16  300 --                                      result := doc;
16  301 -1                      END;
16  302 -0 B       END;
16  303 --
16  304 0- A   BEGIN
16  305 --         result := NIL;
16  306 --
16  307 --         parmVRefnum := volRefnum;
16  308 --         err := GetDirID(parmVRefnum, parmDirID);
16  309 --
16  310 --         IF err = noErr THEN
16  311 --             ForAllDocumentsDo(TestDoc);
16  312 --
16  313 --         AlreadyOpen := result;
16  314 -0 A   END;
16  315 --
16  316 --     {-----------------------------------+
16  317 --     |   CallFileFilter                  |
16  318 --     +-----------------------------------}
16  319 -- A   FUNCTION  CallFileFilter(paramBlock: HParmBlkPtr; routine: ProcPtr): BOOLEAN;
16  320 --         INLINE  $205F,      { MOVEA.L   (A7)+,A0     }
16  321 --                 $4E90;      { JMP       (A0)         }
16  322 --
16  323 --     {$S MAFinder}
16  324 --         { This is called only when opening/printing from the finder; it simulates the
16  325 --             filtering done by Std File. }
16  326 --     {-----------------------------------+
16  327 --     |   CanOpenDocument                 |
16  328 --     +-----------------------------------}
16  329 -- A   FUNCTION  TApplication.CanOpenDocument(itsCmdNumber: CmdNumber; VAR anAppFile: AppFile): BOOLEAN;
16  330 --
16  331 --     VAR
16  332 --         dlgID:          INTEGER;
16  333 --         where:          Point;
16  334 --         fileFilter:     ProcPtr;
16  335 --         dlgHook:        ProcPtr;
16  336 --         filterProc:     ProcPtr;
16  337 --         typeList:       HTypeList;
16  338 --         i:              INTEGER;
16  339 --         paramBlock:     HParamBlockRec;
16  340 --         numTypes:       INTEGER;
16  341 0- A   BEGIN
16  342 --         CanOpenDocument := FALSE;
16  343 --
16  344 --             { First check that file type is in the list of allowed file types.  See SFGetParms
16  345 --                 below for more info. }
16  346 --         typeList := HTypeList(NewHandle(0));
16  347 --         FailNIL(typeList);
16  348 --
16  349 --         SFGetParms(itsCmdNumber, dlgID, where, fileFilter, dlgHook, filterProc, typeList);
16  350 --
16  351 --         numTypes := GetHandleSize(Handle(typeList)) DIV SIZEOF(ResType);
16  352 --         IF numTypes > 0 THEN { if 0 then want all types }
16  353 --             FOR i := 1 TO numTypes DO
16  354 --                     { do coercions because the compiler generates lousy code for comparing 2
16  355 --                         packed arrays of characters }
16  356 --                 IF LONGINT(anAppFile.fType) = LONGINT(typeList^^[i]) THEN
16  357 1-                  BEGIN
16  358 --                     IF fileFilter = NIL THEN
16  359 --                         CanOpenDocument := TRUE
16  360 --                     ELSE IF GetFileInfo(anAppFile.fName,
16  361 --                                         anAppFile.vRefnum, paramBlock) = noErr THEN
16  362 --                         CanOpenDocument := NOT CallFileFilter(@paramBlock, fileFilter)
16  363 --                     ELSE
16  364 --                         CanOpenDocument := FALSE;
16  365 --                     LEAVE;
16  366 -1                  END;
16  367 --
16  368 --         DisposHandle(Handle(typeList));
16  369 -0 A   END;
16  370 --
16  371 --     {$S MARes}
16  372 --     {-----------------------------------+
16  373 --     |   CheckDeskScrap                  |
16  374 --     +-----------------------------------}
16  375 -- A   PROCEDURE TApplication.CheckDeskScrap;
16  376 --     VAR
16  377 --         err:        OSErr;
16  378 --
16  379 0- A   BEGIN
16  380 --         AbsorbScrapStuff;
16  381 --
16  382 --         IF (gOldScrapStuff.scrapCount <> gNewScrapStuff.scrapCount) THEN
16  383 1-             BEGIN
16  384 --             IF gLastCommand <> NIL THEN
16  385 --                 CommitLastCommand;
16  386 --             gClipView.FreeFromClipboard;    { AbandonCurrentClipboard }
16  387 --             gClipView := NIL;               { no reason to have an Undo clipboard }
16  388 --
16  389 --             { If the scrap is in memory and we are low on memory, then we
16  390 --                 dispose of the scrap (ZeroScrap) and use gClipOrphanage as
16  391 --                 the clipboard view. }
16  392 --             IF (gNewScrapStuff.scrapState > 0) AND MemSpaceIsLow THEN
16  393 2-                 BEGIN
16  394 --                 err := ZeroScrap;   { ??? What should we do on an error? }
16  395 --                 AbsorbScrapStuff;
```

```
16  396 --                ShowError(memFullErr, msgImportClipFailed);
16  397 --                ClaimClipboard(gClipOrphanage);
16  398 -2                END
16  399 --            ELSE
16  400 --                ReadFromDeskScrap;
16  401 -1            END;
16  402 -0 A    END;
16  403 --
16  404 --      {$S MAOpen}
16  405 --      {-----------------------------------+
16  406 --      |   ChooseDocument                  |
16  407 --      +-----------------------------------}
16  408 -- A    FUNCTION  TApplication.ChooseDocument(itsCmdNumber: CmdNumber; VAR anAppFile: AppFile): BOOLEAN;
16  409 --      TYPE
16  410 --          HSFTypeList =   ^PSFTypeList;
16  411 --          PSFTypeList =   ^SFTypeList;
16  412 --
16  413 --      VAR
16  414 --          dlgID:          INTEGER;
16  415 --          where:          Point;
16  416 --          fileFilter:     ProcPtr;
16  417 --          dlgHook:        ProcPtr;
16  418 --          filterProc:     ProcPtr;
16  419 --          typeList:       HTypeList;
16  420 --          pTypeList:      PSFTypeList;
16  421 --          numTypes:       INTEGER;
16  422 --          reply:          SFReply;
16  423 0- A    BEGIN
16  424 --          typeList := HTypeList(NewHandle(0));
16  425 --          FailNIL(typeList);
16  426 --
16  427 --          SFGetParms(itsCmdNumber, dlgID, where, fileFilter, dlgHook, filterProc, typeList);
16  428 --          numTypes := GetHandleSize(Handle(typeList)) DIV SIZEOF(ResType);
16  429 --
16  430 --          IF numTypes = 0 THEN
16  431 1-              BEGIN
16  432 --              numTypes := -1;
16  433 --              pTypeList := @pTypeList; { arbitrary, as long as it points to 4 bytes of valid memory }
16  434 -1              END
16  435 --          ELSE
16  436 1-              BEGIN
16  437 --              MoveHHi(Handle(typeList));  { in case Std File does allocations }
16  438 --              HLock(Handle(typeList));    { ditto }
16  439 --              pTypeList := HSFTypeList(typeList)^;
16  440 -1              END;
16  441 --
16  442 --          {$IFC qDebug}
16  443 --          { Causes TApplication.GetEvent to call CheckRsrcUsage. }
16  444 --          gRsrcCheck := 0;
16  445 --          {$ENDC}
16  446 --
16  447 --          UpdateAllWindows; { needed to work around bug in SF; if all windows are
16  448 --                              not updated you wont be able to mount a disk
16  449 --                              correctly }
16  450 --          SFPGetFile(where, '', fileFilter, numTypes, pTypeList^, dlgHook, reply, dlgID, filterProc);
16  451 --
16  452 --          DisposHandle(Handle(typeList));
16  453 --
16  454 --          ChooseDocument := reply.good;
16  455 --          IF reply.good THEN
16  456 1-              BEGIN
16  457 --              anAppFile.vRefnum := reply.vRefnum;
16  458 --              anAppFile.fType := reply.fType;
16  459 --              anAppFile.versNum := reply.version;
16  460 --              anAppFile.fName := reply.fName;
16  461 -1              END;
16  462 -0 A    END;
16  463 --
16  464 --      {$S MAClipboard}
16  465 --      {-----------------------------------+
16  466 --      |   ClaimClipboard                  |
16  467 --      +-----------------------------------}
16  468 -- A    PROCEDURE TApplication.ClaimClipboard(clipView: TView);
16  469 0- A    BEGIN
16  470 --          AbandonUndoClipboard;  { free up any old UNDO stuff }
16  471 --          gClipUndoView := gClipView; { Copy current clipboard contents to the Undo side }
16  472 --          IF clipView <> NIL THEN
16  473 1-              BEGIN
16  474 --      {$IFC FALSE}
16  475 --      {$IFC qDebug}
16  476 --              WITH clipView DO
16  477 2-                  BEGIN
16  478 --                  fShowBorders := TRUE; { ??? }  { temp }
16  479 --                  fShowExtraFeedback := TRUE; { ??? }  { temp }
16  480 -2                  END;
16  481 --      {$ENDC}
16  482 --      {$ENDC}
16  483 --              SetClipView(clipView);  { Will install it as gClipView }
16  484 -1              END
16  485 --          ELSE
16  486 1-              BEGIN
16  487 --      {$IFC qDebug}
16  488 --              ProgramBreak('Claiming clipboard with null view');
16  489 --      {$ENDC}
16  490 -1              END;
16  491 --
16  492 --          gClipClaimed := TRUE;
16  493 -0 A    END;
16  494 --
```

```
16  495 --      {$S MATerminate}
16  496 --      {-----------------------------------+
16  497 --      |   Close                           |
16  498 --      +-----------------------------------}
16  499 -- A    PROCEDURE TApplication.Close;
16  500 --      VAR
16  501 --          wmgrWindow: WindowPtr;
16  502 --
16  503 --          {--------------------------------+
16  504 --          |   FreeIt                       |
16  505 --          +--------------------------------}
16  506 -- B      PROCEDURE FreeIt(anEvtHandler: TEvtHandler);
16  507 0- B      BEGIN
16  508 --          anEvtHandler.Free;  { ??? also call Terminate ??? }
16  509 -0 B      END;
16  510 --
16  511 --          {--------------------------------+
16  512 --          |   CloseADocument               |
16  513 --          +--------------------------------}
16  514 -- B      PROCEDURE CloseADocument (aDocument: TDocument);
16  515 --
16  516 0- B      BEGIN
16  517 --          aDocument.Close;
16  518 -0 B      END;
16  519 --
16  520 0- A    BEGIN
16  521 --          { Close all of the windows }
16  522 1-          REPEAT
16  523 --              wmgrWindow := FrontWindow;
16  524 --              IF wmgrWindow <> NIL THEN
16  525 --                  CloseWmgrWindow(wmgrWindow);
16  526 -1          UNTIL wmgrWindow = NIL;
16  527 --
16  528 --          { Close any windowless documents }
16  529 --          ForAllDocumentsDo(CloseADocument);
16  530 --
16  531 --          gPrintHandler.Terminate;
16  532 --          EachHandler(gHeadCoHandler, FreeIt);
16  533 --          IF LoadScrap <> noErr THEN; { ??? }
16  534 -0 A    END;
16  535 --
16  536 --      {$S MAClose}
16  537 --      {-----------------------------------+
16  538 --      |   CloseWmgrWindow                 |
16  539 --      +-----------------------------------}
16  540 -- A    PROCEDURE TApplication.CloseWmgrWindow(aWmgrWindow: WindowPtr);
16  541 --      VAR
16  542 --          aWindow:    TWindow;
16  543 0- A    BEGIN
16  544 --          IF IsDeskAccessory(aWmgrWindow) THEN
16  545 --              CloseDeskAcc(WindowPeek(aWmgrWindow)^.windowKind)
16  546 --          ELSE
16  547 1-              BEGIN
16  548 --              aWindow := WMgrToWindow(aWMgrWindow);
16  549 --              IF aWindow <> NIL THEN
16  550 --                  aWindow.CloseByUser
16  551 --              ELSE
16  552 --                  HideWindow(aWmgrWindow);
16  553 -1              END;
16  554 -0 A    END;
16  555 --
16  556 --      {$S MARes}
16  557 --      {-----------------------------------+
16  558 --      |   CommitLastCommand               |
16  559 --      +-----------------------------------}
16  560 -- A    PROCEDURE TApplication.CommitLastCommand;
16  561 0- A    BEGIN
16  562 --          AbandonUndoClipboard;
16  563 --
16  564 --          IF gLastCommand <> NIL THEN
16  565 1-              BEGIN
16  566 --              IF gLastCommand.fCmdDone THEN
16  567 --                  gLastCommand.Commit;
16  568 --              gLastCommand.Free;
16  569 --              gLastCommand := NIL;
16  570 -1              END;
16  571 -0 A    END;
16  572 --
16  573 --      {$S MARes}
16  574 --      {-----------------------------------+
16  575 --      |   CountClicks                     |
16  576 --      +-----------------------------------}
16  577 -- A    FUNCTION  TApplication.CountClicks(aPDownEvent:     PEventRecord;
16  578 --                                         whereMouseDown:  INTEGER): INTEGER;
16  579 --
16  580 --      VAR
16  581 --          clickCount:     INTEGER;
16  582 --
16  583 0- A    BEGIN
16  584 --          clickCount := 1;
16  585 --
16  586 --          WITH aPDownEvent^ DO
16  587 1-              BEGIN
16  588 --              { This series of IF's generates less code than short-circuit booleans }
16  589 --              IF whereMouseDown = gLastClickPart THEN
16  590 --                  IF gClickCount > 0 THEN { not the first click and ... }
16  591 --                      IF when - gLastUpTime < GetDblTime THEN { ... close enough in time and ... }
16  592 --                          IF gTarget.DoMultiClick(gLastMsePt, where) { ... close enough in space } THEN
16  593 --                              clickCount := gClickCount + 1;
```

```
16  594 --
16  595 --              gLastMsePt := where;
16  596 -1              END;
16  597 --
16  598 --          gLastClickPart := whereMouseDown;
16  599 --          gClickCount := clickCount;
16  600 --          CountClicks := clickCount;
16  601 -0 A    END;
16  602 --
16  603 --      {$S MAClose}
16  604 --      {-----------------------------------+
16  605 --      |   DeleteDocument                  |
16  606 --      +-----------------------------------}
16  607 -- A    PROCEDURE TApplication.DeleteDocument(docToDelete: TDocument);
16  608 0- A    BEGIN
16  609 --          gDocList.Delete(docToDelete);
16  610 --          IF gDocument = docToDelete THEN
16  611 --              gDocument := NIL;
16  612 -0 A    END;
16  613 --
16  614 --      {$S MAClose}
16  615 --      {-----------------------------------+
16  616 --      |   DeleteFreeWindow                |
16  617 --      +-----------------------------------}
16  618 -- A    PROCEDURE TApplication.DeleteFreeWindow(windowToDelete: TWindow);
16  619 0- A    BEGIN
16  620 --          gFreeWindowList.Delete(windowToDelete);
16  621 -0 A    END;
16  622 --
16  623 --      {$S MARes}
16  624 --      {-----------------------------------+
16  625 --      |   DispatchEvent                   |
16  626 --      +-----------------------------------}
16  627 -- A    PROCEDURE TApplication.DispatchEvent(VAR theEventInfo: EventInfo; VAR commandToPerform: TCommand);
16  628 --
16  629 0- A    BEGIN
16  630 --          commandToPerform := gNoChanges;
16  631 --
16  632 --          WITH theEventInfo.thePEvent^ DO
16  633 1-              BEGIN
16  634 --
16  635 2-              CASE what OF
16  636 --                  mouseUp:
16  637 --                      commandToPerform := HandleMouseUp(theEventInfo);
16  638 --
16  639 --                  mouseDown:
16  640 --                      commandToPerform := HandleMouseDown(theEventInfo);
16  641 --
16  642 --                  activateEvt:
16  643 --                      commandToPerform := HandleActivateEvent(theEventInfo);
16  644 --
16  645 --                  updateEvt:
16  646 --                      commandToPerform := HandleUpdateEvent(theEventInfo);
16  647 --
16  648 --                  keyDown, autoKey:
16  649 --                      commandToPerform := HandleKeyDownEvent(theEventInfo);
16  650 --
16  651 --                  diskEvt:
16  652 --                      commandToPerform := HandleDiskEvent(theEventInfo);
16  653 --
16  654 --                  app4Evt:
16  655 --                      { All app4Evt's are owned by the system }
16  656 --                      commandToPerform := HandleSystemEvent(theEventInfo);
16  657 --
16  658 --                  OTHERWISE
16  659 --                      commandToPerform := HandleAlienEvent(theEventInfo);
16  660 --
16  661 -2                  END; { case }
16  662 --
16  663 -1              END; { with }
16  664 -0 A    END;
16  665 --
16  666 --      {$S MARes}
16  667 --      {-----------------------------------+
16  668 --      |   DoCommandKey                    |
16  669 --      +-----------------------------------}
16  670 -- A    FUNCTION  TApplication.DoCommandKey (ch: CHAR; VAR info: EventInfo): TCommand; OVERRIDE;
16  671 0- A    BEGIN
16  672 --          DoCommandKey := gNoChanges;
16  673 --          IF (NOT info.theAutoKey) & (NOT InModalMenuState) THEN
16  674 1-              BEGIN
16  675 --              SetupTheMenus;
16  676 --              DoCommandKey := MenuEvent(MenuKey(ch));
16  677 -1              END;
16  678 -0 A    END;
16  679 --
16  680 --      {$S MANever}
16  681 --      {-----------------------------------+
16  682 --      |   DoMakeDocument                  |
16  683 --      +-----------------------------------}
16  684 -- A    FUNCTION  TApplication.DoMakeDocument(itsCmdNumber: CmdNumber): TDocument;
16  685 --      (*  VAR aYOURDocument:  TDocument; *)
16  686 0- A    BEGIN
16  687 --      (*
16  688 --          New(aYOURDocument);
16  689 --          aYOURDocument.IYOURDocument(itsDocKind, ...);
16  690 --          DoMakeDocument := aYOURDocument;
16  691 --      *)
16  692 --      {$IFC qDebug}
```

```
16  693 --          ProgramBreak('You forgot to override TApplication.DoMakeDocument in your application');
16  694 --      {$ENDC}
16  695 -0 A    END;
16  696 --
16  697 --      {$S MASelCommand}
16  698 --      {-----------------------------------+
16  699 --      |   DoMenuCommand                   |
16  700 --      +-----------------------------------}
16  701 -- A    FUNCTION  TApplication.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
16  702 --      VAR
16  703 --          succeeded:      BOOLEAN;
16  704 --          aDocument:      TDocument;
16  705 --          deltaCount:     INTEGER;
16  706 --          aWindow:        TWindow;
16  707 --          anAppFile:      AppFile;
16  708 --          fi:             FailInfo;
16  709 --
16  710 --          {-------------------------------+
16  711 --          |   HdlQuit                     |
16  712 --          +-------------------------------}
16  713 -- B        PROCEDURE HdlQuit(error: OSErr; message: LONGINT);
16  714 0- B       BEGIN
16  715 --              gAppDone := FALSE;
16  716 -0 B       END;
16  717 --
16  718 0- A    BEGIN
16  719 --          DoMenuCommand := gNoChanges;
16  720 --
16  721 1-          CASE aCmdNumber OF
16  722 --              cQuit:
16  723 2-                  BEGIN
16  724 --                  CatchFailures(fi, HdlQuit);
16  725 --                  gAppDone := TRUE;
16  726 --                  Close;
16  727 --                  Success(fi);
16  728 -2                  END;
16  729 --
16  730 --              cNew..cNewLast:
16  731 --                  OpenNew(aCmdNumber);
16  732 --
16  733 --              cOpen..cOpenLast:
16  734 2-                  BEGIN
16  735 --                  IF ChooseDocument(aCmdNumber, anAppFile) THEN
16  736 --                      OpenOld(aCmdNumber, anAppFile);
16  737 -2                  END;
16  738 --
16  739 --              cClose:
16  740 --                  CloseWmgrWindow(FrontWindow);
16  741 --
16  742 --              cShowClipboard:
16  743 --                  IF FrontWIndow = gClipWindow.fWMgrWindow THEN
16  744 --                      gClipWindow.Close { Hide the clipboard }
16  745 --                  ELSE
16  746 2-                      BEGIN
16  747 --                      gClipWindow.Open;
16  748 --                      gClipWindow.Select;
16  749 -2                      END;
16  750 --
16  751 --              cAboutApp:
16  752 --                  StdAlert(phAboutApp);
16  753 --
16  754 --      {$IFC qWWNeeded}
16  755 --              cDebugWind:
16  756 2-                  BEGIN
16  757 --                  ShowWindow(gDebugWindowPtr);
16  758 --                  SelectWMgrWindow(gDebugWindowPtr);
16  759 -2                  END;
16  760 --      {$ENDC}
16  761 --
16  762 --      {$IFC qDebug}
16  763 --              cExperimenting:
16  764 --                  gExperimenting := NOT gExperimenting;
16  765 --              cReportEvt:
16  766 --                  gReportEvt := NOT gReportEvt;
16  767 --              cDebugPrinting:
16  768 --                  gDebugPrinting := NOT gDebugPrinting;
16  769 --              cReportMenuChoices:
16  770 --                  gReportMenuChoices := NOT gReportMenuChoices;
16  771 --              cIntenseDebugging:
16  772 --                  gIntenseDebugging := NOT gIntenseDebugging;
16  773 --              cIdentifySoftware:
16  774 2-                  BEGIN
16  775 --                  Writeln('===== Software Version(s): =====');
16  776 --                  gTarget.IdentifySoftware;
16  777 -2                  END;
16  778 --              cRefreshFrontWindow:
16  779 --                  gFrontWindow.ForceRedraw;
16  780 --              cModalToggle:
16  781 --                  gFrontWindow.fIsModal := NOT gFrontWindow.fIsModal;
16  782 --              cDoFirstClick:
16  783 --                  gFrontWindow.fDoFirstClick := NOT gFrontWindow.fDoFirstClick;
16  784 --      {$ENDC}
16  785 --
16  786 --      {$IFC qTrace}
16  787 --              cTraceSetupMenus:
16  788 --                  gTraceSetupMenus := NOT gTraceSetupMenus;
16  789 --              cTraceIdle:
16  790 --                  gTraceIdle := NOT gTraceIdle;
16  791 --      {$ENDC}
```

```
16  792 --
16  793 --      {$IFC qInspector}
16  794 --              cNewInspectorWindow:
16  795 --                  MakeInspectorWindow;
16  796 --      {$ENDC}
16  797 --
16  798 --              cUndo { , cRedo }:
16  799 2-                  BEGIN
16  800 --                  IF gLastCommand.fChangesClipboard THEN
16  801 --                      SwapClipViews;
16  802 --
16  803 --                  IF gLastCommand.fCmdDone THEN
16  804 3-                      BEGIN
16  805 --                      gLastCommand.UndoIt;
16  806 --                      deltaCount := -1;
16  807 -3                      END
16  808 --                  ELSE
16  809 3-                      BEGIN
16  810 --                      gLastCommand.RedoIt;
16  811 --                      deltaCount := 1;
16  812 -3                      END;
16  813 --
16  814 --                  gLastCommand.fCmdDone := NOT gLastCommand.fCmdDone;
16  815 --
16  816 --                  IF gLastCommand.fCausesChange THEN { put this after .UndoIt/.RedoIt,
16  817 --                                                       so they can change the flag }
16  818 --                      WITH gLastCommand DO
16  819 --                          IF fChangedDocument <> NIL THEN
16  820 --                              WITH fChangedDocument DO
16  821 3-                                  BEGIN
16  822 --                                  fChangeCount := fChangeCount + deltaCount;
16  823 --                                  {$IFC qDebug}
16  824 --                                  IF fChangeCount < 0 THEN
16  825 --                                      ProgramBreak('fChangeCount < 0');
16  826 --                                  {$ENDC}
16  827 -3                                  END;
16  828 -2                  END;
16  829 --
16  830 --              OTHERWISE
16  831 --                      DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
16  832 -1              END; { case }
16  833 -0 A   END;
16  834 --
16  835 --      {$S MARes}
16  836 --      {-----------------------------------+
16  837 --      |   DoSetupMenus                    |
16  838 --      +-----------------------------------}
16  839 -- A   PROCEDURE TApplication.DoSetupMenus;
16  840 --      VAR
16  841 --          frontmost:      WindowPtr;
16  842 --
16  843 0- A   BEGIN
16  844 --          INHERITED DoSetupMenus;
16  845 --
16  846 --          Enable(cAboutApp, TRUE);
16  847 --          Enable(cQuit, gEventLevel <= 1);        { Can't enable Quit if in nested event handling }
16  848 --          Enable(cShowClipboard, TRUE);
16  849 --          SetMenuState(cShowClipboard, kIDBuzzString, bzShowClip, bzHideClip,
16  850 --                       FrontWindow = gClipWindow.fWMgrWindow);
16  851 --
16  852 --          Enable(cNew, TRUE);
16  853 --          Enable(cOpen, TRUE);
16  854 --
16  855 --      {$IFC qWriteLnWin}
16  856 --          Enable(cDebugWind, gDebugWindowPtr <> NIL);
16  857 --      {$ENDC}
16  858 --
16  859 --      {$IFC qInspector}
16  860 --          Enable(cNewInspectorWindow, TRUE);
16  861 --      {$ENDC}
16  862 --
16  863 --          frontmost := FrontWindow;
16  864 --          IF frontmost <> NIL THEN
16  865 --              Enable(cClose, WindowPeek(frontmost)^.goAwayFlag <> FALSE);
16  866 -0 A   END;
16  867 --
16  868 --      {$S MARes}
16  869 --      {-----------------------------------+
16  870 --      |   EachFreeWindow                  |
16  871 --      +-----------------------------------}
16  872 -- A   PROCEDURE TApplication.EachFreeWindow(PROCEDURE DoToWindow(aWindow: TWindow));
16  873 0- A   BEGIN
16  874 --          gFreeWindowList.Each(DoToWindow);
16  875 -0 A   END;
16  876 --
16  877 --      {$S MARes}
16  878 --      {-----------------------------------+
16  879 --      |   EachHandler                     |
16  880 --      +-----------------------------------}
16  881 -- A   PROCEDURE TApplication.EachHandler(aFirstHandler: TEvtHandler;
16  882 --                                        PROCEDURE DoToEvtHandler(anEvtHandler: TEvtHandler));
16  883 --          { Performs 'DoToEvtHandler' to aFirstHandler, then to its fNextHandler, etc.,
16  884 --              onward until the fNextHandler chain ends at NIL }
16  885 --
16  886 --      VAR
16  887 --          currHandler:    TEvtHandler;
16  888 --          nextHandler:    TEvtHandler;
16  889 --
16  890 0- A   BEGIN
```

```
16  891 --          currHandler := aFirstHandler;
16  892 --          WHILE currHandler <> NIL DO
16  893 1-              BEGIN
16  894 --              { Get next handler now, in case DoToEvtHandler frees currHandler }
16  895 --              nextHandler := currHandler.fNextHandler;
16  896 --              DoToEvtHandler(currHandler);
16  897 --              currHandler := nextHandler;
16  898 -1              END;
16  899 -0 A    END;
16  900 --
16  901 --      {$IFC qDebug}
16  902 --      {$S MAFields}
16  903 --      {-----------------------------------+
16  904 --      |   Fields                          |
16  905 --      +-----------------------------------}
16  906 -- A    PROCEDURE TApplication.Fields (PROCEDURE DoToField (fieldName: Str255;
16  907 --                                                         fieldAddr: Ptr;
16  908 --                                                         fieldType: INTEGER)); OVERRIDE;
16  909 --
16  910 0- A    BEGIN
16  911 --          DoToField('TApplication', NIL, bClass);
16  912 --          DoToField('gAppDone', @gAppDone, bBoolean);
16  913 --          DoToField('gApplication', @gApplication, bObject);
16  914 --          DoToField('gBusyTempRgn', @gBusyTempRgn, bBoolean);
16  915 --          DoToField('gChooserOK', @gChooserOK, bBoolean);
16  916 --          DoToField('gClickCount', @gClickCount, bInteger);
16  917 --          DoToField('gClipClaimed', @gClipClaimed, bBoolean);
16  918 --          DoToField('gClipOrphanage', @gClipOrphanage, bObject);
16  919 --          DoToField('gClipUndoView', @gClipUndoView, bObject);
16  920 --          DoToField('gClipView', @gClipView, bObject);
16  921 --          DoToField('gClipWindow', @gClipWindow, bObject);
16  922 --          DoToField('gClipWrittenToDeskScrap', @gClipWrittenToDeskScrap, bBoolean);
16  923 --          DoToField('gCouldPrint', @gCouldPrint, bBoolean);
16  924 --          DoToField('gCurrPrintHandler', @gCurrPrintHandler, bObject);
16  925 --          DoToField('gCursorRgn', @gCursorRgn, bRgnHandle);
16  926 --          DoToField('gDebugPrinting', @gDebugPrinting, bBoolean);
16  927 --          DoToField('gDocument', @gDocument, bObject);
16  928 --          DoToField('gDocList', @gDocList, bObject);
16  929 --          DoToField('gDrawingPictScrap', @gDrawingPictScrap, bBoolean);
16  930 --          DoToField('gErrorParm3', @gErrorParm3, bString);
16  931 --          DoToField('gEventLevel', @gEventLevel, bInteger);
16  932 --          DoToField('gExperimenting', @gExperimenting, bBoolean);
16  933 --          DoToField('gFileCount', @gFileCount, bInteger);
16  934 --          DoToField('gFinderPrinting', @gFinderPrinting, bBoolean);
16  935 --          DoToField('gFocusedView', @gFocusedView, bObject);
16  936 --          DoToField('gFreeWindowList', @gFreeWindowList, bObject);
16  937 --          DoToField('gFrontWindow', @gFrontWindow, bObject);
16  938 --          DoToField('gGotClipType', @gGotClipType, bBoolean);
16  939 --          DoToField('gHeadCohandler', @gHeadCohandler, bObject);
16  940 --          DoToField('gIdleFreq', @gIdleFreq, bLongInt);
16  941 --          DoToField('gIdlePhase', @gIdlePhase, bByte);
16  942 --          DoToField('gInBackground', @gInBackground, bBoolean);
16  943 --          DoToField('gInitialized', @gInitialized, bBoolean);
16  944 --          DoToField('gIntenseDebugging', @gIntenseDebugging, bBoolean);
16  945 --          DoToField('gLastClickPart', @gLastClickPart, bInteger);
16  946 --          DoToField('gLastCommand', @gLastCommand, bObject);
16  947 --          DoToField('gLastDeskAcc', @gLastDeskAcc, bLongInt);
16  948 --          DoToField('gLastIdle', @gLastIdle, bLongInt);
16  949 --          DoToField('gLastMsePt', @gLastMsePt, bPoint);
16  950 --          DoToField('gLastUpTime', @gLastUpTime, bLongInt);
16  951 --          DoToField('gLongOffset', @gLongOffset, bVPoint);
16  952 --          DoToField('gLowSpaceInterval', @gLowSpaceInterval, bLongInt);
16  953 --          DoToField('gMainEventMask', @gMainEventMask, bHexInteger);
16  954 --          DoToField('gMainFileType', @gMainFileType, bOSType);
16  955 --          DoToField('gMBarDisplayed', @gMBarDisplayed, bInteger);
16  956 --          DoToField('gMBarHeight', @gMBarHeight, bInteger);
16  957 --          DoToField('gMBarHierarchical', @gMBarHierarchical, bInteger);
16  958 --          DoToField('gMBarNotDisplayed', @gMBarNotDisplayed, bInteger);
16  959 --          DoToField('gMenusAreSetup', @gMenusAreSetup, bBoolean);
16  960 --          DoToField('gNextSpaceMsg', @gNextSpaceMsg, bLongInt);
16  961 --          DoToField('gNoChanges', @gNoChanges, bObject);
16  962 --          DoToField('gNullPrintHandler', @gNullPrintHandler, bObject);
16  963 --          DoToField('gNumUntitled', @gNumUntitled, bInteger);
16  964 --          DoToField('gOldChooserFlag', @gOldChooserFlag, bBoolean);
16  965 --          DoToField('gOrthogonal[h]', @gOrthogonal[h], bByte);
16  966 --          DoToField('gOrthogonal[v]', @gOrthogonal[v], bByte);
16  967 --          DoToField('gPageOffset', @gPageOffset, bVPoint);
16  968 --          DoToField('gPrefClipType', @gPrefClipType, bOSType);
16  969 --          DoToField('gPrintHandler', @gPrintHandler, bObject);
16  970 --          DoToField('gPrinting', @gPrinting, bBoolean);
16  971 --      {$IFC qTemplateViews}
16  972 --          DoToField('gProtoList', @gProtoList, bObject);
16  973 --      {$ENDC}
16  974 --          DoToField('gRedrawMenuBar', @gRedrawMenuBar, bBoolean);
16  975 --          DoToField('gReportEvt', @gReportEvt, bBoolean);
16  976 --          DoToField('gReportMenuChoices', @gReportMenuChoices, bBoolean);
16  977 --          DoToField('gRsrcCheck', @gRsrcCheck, bBoolean);
16  978 --          DoToField('gSaveClip', @gSaveClip, bHandle);
16  979 --          DoToField('gSaveFocusedView', @gSaveFocusedView, bObject);
16  980 --          DoToField('gSaveLongOffset', @gSaveLongOffset, bVPoint);
16  981 --          DoToField('gSaveOrg', @gSaveOrg, bPoint);
16  982 --          DoToField('gSavePort', @gSavePort, bWindowPtr);
16  983 --      {$IFC qTemplateViews}
16  984 --          DoToField('gSignatureCount', @gSignatureCount, bInteger);
16  985 --      {$ENDC}
16  986 --          DoToField('gStdHysteresis', @gStdHysteresis, bPoint);
16  987 --          DoToField('gStdStaggerCount', @gStdStaggerCount, bInteger);
16  988 --          DoToField('gStdWMoveBounds', @gStdWMoveBounds, bRect);
16  989 --          DoToField('gStdWSizeRect', @gStdWSizeRect, bRect);
```

```
16  990 --          DoToField('gStdWScreenRect', @gStdWScreenRect, bRect);
16  991 --          DoToField('gSysWindowActive', @gSysWindowActive, bBoolean);
16  992 --          DoToField('gSystemStyle', @gSystemStyle, bTitle);
16  993 --          DoToField('  tsFont', @gSystemStyle.tsFont, bInteger);
16  994 --          DoToField('  tsFace', @gSystemStyle.tsFace, bStyle);
16  995 --          DoToField('  tsSize', @gSystemStyle.tsSize, bInteger);
16  996 --          DoToField('  tsColor', @gSystemStyle.tsColor, bRGBcolor);
16  997 --          DoToField('gTarget', @gTarget, bObject);
16  998 --          DoToField('gTempRgn', @gTempRgn, bHandle);
16  999 --      {$IFC qTrace}
16 1000 --          DoToField('gTraceIdle', @gTraceIdle, bBoolean);
16 1001 --      {$ENDC}
16 1002 --          DoToField('gUndoState', @gUndoState, bBoolean);
16 1003 --          DoToField('gUndoCmd', @gUndoCmd, bInteger);
16 1004 --          DoToField('gUsedBy', @gUsedBy, bString);
16 1005 --          DoToField('gVarClipPicSize', @gVarClipPicSize, bBoolean);
16 1006 --          DoToField('gWorkPort', @gWorkPort, bGrafPtr);
16 1007 --          DoToField('gWResSignature', @gWResSignature, bOSType);
16 1008 --          DoToField('gWResType', @gWResType, bString);
16 1009 --          DoToField('gZeroRect', @gZeroRect, bRect);
16 1010 --          DoToField('gZeroVPt', @gZeroVPt, bVRect);
16 1011 --          DoToField('gZeroVRect', @gZeroVRect, bVRect);
16 1012 --
16 1013 --          INHERITED Fields(DoToField);
16 1014 -0 A   END { TApplication.Fields };
16 1015 --      {$ENDC}
16 1016 --
16 1017 --      {$S MARes}
16 1018 --      {-----------------------------------+
16 1019 --      |   FirstHandlerThat                |
16 1020 --      +-----------------------------------}
16 1021 -- A   FUNCTION  TApplication.FirstHandlerThat(aFirstHandler: TEvtHandler;
16 1022 --              FUNCTION  TestEvtHandler(anEvtHandler: TEvtHandler): BOOLEAN): TEvtHandler;
16 1023 --          { Calls TestEvtHandler for each event handler until TestEvtHandler returns
16 1024 --              true.  Returns the TEvtHandler object for which TestEvtHandler returned
16 1025 --              true, or NIL if TestEvtHandler never returned true. }
16 1026 --
16 1027 --      VAR
16 1028 --          currHandler:    TEvtHandler;
16 1029 --          nextHandler:    TEvtHandler;
16 1030 --
16 1031 0- A   BEGIN
16 1032 --          currHandler := aFirstHandler;
16 1033 --          WHILE currHandler <> NIL DO
16 1034 1-              BEGIN
16 1035 --              { Get next handler now, in case DoToEvtHandler free currHandler }
16 1036 --              nextHandler := currHandler.fNextHandler;
16 1037 --              IF TestEvtHandler(currHandler) THEN
16 1038 2-                  BEGIN
16 1039 --                  FirstHandlerThat := currHandler;
16 1040 --                  EXIT(FirstHandlerThat);
16 1041 -2                  END;
16 1042 --              currHandler := nextHandler;
16 1043 -1              END;
16 1044 --          FirstHandlerThat := NIL;
16 1045 -0 A   END;
16 1046 --
16 1047 --      {$S MARes}
16 1048 --      {-----------------------------------+
16 1049 --      |   ForAllDocumentsDo               |
16 1050 --      +-----------------------------------}
16 1051 -- A   PROCEDURE TApplication.ForAllDocumentsDo(PROCEDURE DoToDoc(aDocument: TDocument));
16 1052 0- A   BEGIN
16 1053 --          gDocList.Each(DoToDoc);
16 1054 -0 A   END;
16 1055 --
16 1056 --      {$S MARes}
16 1057 --      {-----------------------------------+
16 1058 --      |   ForAllWindowsDo                 |
16 1059 --      +-----------------------------------}
16 1060 -- A   PROCEDURE TApplication.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
16 1061 --          {-------------------------------+
16 1062 --          |   DoToYourWindows             |
16 1063 --          +-------------------------------}
16 1064 -- B       PROCEDURE DoToYourWindows(aDocument: TDocument);
16 1065 0- B           BEGIN
16 1066 --              aDocument.ForAllWindowsDo(DoToWind);
16 1067 -0 B           END;
16 1068 0- A   BEGIN
16 1069 --          ForAllDocumentsDo(DoToYourWindows);
16 1070 --          EachFreeWindow(DoToWind);
16 1071 -0 A   END;
16 1072 --
16 1073 --      {$IFC qInspector}
16 1074 --      {$S MAInspector}
16 1075 --      {-----------------------------------+
16 1076 --      |   GetInspectorName                |
16 1077 --      +-----------------------------------}
16 1078 -- A   PROCEDURE TApplication.GetInspectorName (VAR inspectorName: Str255);
16 1079 --
16 1080 0- A   BEGIN
16 1081 --          IF SELF = gApplication THEN
16 1082 --              inspectorName := 'gApplication';
16 1083 -0 A   END;
16 1084 --      {$ENDC}
16 1085 --
16 1086 --      {$S MAClipboard}
16 1087 --      {-----------------------------------+
16 1088 --      |   GetDataToPaste                  |
```

```
16 1089 --      +-----------------------------------}
16 1090 -- A    FUNCTION  TApplication.GetDataToPaste(aDataHandle: Handle; VAR dataType: ResType): LONGINT;
16 1091 --      VAR
16 1092 --          err:        LONGINT;
16 1093 --          myType:     ResType;
16 1094 --
16 1095 0- A    BEGIN
16 1096 --          IF gGotClipType THEN
16 1097 1-              BEGIN
16 1098 --              dataType := gPrefClipType;
16 1099 --
16 1100 --              err := gClipView.GivePasteData(aDataHandle, dataType);
16 1101 --
16 1102 --              IF err < 0 THEN
16 1103 --                  Failure(err, 0);
16 1104 -1             END
16 1105 --          ELSE
16 1106 1-              BEGIN
16 1107 --      {$IFC qDebug}
16 1108 --              ProgramBreak('GetDataToPaste called when gGotClipType was FALSE');
16 1109 --      {$ENDC}
16 1110 -1              END;
16 1111 --
16 1112 --          GetDataToPaste := err;
16 1113 -0 A    END;
16 1114 --
16 1115 --      {$S MARes}
16 1116 --      {-----------------------------------+
16 1117 --      |   GetEvent                        |
16 1118 --      +-----------------------------------}
16 1119 -- A    FUNCTION  TApplication.GetEvent(eventMask: INTEGER; sleep: LONGINT; cursorRgn: RgnHandle;
16 1120 --                                      VAR anEvent: EventRecord): BOOLEAN;
16 1121 --
16 1122 --      VAR
16 1123 --          haveEvent:      BOOLEAN;
16 1124 --          dummyEvent:     EventRecord;
16 1125 --
16 1126 0- A    BEGIN
16 1127 --
16 1128 --      {$IFC qDebug}
16 1129 --          gRsrcCheck := gRsrcCheck - 1;
16 1130 --          IF gRsrcCheck <= 0 THEN
16 1131 1-              BEGIN
16 1132 --              CheckRsrcUsage;
16 1133 --              gRsrcCheck := kRsrcCheckInterval;
16 1134 -1              END;
16 1135 --      {$ENDC qDebug}
16 1136 --
16 1137 --          IF gConfiguration.hasWaitNextEvent THEN
16 1138 1-              BEGIN
16 1139 --      {$IFC qDebug}
16 1140 --              IF gIntenseDebugging & gReportEvt THEN
16 1141 2-                  BEGIN
16 1142 --                  WRITE('WaitNextEvent: sleep=', sleep:0);
16 1143 --                  IF cursorRgn = NIL THEN
16 1144 --                      WRITE(', cursor region=NIL')
16 1145 --                  ELSE
16 1146 --                      WrLblRect(', cursor', cursorRgn^^.rgnBBox);
16 1147 --                  WRITELN;
16 1148 -2                  END;
16 1149 --      {$ENDC}
16 1150 --              BusyActivate(FALSE);                        { Turn off busy cursor while we're away.}
16 1151 --
16 1152 --              IF sleep > 0 THEN
16 1153 --                  IF NOT EventAvail(eventMask, anEvent) AND   { If no events pending, we may not come }
16 1154 --                     (gIdlePhase = idleBegin) THEN            { _back for a while, so idle to make    }
16 1155 2-                     BEGIN
16 1156 --                     Idle(idleBegin);                         { _sure menus, etc. reflect reality.    }
16 1157 --                     gIdlePhase := idleContinue;
16 1158 -2                     END;
16 1159 --
16 1160 --              haveEvent := WaitNextEvent(eventMask, anEvent, sleep, cursorRgn);
16 1161 --
16 1162 --              { Here we decide what to do if we get a mouse-moved event.  If there are NO pending
16 1163 --                events then we don't consider it an event and idle.  If there ARE pending events,
16 1164 --                then we consider it an event in order to avoid idling. }
16 1165 --
16 1166 --              WITH anEvent DO
16 1167 --                  IF (what = app4Evt) &                   { Is this a mouse-moved event?          }
16 1168 --                     (BAND(BSR(message, 24), $FF) = kMouseMovedMessage) &
16 1169 --                     (NOT EventAvail(eventMask, dummyEvent)) THEN
16 1170 2-                     BEGIN
16 1171 --      {$IFC qDebug}
16 1172 --                     IF gReportEvt THEN
16 1173 --                         WriteLn('Mouse moved');
16 1174 --      {$ENDC}
16 1175 --                     haveEvent := FALSE;                     { _but don't consider it a real event.  }
16 1176 -2                     END;
16 1177 --
16 1178 --              IF NOT gInBackground THEN                   { If we're not in the background, then  }
16 1179 --                  BusyActivate(TRUE);                     { _enable the busy cursor mechanism.    }
16 1180 -1              END
16 1181 --
16 1182 --          ELSE
16 1183 --
16 1184 1-              BEGIN
16 1185 --              SystemTask;
16 1186 --              haveEvent := GetNextEvent(eventMask, anEvent);
16 1187 --      {$IFC FALSE}        { this is code for measuring event loop performance }
```

```
16 1188 --              IF anEvent.what <> keyDown THEN
16 1189 --                  IF gIdlePhase <> idleBegin THEN
16 1190 2-                      BEGIN
16 1191 --                      haveEvent := TRUE;
16 1192 --                      anEvent.what := keyDown;
16 1193 --                      anEvent.modifiers := 0;
16 1194 -2                     END
16 1195 --                  ELSE
16 1196 --                      haveEvent := FALSE;
16 1197 --      {$ENDC}
16 1198 -1              END;
16 1199 --
16 1200 --          GetEvent := haveEvent;
16 1201 --
16 1202 -0 A    END { TApplication.GetEvent };
16 1203 --
16 1204 --      {$S MAOpen}
16 1205 --          { ??? We should not muck with the window template; the extra code isn't worth it
16 1206 --              since programmer can easily change the resource file ??? }
16 1207 --      {-----------------------------------+
16 1208 --      |   GetRsrcWindow                   |
16 1209 --      +-----------------------------------}
16 1210 -- A    FUNCTION  TApplication.GetRsrcWindow(storage: Ptr; rsrcId: INTEGER;
16 1211 --                                  VAR isResizable, isClosable: BOOLEAN): WindowPtr;
16 1212 --          { We force INVISIBLE in the WIND definition so the screen won't flash. }
16 1213 --
16 1214 --      TYPE
16 1215 --          WindowTemplate = RECORD
16 1216 --                              bounds: RECT;
16 1217 --                              procID: INTEGER;
16 1218 --                              visible, filler1: BOOLEAN;
16 1219 --                              goAway, filler2: BOOLEAN;
16 1220 --                              refcon: LONGINT;
16 1221 --                              itemsID: INTEGER; { only for DLOG resource }
16 1222 --                            END;
16 1223 --          PTemplate = ^WindowTemplate;
16 1224 --          HTemplate = ^PTemplate;
16 1225 --
16 1226 --      VAR
16 1227 --          aWmgrWindow:    WindowPtr;
16 1228 --          templateHandle: HTemplate;
16 1229 --          rsrcType:       ResType;
16 1230 --          ditl:           Handle;
16 1231 --          oldPerm:        BOOLEAN;
16 1232 --          fi:             FailInfo;
16 1233 --
16 1234 --          {-------------------------------+
16 1235 --          |   HdlFailure                  |
16 1236 --          +-------------------------------}
16 1237 -- B       PROCEDURE HdlFailure(error: INTEGER; message: LONGINT); { ??? ERROR ??? }
16 1238 0- B       BEGIN
16 1239 --              { Make sure the perm allocation flag is set back to what it was
16 1240 --                  when we entered GetRsrcWindow. }
16 1241 --              oldPerm := PermAllocation(oldPerm);
16 1242 -0 B       END;
16 1243 --
16 1244 --
16 1245 0- A    BEGIN
16 1246 --          oldPerm := PermAllocation(FALSE);
16 1247 --                  { Even though the window is permanent, we allocate it
16 1248 --                      under a temporary flag so that the maximum memory
16 1249 --                      is available.  Quickdraw can blow up if it can't
16 1250 --                      allocate a grafPort. }
16 1251 --
16 1252 --          CatchFailures(fi, HdlFailure);
16 1253 --
16 1254 --          templateHandle := HTemplate(GetResource('WIND', rsrcId));
16 1255 --          FailNIL(templateHandle);
16 1256 --          MoveHHi(Handle(templateHandle)); { in case it is locked by the ROM }
16 1257 --
16 1258 --          WITH templateHandle^^ DO
16 1259 1-              BEGIN
16 1260 --      {$IFC NOT qNeedsROM128K}
16 1261 --                  { ignore request for zoomDocProc if not 128K ROM, because
16 1262 --                      the user might be running pre-3.0 System, which can't
16 1263 --                      handle zoomDocProc }
16 1264 --              IF NOT gConfiguration.hasROM128K THEN
16 1265 --                  procID := BAND(procID, $FFF7);
16 1266 --      {$ENDC}
16 1267 --
16 1268 --              visible := FALSE;
16 1269 --              isClosable := goAway;
16 1270 --              isResizable := (procID = documentProc) OR (procID = zoomDocProc);
16 1271 --                  { If your own defProc is resizable, too, then after
16 1272 --                          the call on GetRsrcWindow, set isResizable TRUE }
16 1273 -1              END;
16 1274 --
16 1275 --          IF gConfiguration.hasColorQD THEN
16 1276 --              aWmgrWindow := WindowPtr(GetNewCWindow(rsrcId, Pointer(storage), Pointer(-1)))
16 1277 --          ELSE
16 1278 --              aWmgrWindow := GetNewWindow(rsrcId, Pointer(storage), Pointer(-1));
16 1279 --
16 1280 --          SetWRefCon(aWmgrWindow, templateHandle^^.procID);   { Added for purposes of window template writing }
16 1281 --
16 1282 --          oldPerm := PermAllocation(oldPerm);
16 1283 --          Success(fi);    { Don't need the failure handler since we've set the
16 1284 --                              perm allocation flag back. }
16 1285 --
16 1286 --          FailNIL(aWmgrWindow);
```

```
16 1287 --
16 1288 --          { Now we must make sure that the code reserve is still intact.}
16 1289 --          IF NOT CheckReserve THEN
16 1290 1-              BEGIN
16 1291 --              FreeWmgrWindow(aWmgrWindow, storage = NIL);
16 1292 --
16 1293 --              Failure(memFullErr, 0);
16 1294 -1             END;
16 1295 --
16 1296 --          GetRsrcWindow := aWmgrWindow;
16 1297 -0 A    END;
16 1298 --
16 1299 --      {$S MARes}
16 1300 --      {-----------------------------------+
16 1301 --      |   HandleActivateEvent             |
16 1302 --      +-----------------------------------}
16 1303 -- A    FUNCTION TApplication.HandleActivateEvent (VAR theEventInfo: EventInfo): TCommand;
16 1304 --
16 1305 --      VAR
16 1306 --          aWindow:        TWindow;
16 1307 --
16 1308 0- A    BEGIN
16 1309 --          WITH theEventInfo, thePEvent^ DO
16 1310 --      {$IFC qWWNeeded}
16 1311 --              IF WindowPtr(message) = gDebugWindowPtr THEN
16 1312 --                  WWActivateEvent(modifiers)
16 1313 --              ELSE
16 1314 --      {$ENDC}
16 1315 1-                  BEGIN
16 1316 --                  aWindow := WMgrToWindow(WindowPtr(message));
16 1317 --                  IF aWindow <> NIL THEN
16 1318 --                      aWindow.Activate(Odd(modifiers));
16 1319 -1                 END;
16 1320 --
16 1321 --          HandleActivateEvent := gNoChanges;
16 1322 -0 A    END { TApplication.HandleActivateEvent };
16 1323 --
16 1324 --      {$S MARes}
16 1325 --      {-----------------------------------+
16 1326 --      |   HandleAlienEvent                |
16 1327 --      +-----------------------------------}
16 1328 -- A    FUNCTION  TApplication.HandleAlienEvent(VAR theEventInfo: EventInfo): TCommand;
16 1329 --
16 1330 --      VAR
16 1331 --          aCommand:       TCommand;
16 1332 --          anEvtHandler:   TEvtHandler;
16 1333 --
16 1334 --          {-------------------------------+
16 1335 --          |   TakeEvent                   |
16 1336 --          +-------------------------------}
16 1337 -- B        FUNCTION  TakeEvent(anEvtHandler: TEvtHandler): BOOLEAN;
16 1338 0- B        BEGIN
16 1339 --              TakeEvent := anEvtHandler.DoHandleEvent(theEventInfo.thePEvent, aCommand);
16 1340 -0 B        END;
16 1341 --
16 1342 0- A    BEGIN
16 1343 --          aCommand := gNoChanges;
16 1344 --          anEvtHandler := FirstHandlerThat(gHeadCohandler, TakeEvent);
16 1345 --          HandleAlienEvent := aCommand;
16 1346 -0 A    END;
16 1347 --
16 1348 --      {$S MADoCommand}
16 1349 --      {-----------------------------------+
16 1350 --      |   HandleDiskEvent                 |
16 1351 --      +-----------------------------------}
16 1352 -- A    FUNCTION  TApplication.HandleDiskEvent (VAR theEventInfo: EventInfo): TCommand;
16 1353 --
16 1354 --      CONST
16 1355 --          topLeft =   $00500070;
16 1356 --
16 1357 --      VAR
16 1358 --          err:        INTEGER;
16 1359 --
16 1360 0- A    BEGIN
16 1361 --          WITH theEventInfo.thePEvent^ DO
16 1362 --              IF HiWrd(message) <> noErr THEN
16 1363 1-                  BEGIN
16 1364 --                  err := DIBadMount(Point(topLeft), message);  { ??? do something with the error ??? }
16 1365 --          {$IFC qDebug}
16 1366 --                  IF err <> noErr THEN
16 1367 --                      Writeln('error from DIBadMount is ', err:1);
16 1368 --          {$ENDC}
16 1369 -1                 END;
16 1370 --
16 1371 --          HandleDiskEvent := gNoChanges;
16 1372 --
16 1373 -0 A    END;
16 1374 --
16 1375 --      {$S MARes}
16 1376 --      {-----------------------------------+
16 1377 --      |   HandleEvent                     |
16 1378 --      +-----------------------------------}
16 1379 -- A    PROCEDURE TApplication.HandleEvent (VAR theEvent: EventRecord);
16 1380 --
16 1381 --      VAR
16 1382 --          fi:                 FailInfo;
16 1383 --          commandToPerform:   TCommand;
16 1384 --          theEventInfo:       EventInfo;
16 1385 --
```

```
16 1386 --          {------------------------------+
16 1387 --          |   HandleFailure              |
16 1388 --          +------------------------------}
16 1389 -- B        PROCEDURE HandleFailure(error: OSErr; message: LONGINT);
16 1390 0- B        BEGIN
16 1391 --              PostHandleEvent(theEventInfo);
16 1392 -0 B        END;
16 1393 --
16 1394 0- A    BEGIN
16 1395 --          WITH theEventInfo, theEvent DO
16 1396 1-              BEGIN
16 1397 --              thePEvent := @theEvent;
16 1398 --              theBtnState := BTst(modifiers, 7 { btnState });
16 1399 --              theCmdKey := BTst(modifiers, 8 { cmdKey });
16 1400 --              theShiftKey := BTst(modifiers, 9 { shiftKey });
16 1401 --              theAlphaLock := BTst(modifiers, 10 { alphaLock });
16 1402 --              theOptionKey := BTst(modifiers, 11 { optionKey });
16 1403 --              theControlKey := BTst(modifiers, 12);
16 1404 --              theAutoKey := what = autoKey;
16 1405 --              theClickCount := gClickCount;
16 1406 -1              END;
16 1407 --
16 1408 --      {$IFC qDebug}
16 1409 --          IF gReportEvt AND ((theEventInfo.thePEvent^.what = activateEvt) OR
16 1410 --                             (FrontWindow <> gDebugWindowPtr)) THEN
16 1411 --              ReportEvent(theEventInfo);
16 1412 --      {$ENDC}
16 1413 --
16 1414 --          CatchFailures(fi, HandleFailure);
16 1415 --
16 1416 --          DispatchEvent(theEventInfo, commandToPerform);
16 1417 --          IF commandToPerform <> gNoChanges THEN
16 1418 --              PerformCommand(commandToPerform);
16 1419 --
16 1420 --          Success(fi);
16 1421 --
16 1422 --          PostHandleEvent(theEventInfo);
16 1423 --
16 1424 -0 A    END { TApplication.HandleEvent };
16 1425 --
16 1426 --      {$S MAFinder}
16 1427 --      {-----------------------------------+
16 1428 --      |   HandleFinderRequest             |
16 1429 --      +-----------------------------------}
16 1430 -- A    PROCEDURE TApplication.HandleFinderRequest;
16 1431 --      LABEL 1, 2;
16 1432 --
16 1433 --      VAR
16 1434 --          i:                  INTEGER;
16 1435 --          anAppFile:          AppFile;
16 1436 --          doc:                TDocument;
16 1437 --          continuePrinting:   BOOLEAN;
16 1438 --          cmd:                CmdNumber;
16 1439 --          fi:                 FailInfo;
16 1440 --
16 1441 --                  { ??? need better error messages here ??? }
16 1442 --              {------------------------------+
16 1443 --              |   HdlORequest                |
16 1444 --              +------------------------------}
16 1445 -- B            PROCEDURE HdlORequest(error: OSErr; message: LONGINT);
16 1446 0- B            BEGIN
16 1447 --                  IF error <> noErr THEN
16 1448 1-                      BEGIN
16 1449 --                      IF message = 0 THEN
16 1450 2-                          BEGIN
16 1451 --                          gErrorParm3 := anAppFile.fName;
16 1452 --                          IF cmd = cFinderPrint THEN
16 1453 --                              message := msgPrintFailed
16 1454 --                          ELSE
16 1455 --                              message := msgOpenFailed;
16 1456 -2                          END;
16 1457 --                      ShowError(error, message);
16 1458 -1                      END;
16 1459 --                  GOTO 1; { continue the loop }
16 1460 -0 B            END;
16 1461 --
16 1462 --              {------------------------------+
16 1463 --              |   HdlNRequest                |
16 1464 --              +------------------------------}
16 1465 -- B            PROCEDURE HdlNRequest(error: OSErr; message: LONGINT);
16 1466 0- B            BEGIN
16 1467 --                  IF error <> noErr THEN
16 1468 --                      ShowError(error, message);
16 1469 --                  GOTO 2; { exit the method }
16 1470 -0 B            END;
16 1471 0- A    BEGIN
16 1472 --      {$IFC qDebug}
16 1473 --          IF gExperimenting THEN
16 1474 --              WriteLn('File count: ', gFileCount:1);
16 1475 --      {$ENDC}
16 1476 --
16 1477 --          continuePrinting := TRUE;
16 1478 --
16 1479 --          IF gFileCount = 0 THEN
16 1480 1-              BEGIN
16 1481 --              CatchFailures(fi, HdlNRequest);
16 1482 --              OpenNew(cFinderNew);
16 1483 --              Success(fi);
16 1484 -1              END
```

```
16 1485 --          ELSE { it's an OPEN or PRINT of 1 or more existing files }
16 1486 1-              BEGIN
16 1487 --              IF gFinderPrinting THEN
16 1488 --                  cmd := cFinderPrint
16 1489 --              ELSE
16 1490 --                  cmd := cFinderOpen;
16 1491 --
16 1492 --              FOR i := 1 TO gFileCount DO
16 1493 2-                  BEGIN
16 1494 --                  CatchFailures(fi, HdlORequest);
16 1495 --
16 1496 --                  GetAppFiles(i, anAppFile);
16 1497 --
16 1498 --                  IF CanOpenDocument(cmd, anAppFile) THEN
16 1499 3-                      BEGIN
16 1500 --                      ClrAppFiles(i);
16 1501 --
16 1502 --                      IF gFinderPrinting THEN
16 1503 4-                          BEGIN
16 1504 --                          IF continuePrinting THEN
16 1505 --                              continuePrinting := PrintDocument(anAppFile);
16 1506 -4                          END
16 1507 --                      ELSE
16 1508 --                          OpenOld(cFinderOpen, anAppFile);
16 1509 -3                      END
16 1510 --                  ELSE
16 1511 --                      Failure(errNotMyType, 0);
16 1512 --
16 1513 --                  Success(fi);
16 1514 --      1:              { continue the loop }
16 1515 -2                  END;
16 1516 -1              END;
16 1517 --      2:  { exit the method }
16 1518 -0 A   END;
16 1519 --
16 1520 --      {$S MARes}
16 1521 --      {-----------------------------------+
16 1522 --      |   HandleKeyDownEvent              |
16 1523 --      +-----------------------------------}
16 1524 -- A   FUNCTION  TApplication.HandleKeyDownEvent (VAR theEventInfo: EventInfo): TCommand;
16 1525 --
16 1526 --      VAR
16 1527 --          ch:         CHAR;
16 1528 --          keycode:    INTEGER;
16 1529 --
16 1530 0- A   BEGIN
16 1531 --          WITH theEventInfo, thePEvent^ DO
16 1532 1-              BEGIN
16 1533 --              ch := CHR(BAND(message, charCodeMask));
16 1534 --              keyCode := BSR(BAND(message, keyCodeMask), 8);
16 1535 --
16 1536 --              IF theCmdKey THEN
16 1537 --                  HandleKeyDownEvent := gTarget.DoCommandKey(ch, theEventInfo)
16 1538 --              ELSE
16 1539 --                  HandleKeyDownEvent := gTarget.DoKeyCommand(ch, keycode, theEventInfo);
16 1540 -1              END;
16 1541 -0 A   END { TApplication.HandleKeyDown };
16 1542 --
16 1543 --      {$S MARes}
16 1544 --      {-----------------------------------+
16 1545 --      |   HandleMouseDown                 |
16 1546 --      +-----------------------------------}
16 1547 -- A   FUNCTION  TApplication.HandleMouseDown(VAR theEventInfo: EventInfo): TCommand;
16 1548 --
16 1549 --      VAR
16 1550 --          doClick:            BOOLEAN;
16 1551 --          aWindow:            TWindow;
16 1552 --          aWMgrWindow:        WindowPtr;
16 1553 --          whereMouseDown:     INTEGER;
16 1554 --          sysWindowAct:       BOOLEAN;
16 1555 --          aCommand:           TCommand;
16 1556 --          theMouse:           Point;
16 1557 --          theVMouse:          VPoint;
16 1558 --          hysteresis:         Point;
16 1559 --
16 1560 0- A   BEGIN
16 1561 --          HandleMouseDown := gNoChanges;
16 1562 --
16 1563 --          WITH theEventInfo, thePEvent^ DO
16 1564 1-              BEGIN
16 1565 --              whereMouseDown := FindWindow(where, aWmgrWindow);
16 1566 --              theClickCount := CountClicks(thePEvent, whereMouseDown);
16 1567 --
16 1568 --      {$IFC qDebug}
16 1569 --              IF gReportEvt AND (FrontWindow <> gDebugWindowPtr) AND (aWmgrWindow <> gDebugWindowPtr) THEN
16 1570 --                  ReportMouseDown(whereMouseDown);
16 1571 --      {$ENDC}
16 1572 --
16 1573 --              aWindow := WMgrToWindow(aWMgrWindow);
16 1574 --
16 1575 --              IF ((whereMouseDown = inMenuBar) & InModalMenuState) |
16 1576 --                 ((whereMouseDown <> inMenuBar) & InModalState & (aWindow <> gFrontWindow)) THEN
16 1577 2-                  BEGIN
16 1578 --                  SysBeep(2);
16 1579 --                  EXIT(HandleMouseDown);
16 1580 -2                  END;
16 1581 --
16 1582 --      {$IFC qWWNeeded}
16 1583 --              IF aWmgrWindow = gDebugWindowPtr THEN
```

```
16 1584 2-                BEGIN
16 1585 --                IF gDebugWindowPtr <> FrontWindow THEN
16 1586 --                    SelectWMgrWindow(gDebugWindowPtr);
16 1587 --                WWMouseDown(whereMouseDown, where, modifiers);
16 1588 --                EXIT(HandleMouseDown);
16 1589 -2                END;
16 1590 --      {$ENDC}
16 1591 -1            END;
16 1592 --
16 1593 --          IF whereMouseDown <> inContent THEN
16 1594 --              SetCursor(arrow);
16 1595 --
16 1596 --          WITH theEventInfo, thePEvent^ DO
16 1597 1-              CASE whereMouseDown OF
16 1598 --                  inMenuBar:
16 1599 2-                      BEGIN
16 1600 --                      SetupTheMenus; { gives application a chance to setup individual menu items }
16 1601 --                      HandleMouseDown := MenuEvent(MenuSelect(where));
16 1602 -2                      END;
16 1603 --
16 1604 --                  inSysWindow:
16 1605 --                      SystemClick(thePEvent^, aWmgrWindow);
16 1606 --
16 1607 --                  inContent:
16 1608 --                      IF aWindow.Focus THEN       { if we can't focus, we're in trouble }
16 1609 2-                          BEGIN
16 1610 --                          theMouse := where;
16 1611 --                          GlobalToLocal(theMouse);
16 1612 --                          aWindow.QDToViewPt(theMouse, theVMouse);
16 1613 --                          hysteresis := gStdHysteresis;
16 1614 --                          IF aWindow.HandleMouseDown(theVMouse, theEventInfo,
16 1615 --                                                     hysteresis, aCommand) THEN
16 1616 --                              IF aCommand <> gNoChanges THEN
16 1617 --                                  HandleMouseDown := TrackMouse(where, hysteresis, aCommand);
16 1618 -2                          END;
16 1619 --
16 1620 --                  inDrag:
16 1621 --                      aWindow.MoveByUser(where);
16 1622 --
16 1623 --                  inGrow:
16 1624 --                      aWindow.ResizeByUser(where);
16 1625 --
16 1626 --                  inGoAway:
16 1627 --                      IF TrackGoAway(aWmgrWindow, where) THEN
16 1628 --                          CloseWmgrWindow(aWmgrWindow); { ??? }
16 1629 --
16 1630 --                  inZoomIn, inZoomOut:
16 1631 --                      IF TrackBox(aWmgrWindow, where, whereMouseDown) THEN
16 1632 --                              aWindow.Zoom(whereMouseDown);
16 1633 --
16 1634 --                  inDesk:
16 1635 --                      ;
16 1636 --
16 1637 -1                  END;  { CASE }
16 1638 -0 A   END;
16 1639 --
16 1640 --      {$S MARes}
16 1641 --      {-----------------------------------+
16 1642 --      |   HandleMouseUp                   |
16 1643 --      +-----------------------------------}
16 1644 -- A   FUNCTION  TApplication.HandleMouseUp (VAR theEventInfo: EventInfo): TCommand;
16 1645 --
16 1646 0- A   BEGIN
16 1647 --          { Remember time of last mouse up, in order to detect double clicks }
16 1648 --          gLastUpTime := theEventInfo.thePEvent^.when;
16 1649 --          HandleMouseUp := gNoChanges;
16 1650 -0 A   END { TApplication.HandleMouseUp };
16 1651 --
16 1652 --      {$S MARes}
16 1653 --      {-----------------------------------+
16 1654 --      |   HandleSystemEvent               |
16 1655 --      +-----------------------------------}
16 1656 -- A   FUNCTION  TApplication.HandleSystemEvent(VAR theEventInfo: EventInfo): TCommand;
16 1657 --
16 1658 --      VAR
16 1659 --          e:                  OSErr;
16 1660 --          switchingIn:        BOOLEAN;
16 1661 --          convertClipboard:   BOOLEAN;
16 1662 --          aWMgrWindow:        WindowPtr;
16 1663 --          aWindow:            TWindow;
16 1664 --          anEvent:            EventRecord;
16 1665 --          dontCare:           BOOLEAN;
16 1666 --
16 1667 0- A   BEGIN
16 1668 --          WITH theEventInfo.thePEvent^ DO
16 1669 1-              CASE BSR(BAND(message, $FF000000), 24) OF
16 1670 --                  kSuspendOrResume:
16 1671 2-                      BEGIN
16 1672 --                      switchingIn := Odd(message);
16 1673 --                      convertClipboard := BAND(message, $00000002) <> 0;
16 1674 --
16 1675 --                      IF switchingIn THEN
16 1676 --                          RegainControl(convertClipboard)
16 1677 --                      ELSE
16 1678 --                          AboutToLoseControl(convertClipboard);
16 1679 --
16 1680 --                      aWMgrWindow := FrontWindow;
16 1681 --      {$IFC qWWNeeded}
16 1682 --                      IF aWMgrWindow = gDebugWindowPtr THEN
```

```
16 1683 --                          WWActivateEvent(ORD(switchingIn))
16 1684 --                      ELSE
16 1685 --      {$ENDC}
16 1686 --                      IF aWMgrWindow <> NIL THEN
16 1687 3-                          BEGIN
16 1688 --                          aWindow := WMgrToWindow(aWMgrWindow);
16 1689 --                          IF aWindow <> NIL THEN
16 1690 --                              aWindow.Activate(switchingIn);
16 1691 -3                         END;
16 1692 --                      gInBackground := NOT switchingIn;
16 1693 -2                      END;
16 1694 -1                  END;
16 1695 --
16 1696 --          HandleSystemEvent := gNoChanges;
16 1697 -0 A    END;
16 1698 --
16 1699 --      {$S MARes}
16 1700 --      {-----------------------------------+
16 1701 --      |   HandleUpdateEvent               |
16 1702 --      +-----------------------------------}
16 1703 -- A    FUNCTION  TApplication.HandleUpdateEvent (VAR theEventInfo: EventInfo): TCommand;
16 1704 --
16 1705 --      VAR
16 1706 --          aWindow:        TWindow;
16 1707 --
16 1708 0- A    BEGIN
16 1709 --          WITH theEventInfo.thePEvent^ DO
16 1710 1-              BEGIN
16 1711 --      {$IFC qWWNeeded}
16 1712 --              IF WindowPtr(message) = gDebugWindowPtr THEN
16 1713 --                  WWUpdateEvent
16 1714 --              ELSE
16 1715 --      {$ENDC}
16 1716 2-                  BEGIN
16 1717 --                  aWindow := WMgrToWindow(WindowPtr(message));
16 1718 --                  IF aWindow <> NIL THEN
16 1719 --                      aWindow.Update;
16 1720 -2                  END;
16 1721 -1              END;
16 1722 --          HandleUpdateEvent := gNoChanges;
16 1723 -0 A    END { TApplication.HandleUpdateEvent };
16 1724 --
16 1725 --      {$IFC qDebug}
16 1726 --      {$S MADebug}
16 1727 --      {$IFC qTrace}{$D+}{$ENDC}
16 1728 --      {-----------------------------------+
16 1729 --      |   IdentifySoftware                |
16 1730 --      +-----------------------------------}
16 1731 -- A    PROCEDURE TApplication.IdentifySoftware;
16 1732 0- A    BEGIN
16 1733 --          Writeln('UMacApp of ', COMPDATE, ' @ ', COMPTIME);
16 1734 --          IDUObject;
16 1735 --          IDUWritelnWindow;
16 1736 --          IDUTrace;
16 1737 -0 A    END;
16 1738 --      {$IFC qTrace}{$D++}{$ENDC}
16 1739 --      {$ENDC}
16 1740 --
16 1741 --      {$S MARes}
16 1742 --      {-----------------------------------+
16 1743 --      |   Idle                            |
16 1744 --      +-----------------------------------}
16 1745 -- A    PROCEDURE TApplication.Idle(phase: IdlePhase);
16 1746 --
16 1747 --      VAR
16 1748 --          currTick:       LONGINT;
16 1749 --          dontCare:       BOOLEAN;
16 1750 --
16 1751 --          {-------------------------------+
16 1752 --          |   DoIdleAction                |
16 1753 --          +-------------------------------}
16 1754 -- B        PROCEDURE DoIdleAction(anEvtHandler: TEvtHandler);
16 1755 --
16 1756 0- B        BEGIN
16 1757 --              { If this handler needs idling, and enough ticks have elapsed
16 1758 --                since the last time it was idled, call its DoIdle.  (This was not
16 1759 --                made a TEvtHandler method in order to optimize idling speed.) }
16 1760 --              WITH anEvtHandler DO
16 1761 --                  IF fIdleFreq <> kMaxIdleTime THEN   { Does it idle at all? }
16 1762 1-                      BEGIN
16 1763 --                      IF (phase = idleBegin) | (currTick >= fLastIdle + fIdleFreq) THEN
16 1764 2-                          BEGIN
16 1765 --                          anEvtHandler.DoIdle(phase);
16 1766 --                          fLastIdle := currTick;
16 1767 -2                          END;
16 1768 --                      gIdleFreq := Min(fIdleFreq, gIdleFreq);
16 1769 -1                      END;
16 1770 -0 B        END;
16 1771 --
16 1772 0- A    BEGIN
16 1773 --
16 1774 --          currTick := TickCount;
16 1775 --
16 1776 --          IF phase = idleBegin THEN
16 1777 1-              BEGIN
16 1778 --      {$IFC qTrace}
16 1779 --              TRCEnable(gTraceIdle);      { Trace during idle only if user wants to. }
16 1780 --      {$ENDC}
16 1781 --              IF MemSpaceIsLow THEN
```

```
16 1782 --                  SpaceIsLow
16 1783 --              ELSE
16 1784 --                  gNextSpaceMsg := currTick;
16 1785 --
16 1786 --              SetupTheMenus;              { To get the menu bar redrawn if necessary. }
16 1787 --              gLastIdle := 0;             { Force idling event handlers & co-handlers. }
16 1788 --              gIdleFreq := 0;
16 1789 -1             END;
16 1790 --
16 1791 --          IF currTick >= gLastIdle + gIdleFreq THEN
16 1792 1-              BEGIN
16 1793 --              gIdleFreq := kMaxIdleTime;
16 1794 --              EachHandler(gHeadCohandler, DoIdleAction);
16 1795 --              EachHandler(gTarget, DoIdleAction);
16 1796 --              gLastIdle := currTick;
16 1797 -1              END;
16 1798 --
16 1799 --          IF phase <> idleEnd THEN
16 1800 --              dontCare := TrackCursor
16 1801 --      {$IFC qTrace}
16 1802 --          ELSE
16 1803 --              TRCEnable(TRUE)     { Make sure tracing back on at end of idle. }
16 1804 --      {$ENDC}
16 1805 --          ;
16 1806 --
16 1807 -0 A    END;
16 1808 --
16 1809 --      {-----------------------------------+
16 1810 --      |   InModalState                    |
16 1811 --      +-----------------------------------}
16 1812 --      {$S MARes}
16 1813 -- A    FUNCTION  TApplication.InModalState: BOOLEAN;
16 1814 --
16 1815 0- A    BEGIN
16 1816 --          InModalState := (gFrontWindow <> NIL) & (gFrontWindow.fIsModal);
16 1817 -0 A    END { TApplication.InModalState };
16 1818 --
16 1819 --      {-----------------------------------+
16 1820 --      |   InModalMenuState                |
16 1821 --      +-----------------------------------}
16 1822 --      {$S MARes}
16 1823 -- A    FUNCTION  TApplication.InModalMenuState: BOOLEAN;
16 1824 --
16 1825 0- A    BEGIN
16 1826 --          InModalMenuState := (gFrontWindow <> NIL) & (NOT gFrontWindow.AllowsMenuAccess);
16 1827 -0 A    END { TApplication.InModalMenuState };
16 1828 --
16 1829 --
16 1830 --      {$S MANonRes}
16 1831 --      {-----------------------------------+
16 1832 --      |   InstallCohandler                |
16 1833 --      +-----------------------------------}
16 1834 -- A    PROCEDURE TApplication.InstallCohandler(aCohandler: TEvtHandler; addIt: BOOLEAN);
16 1835 --      VAR
16 1836 --          prevCohandler:    TEvtHandler;
16 1837 --          currCohandler:    TEvtHandler;
16 1838 0- A    BEGIN
16 1839 --          IF addIt THEN
16 1840 1-              BEGIN
16 1841 --              aCohandler.fNextHandler := gHeadCohandler;
16 1842 --              gHeadCohandler := aCohandler;
16 1843 -1              END
16 1844 --          ELSE
16 1845 1-              BEGIN
16 1846 --              prevCohandler := NIL;
16 1847 --              currCohandler := gHeadCohandler;
16 1848 --              WHILE currCohandler <> NIL DO
16 1849 2-                  BEGIN
16 1850 --                  IF currCohandler = aCohandler THEN { found it }
16 1851 3-                      BEGIN
16 1852 --                      IF prevCohandler = NIL THEN
16 1853 --                          gHeadCohandler := aCohandler.fNextHandler
16 1854 --                      ELSE
16 1855 --                          prevCohandler.fNextHandler := aCohandler.fNextHandler;
16 1856 --                      currCohandler := NIL;  { So loop will end }
16 1857 -3                      END
16 1858 --                  ELSE
16 1859 3-                      BEGIN
16 1860 --                      prevCohandler := currCohandler;
16 1861 --                      currCohandler := currCohandler.fNextHandler;
16 1862 -3                      END;
16 1863 -2                  END;
16 1864 -1              END;
16 1865 -0 A    END;
16 1866 --
16 1867 --      {$S MARes}
16 1868 --      {-----------------------------------+
16 1869 --      |   IsDeskAccessory                 |
16 1870 --      +-----------------------------------}
16 1871 -- A    FUNCTION  TApplication.IsDeskAccessory (aWMgrWindow: WindowPtr): BOOLEAN;
16 1872 0- A    BEGIN
16 1873 --          IF aWMgrWindow <> NIL THEN
16 1874 --              IsDeskAccessory := WindowPeek(aWMgrWindow)^.windowKind < 0
16 1875 --          ELSE
16 1876 --              IsDeskAccessory := FALSE;
16 1877 -0 A    END;
16 1878 --
16 1879 --      {$S MAOpen}
16 1880 --      {-----------------------------------+
```

```
16 1881 --      |   KindOfDocument                  |
16 1882 --      +-----------------------------------}
16 1883 -- A    FUNCTION  TApplication.KindOfDocument(itsCmdNumber: CmdNumber; itsPAppFile: PAppFile): CmdNumber;
16 1884 0- A    BEGIN
16 1885 --          KindOfDocument := itsCmdNumber;
16 1886 -0 A    END;
16 1887 --
16 1888 --      {$S MAInit}
16 1889 --      {-----------------------------------+
16 1890 --      |   LaunchClipboard                 |
16 1891 --      +-----------------------------------}
16 1892 -- A    PROCEDURE TApplication.LaunchClipboard;
16 1893 0- A    BEGIN
16 1894 --          AbsorbScrapStuff;   { Get current scrapCount as a baseline }
16 1895 --          ReadFromDeskScrap;
16 1896 -0 A    END;
16 1897 --
16 1898 --      {$S Main} { must be in the main segment }
16 1899 --      {-----------------------------------+
16 1900 --      |   MainEventLoop                   |
16 1901 --      +-----------------------------------}
16 1902 -- A    PROCEDURE TApplication.MainEventLoop;
16 1903 0- A    BEGIN
16 1904 --          gIdlePhase := idleBegin;
16 1905 1-          REPEAT
16 1906 --              IF gIdlePhase = idleBegin THEN
16 1907 --                  UnloadAllSegments; { don't unload segments after idle has begun }
16 1908 --
16 1909 --                  { ??? should we (1) unload segs after completing idle but before doing the event?
16 1910 --                                  (2) unload segs while processing event during background printing? }
16 1911 --
16 1912 --              PollEvent;
16 1913 -1          UNTIL
16 1914 --              gAppDone; { gAppDone is a global BOOLEAN; that we set TRUE when the user chooses 'Quit' }
16 1915 -0 A    END;
16 1916 --
16 1917 --      {$S MAInit}
16 1918 --      {-----------------------------------+
16 1919 --      |   MakeClipboardWindow             |
16 1920 --      +-----------------------------------}
16 1921 -- A    FUNCTION  TApplication.MakeClipboardWindow: TWindow;
16 1922 --
16 1923 0- A    BEGIN
16 1924 --      {$IFC qTemplateViews}
16 1925 --          MakeClipboardWindow := NewTemplateWindow(kIDClipWindow, NIL);
16 1926 --      {$ELSEC}
16 1927 --          MakeClipboardWindow := NewSimpleWindow(kIDClipWindow, TRUE, TRUE, NIL, NIL);
16 1928 --      {$ENDC}
16 1929 -0 A    END;
16 1930 --
16 1931 --      {$S MAClipboard}
16 1932 --      {-----------------------------------+
16 1933 --      |   MakeViewForAlienClipboard       |
16 1934 --      +-----------------------------------}
16 1935 -- A    FUNCTION  TApplication.MakeViewForAlienClipboard: TView;
16 1936 --
16 1937 0- A    BEGIN
16 1938 --          { If the application doesn't override this then we just set the
16 1939 --              clipboard view to the orphanage, which handles TEXT and PICT
16 1940 --              scraps in a standard way. }
16 1941 --          MakeViewForAlienClipboard := gClipOrphanage;
16 1942 -0 A    END;
16 1943 --
16 1944 --      {$S MASelCommand}
16 1945 --      {-----------------------------------+
16 1946 --      |   MenuEvent                       |
16 1947 --      +-----------------------------------}
16 1948 -- A    FUNCTION  TApplication.MenuEvent(menuItem: LONGINT): TCommand;
16 1949 --
16 1950 --      VAR
16 1951 --          fi:                 FailInfo;
16 1952 --          alreadyDidCommand:  BOOLEAN;
16 1953 --          newCommand:         TCommand;
16 1954 --          cmd:                CmdNumber;
16 1955 --          s:                  Str255;
16 1956 --          theMenuNumber:      INTEGER;
16 1957 --          theItemNumber:      INTEGER;
16 1958 --
16 1959 --          {-------------------------------+
16 1960 --          |   HdlMenuEvt                  |
16 1961 --          +-------------------------------}
16 1962 -- B        PROCEDURE HdlMenuEvt(error: OSErr; message: LONGINT);
16 1963 0- B        BEGIN
16 1964 --              IF gSysWindowActive THEN
16 1965 --                  BusyActivate(FALSE);
16 1966 --
16 1967 --              FailNewMessage(error, message, BuildMessage(cmd, msgCmdErr));
16 1968 -0 B        END;
16 1969 --
16 1970 0- A    BEGIN
16 1971 --          MenuEvent := gNoChanges;
16 1972 --          theMenuNumber := HiWrd(menuItem);
16 1973 --          theItemNumber := LoWrd(menuItem);
16 1974 --
16 1975 --          alreadyDidCommand := FALSE;
16 1976 --          IF theMenuNumber <> 0 THEN
16 1977 1-              BEGIN
16 1978 --
16 1979 --              cmd := CmdFromMenuItem(theMenuNumber, theItemNumber);
```

```
16 1980 --      {$IFC qDebug}
16 1981 --              IF cmd = cCantUndo THEN
16 1982 2-                  BEGIN
16 1983 --                  Writeln('Command number ', cCantUndo:1,
16 1984 --                                              ' is reserved for MacApp.');
16 1985 --                  ProgramBreak('Use of reserved command number.');
16 1986 -2                  END;
16 1987 --
16 1988 --              IF gReportMenuChoices AND (cmd > 0) THEN
16 1989 --                  WriteLn('Menu Choice Command Number = ', cmd:1);
16 1990 --      {$ENDC qDebug}
16 1991 --
16 1992 --              IF (cmd >= cEditBase) AND (cmd <= cEditLast) THEN
16 1993 --
16 1994 --                  alreadyDidCommand := SystemEdit(cmd - cEditBase)
16 1995 --
16 1996 --              ELSE IF (cmd < 0) AND (theMenuNumber = mApple) THEN
16 1997 2-                  BEGIN
16 1998 --                  GetItem(GetMHandle(mApple), theItemNumber, s);
16 1999 --                  OpenDeskAccessory(s);
16 2000 --                  alreadyDidCommand := TRUE;
16 2001 -2                  END;
16 2002 --
16 2003 --              IF NOT alreadyDidCommand THEN
16 2004 2-                  BEGIN
16 2005 --                  CatchFailures(fi, HdlMenuEvt);
16 2006 --
16 2007 --                  IF gSysWindowActive THEN
16 2008 --                      BusyActivate(TRUE);
16 2009 --
16 2010 --                  MenuEvent := gTarget.DoMenuCommand(cmd);
16 2011 --
16 2012 --                  IF gSysWindowActive THEN
16 2013 --                      BusyActivate(FALSE);
16 2014 --
16 2015 --                  Success(fi);
16 2016 -2                  END;
16 2017 -1              END;
16 2018 -0 A    END;
16 2019 --
16 2020 --      {$S MASelCommand}
16 2021 --      {---------------------------------+
16 2022 --      |   OpenDeskAccessory             |
16 2023 --      +---------------------------------}
16 2024 -- A    PROCEDURE TApplication.OpenDeskAccessory (deskAccName: Str255);
16 2025 --
16 2026 --      VAR
16 2027 --          aRefNum:            INTEGER;
16 2028 --          drvrH:              Handle;
16 2029 --          theID:              INTEGER;
16 2030 --          theType:            ResType;
16 2031 --          theName:            Str255;
16 2032 --          oldPerm:            BOOLEAN;
16 2033 --          ourHeap:            BOOLEAN;
16 2034 --          fi:                 FailInfo;
16 2035 --          err:                OSErr;
16 2036 --
16 2037 -- B        PROCEDURE HdlOpenDeskAcc(error: OSErr; message: LONGINT);
16 2038 0- B        BEGIN
16 2039 --              IF aRefNum <> 0 THEN
16 2040 --                  CloseDeskAcc(aRefNum);
16 2041 --
16 2042 --              IF message = 0 THEN
16 2043 1-                  BEGIN
16 2044 --                  gErrorParm3 := deskAccName;
16 2045 --                  { Get rid of leading null character }
16 2046 --                  IF Ord(gErrorParm3[1]) = 0 THEN
16 2047 --                      Delete(gErrorParm3, 1, 1);
16 2048 -1                  END;
16 2049 --
16 2050 --              FailNewMessage(error, message, msgOpenFailed);
16 2051 -0 B        END;
16 2052 --
16 2053 -- B        FUNCTION IsOpen (itsID: INTEGER): BOOLEAN;
16 2054 --
16 2055 --          CONST
16 2056 --              UnitNtryCnt =   $01D2;
16 2057 --              UTableBase =    $011C;
16 2058 --
16 2059 --          TYPE
16 2060 --              UnitTable =     ARRAY [0..128] OF DCtlhandle;
16 2061 --              UnitTablePtr =  ^UnitTable;
16 2062 --
16 2063 --          VAR
16 2064 --              dceHnd:         DCtlHandle;
16 2065 --
16 2066 0- B        BEGIN
16 2067 --              IsOpen := FALSE;
16 2068 --              IF (itsID >= 0) AND (itsID < PInteger(UnitNtryCnt)^) THEN
16 2069 1-                  BEGIN
16 2070 --                  dceHnd := UnitTablePtr(PLongInt(UTableBase)^)^[itsId];
16 2071 --                  IF (dceHnd <> NIL) & BTST(dceHnd^^.dCtlFlags, 5) THEN
16 2072 --                      IsOpen := TRUE;
16 2073 -1                  END;
16 2074 -0 B        END;
16 2075 --
16 2076 0- A    BEGIN
16 2077 --          CatchFailures(fi, HdlOpenDeskAcc);
16 2078 --          aRefNum := 0;                           { Make sure failure handler works.  }
```

```
16 2079 --
16 2080 --          { Attempt to load the DA into memory.  If 'deskAccName' refers to another app   }
16 2081 --          { rather than a read desk acc, then GetNamedResource returns a faked up handle  }
16 2082 --          { courtesy of MultiFinder.  We open the DA with permanent allocation so as to   }
16 2083 --          { ensure that we don't take space from our code segments.                        }
16 2084 --
16 2085 --          oldPerm := PermAllocation(TRUE);
16 2086 --          drvrH := GetNamedResource('DRVR', deskAccName);
16 2087 --          oldPerm := PermAllocation(oldPerm);
16 2088 --
16 2089 --          IF drvrH = NIL THEN                          { Either there wasn't enough memory }
16 2090 --              IF ResError <> noErr THEN                { _to load the DA, or something is  }
16 2091 --                  FailResError                         { _seriously wrong.                 }
16 2092 --              ELSE
16 2093 --                  Failure(resNotFound, 0);
16 2094 --
16 2095 --          { At this point if we are really opening a DA we know it fits in memory.         }
16 2096 --
16 2097 --          GetResInfo(drvrH, theID, theType, theName); { If it's a not a real DA then this }
16 2098 --                                                       { will generate a ResError.         }
16 2099 --          ourHeap := HandleZone(drvrH) = ApplicZone;   { Find out which zone it lives in.  }
16 2100 --
16 2101 --          IF (ResError <> noErr) |                     { If it's a MultiFinder fake DA,    }
16 2102 --             IsOpen(theID) |                           { _or if the DA is already open,    }
16 2103 --             (NOT ourHeap) THEN                        { _or if it's not going in our heap }
16 2104 --              aRefNum := OpenDeskAcc(deskAccName)      { _then go ahead and open it.       }
16 2105 --
16 2106 --          ELSE
16 2107 --
16 2108 1-              BEGIN
16 2109 --              { If we get this far, we know we have a real DA and it's going into our     }
16 2110 --              { heap.  Open it, but them make sure we have enough memory to continue      }
16 2111 --              { running.                                                                  }
16 2112 --
16 2113 --              FailSpaceIsLow;                          { In case we're already low on mem. }
16 2114 --
16 2115 --              aRefNum := OpenDeskAcc(deskAccName);     { Use temporary allocation.         }
16 2116 --
16 2117 --              FailSpaceIsLow;                          { Fail if not enough memory left.   }
16 2118 --              FailNIL(drvrH^);                         { _or if the driver was purged to   }
16 2119 --                                                       { _satisfy a code space requirement.}
16 2120 -1             END;
16 2121 --
16 2122 --          Success(fi);
16 2123 --
16 2124 -0 A    END { TApplication.OpenDeskAccessory };
16 2125 --
16 2126 --      {$S MAOpen}
16 2127 --      {-----------------------------------+
16 2128 --      |   OpenNew                         |
16 2129 --      +-----------------------------------}
16 2130 -- A    PROCEDURE TApplication.OpenNew(itsCmdNumber: CmdNumber);
16 2131 --      VAR
16 2132 --          aDocument:  TDocument;
16 2133 --          fi:         FailInfo;
16 2134 --          newTitle:   Str255;
16 2135 --          aWindow:    TWindow;
16 2136 --
16 2137 --          {-------------------------------+
16 2138 --          |   HdlOpenNew                  |
16 2139 --          +-------------------------------}
16 2140 -- B        PROCEDURE HdlOpenNew(error: INTEGER; message: LONGINT);
16 2141 0- B        BEGIN
16 2142 --              FreeObject(aDocument);
16 2143 --              FailNewMessage(error, message, msgNewFailed);
16 2144 -0 B        END;
16 2145 --
16 2146 0- A    BEGIN
16 2147 --          aDocument := NIL;
16 2148 --          CatchFailures(fi, HdlOpenNew);
16 2149 --
16 2150 --          aDocument := DoMakeDocument(KindOfDocument(itsCmdNumber, NIL));
16 2151 --          aDocument.DoInitialState;
16 2152 --          aDocument.DoMakeViews(kForDisplay);
16 2153 --          aDocument.DoMakeWindows;
16 2154 --
16 2155 --          aDocument.UntitledName(newTitle);
16 2156 --                  { For MacApp 1.1, newTitle should be always <> '' }
16 2157 --          IF newTitle <> '' THEN
16 2158 --              aDocument.SetTitle(newTitle)
16 2159 --          ELSE IF aDocument.fWindowList.fSize > 0 THEN
16 2160 1-              BEGIN   { must set fTitle field anyways }
16 2161 --              aWindow := TWindow(aDocument.fWindowList.First);
16 2162 --
16 2163 --              GetWTitle(aWindow.fWmgrWindow, newTitle);
16 2164 --              aDocument.fTitle := Copy(newTitle,
16 2165 --                                       aWindow.fPreDocname,
16 2166 --                                       Length(newTitle)-aWindow.fConstTitle);
16 2167 -1             END;
16 2168 --
16 2169 --          AddDocument(aDocument);
16 2170 --
16 2171 --          FailSpaceIsLow;     { Fail if document leaves us with no room }
16 2172 --
16 2173 --          { Don't attempt to show the windows until we're sure we won't fail }
16 2174 --          aDocument.ShowWindows;
16 2175 --
16 2176 --          Success(fi);
16 2177 -0 A    END;
```

```
16 2178 --
16 2179 --      {$S MAOpen}
16 2180 --      {-----------------------------------+
16 2181 --      |   OpenOld                         |
16 2182 --      +-----------------------------------}
16 2183 -- A    PROCEDURE TApplication.OpenOld(itsOpenCmd: CmdNumber; anAppFile: AppFile);
16 2184 --              { Called for opening a document, given its name }
16 2185 --      VAR
16 2186 --          aDocument:      TDocument;
16 2187 --          otherDoc:       TDocument;
16 2188 --          oldCodeReserve,
16 2189 --          oldMemReserve:  Size;
16 2190 --          fi:             FailInfo;
16 2191 --
16 2192 --          {-------------------------------+
16 2193 --          |   HdlOpenOld                  |
16 2194 --          +-------------------------------}
16 2195 -- B        PROCEDURE HdlOpenOld(error: INTEGER; message: LONGINT);
16 2196 0- B       BEGIN
16 2197 --              FreeObject(aDocument);
16 2198 --
16 2199 --              IF message = 0 THEN
16 2200 --                  gErrorParm3 := anAppFile.fName;
16 2201 --              { Set the reserve back to where it was }
16 2202 --              SetReserveSize(oldCodeReserve, oldMemReserve);
16 2203 --              FailNewMessage(error, message, msgOpenFailed);
16 2204 -0 B       END;
16 2205 --
16 2206 0- A    BEGIN
16 2207 --          aDocument := NIL;
16 2208 --
16 2209 --          CatchFailures(fi, HdlOpenOld);
16 2210 --
16 2211 --          { Set reserve down a little to ensure that we can open existing documents }
16 2212 --          GetReserveSize(oldCodeReserve, oldMemReserve);
16 2213 --          SetReserveSize(oldCodeReserve, oldMemReserve DIV 2);
16 2214 --
16 2215 --          otherDoc := AlreadyOpen(anAppFile.fName, anAppFile.vRefnum);
16 2216 --          IF otherDoc <> NIL THEN
16 2217 --              otherDoc.OpenAgain(itsOpenCmd, aDocument);
16 2218 --
16 2219 --          aDocument := DoMakeDocument(KindOfDocument(itsOpenCmd, @anAppFile));
16 2220 --
16 2221 --          aDocument.ReadFromFile(anAppFile, kForDisplay);
16 2222 --          aDocument.DoMakeViews(kForDisplay);
16 2223 --          aDocument.DoMakeWindows;
16 2224 --          (** aDocument.SetTitle(aDocument.fTitle); **)
16 2225 --              { ??? The preceeding should not be necessary because ReadFromFile correctly sets
16 2226 --                      fTitle, and TWindow.IWindow correctly sets the window title based on the
16 2227 --                      fTitle field.   (Good style?) ??? }
16 2228 --          AddDocument(aDocument);
16 2229 --
16 2230 --          FailSpaceIsLow;     { Fail if the document leaves us with no memory }
16 2231 --          { Set the reserve back to where it was }
16 2232 --          SetReserveSize(oldCodeReserve, oldMemReserve);
16 2233 --
16 2234 --          { Don't attempt to show the windows until we're sure we won't fail }
16 2235 --          aDocument.ShowWindows;
16 2236 --
16 2237 --          Success(fi);
16 2238 -0 A    END;
16 2239 --
16 2240 --      {$S MARes}
16 2241 --      {-----------------------------------+
16 2242 --      |   PerformCommand                  |
16. 2243 --      +-----------------------------------}
16 2244 -- A    PROCEDURE TApplication.PerformCommand(command: TCommand);
16 2245 --      VAR
16 2246 --          fi:         FailInfo;
16 2247 --          saveCmd:    BOOLEAN;
16 2248 --
16 2249 --          {-------------------------------+
16 2250 --          |   HdlDoit                     |
16 2251 --          +-------------------------------}
16 2252 -- B        PROCEDURE HdlDoit(error: INTEGER; message: LONGINT);
16 2253 --
16 2254 0- B       BEGIN
16 2255 --              IF gClipClaimed THEN
16 2256 1-                  BEGIN
16 2257 --                  SetClipView(gClipUndoView);
16 2258 --                  gClipUndoView := NIL;
16 2259 --                  { The newly-installed view needs to be freed also }
16 2260 --              (*  SwapClipViews;  { Get original back there } { would be nice but doesn't do right thing yet } *)
16 2261 -1                  END;
16 2262 --
16 2263 --              command.Free;
16 2264 --              gLastCommand := NIL;
16 2265 --
16 2266 --              FailNewMessage(error, message, BuildMessage(command.fCmdNumber, msgCmdErr));
16 2267 -0 B       END;
16 2268 --
16 2269 0- A    BEGIN
16 2270 --          {$IFC qDebug}
16 2271 --          IF command = NIL THEN
16 2272 --              ProgramBreak('NIL passed to TApplication.PerformCommand')
16 2273 --          ELSE IF command = gNoChanges THEN
16 2274 --              ProgramBreak('gNoChanges passed to TAppliation.PerformCommand')
16 2275 --          ELSE
16 2276 --          {$ENDC}
```

```
16 2277 --          IF command <> gLastCommand THEN
16 2278 1-              BEGIN
16 2279 --              saveCmd := command.fCausesChange OR command.fCanUndo;
16 2280 --
16 2281 --              IF saveCmd THEN
16 2282 --                  CommitLastCommand;
16 2283 --
16 2284 --
16 2285 --              command.fTarget := gTarget;
16 2286 --
16 2287 --              CatchFailures(fi, HdlDoit);
16 2288 --              IF gEventLevel = 1 THEN     { Don't unload segs if in nested event handling }
16 2289 --                  UnloadAllSegments;
16 2290 --
16 2291 --              gClipClaimed := FALSE;
16 2292 --              command.Doit;
16 2293 --              Success(fi);
16 2294 --
16 2295 --              IF saveCmd THEN
16 2296 2-                 BEGIN
16 2297 --                  gLastCommand := command;
16 2298 --                  command.fCmdDone := TRUE;
16 2299 -2                 END;
16 2300 --
16 2301 --              WITH command DO
16 2302 --                  IF fCausesChange THEN { put this after .DoIt, so .DoIt can change this flag }
16 2303 2-                     BEGIN
16 2304 --                      IF fChangedDocument <> NIL THEN
16 2305 --                          WITH fChangedDocument DO
16 2306 --                              fChangeCount := fChangeCount + 1;
16 2307 -2                     END;
16 2308 --
16 2309 --              IF NOT saveCmd THEN
16 2310 --                  command.Free;
16 2311 -1              END;
16 2312 -0 A   END;
16 2313 --
16 2314 --      {$S MARes}
16 2315 --      {-----------------------------------+
16 2316 --      |   PollEvent                       |
16 2317 --      +-----------------------------------}
16 2318 -- A   PROCEDURE TApplication.PollEvent;
16 2319 --          LABEL   1000;
16 2320 --
16 2321 --      VAR
16 2322 --          fi:             FailInfo;
16 2323 --          theEvent:       EventRecord;
16 2324 --          aWindow:        TWindow;
16 2325 --
16 2326 --          {-------------------------------+
16 2327 --          |   HdlPollEvt                  |
16 2328 --          +-------------------------------}
16 2329 -- B       PROCEDURE HdlPollEvt(error: INTEGER; message: LONGINT);
16 2330 0- B       BEGIN
16 2331 --      {$IFC qDebug}
16 2332 --              Writeln;  { add a blank line after all the messages from Failure }
16 2333 --      {$ENDC}
16 2334 --              IF error <> noErr THEN
16 2335 1-                 BEGIN
16 2336 --                  IF gEventLevel = 1 THEN     { Don't unload segs if nested event handling.   }
16 2337 --                      UnloadAllSegments;
16 2338 --                  ShowError(error, message);
16 2339 -1                 END;
16 2340 --              HiliteMenu(0);                  { Make sure menu isn't left highlighted.        }
16 2341 --              GOTO 1000;                      { Keep the application running.                 }
16 2342 -0 B       END;
16 2343 --
16 2344 --
16 2345 0- A   BEGIN
16 2346 --          gEventLevel := gEventLevel + 1;
16 2347 --
16 2348 --      {$IFC qDebug}
16 2349 --          IF gTarget = NIL THEN
16 2350 --              WriteLn('Serious Error!!! in TApplication.PollEvent: target = NIL');
16 2351 --      {$ENDC}
16 2352 --
16 2353 --          CatchFailures(fi, HdlPollEvt);
16 2354 --
16 2355 --          IF GetEvent(gMainEventMask, gIdleFreq, gCursorRgn, theEvent) THEN
16 2356 1-              BEGIN
16 2357 --              IF gIdlePhase <> idleBegin THEN
16 2358 2-                  BEGIN
16 2359 --                  Idle(idleEnd);
16 2360 --                  gIdlePhase := idleBegin;
16 2361 -2                 END;
16 2362 --              HandleEvent(theEvent);
16 2363 --              HiliteMenu(0);
16 2364 --      {$IFC qDebug}
16 2365 --              gErrorParm3 := '?????';     { to prevent anyone from using old values }
16 2366 --      {$ENDC}
16 2367 -1              END
16 2368 --
16 2369 --          ELSE { idle }
16 2370 1-              BEGIN
16 2371 --              Idle(gIdlePhase);
16 2372 --              gIdlePhase := idleContinue;
16 2373 -1              END;
16 2374 --
16 2375 --          { The desk scrap may have been changed by use of Cmd-X or Cmd-C in
```

```
16 2376 --              desk accessories. }
16 2377 --          IF gSysWindowActive THEN
16 2378 --              CheckDeskScrap;
16 2379 --
16 2380 --          Success(fi);
16 2381 --
16 2382 --      1000:
16 2383 --          gEventLevel := gEventLevel - 1;
16 2384 -0 A    END;
16 2385 --
16 2386 --      {$S MARes}
16 2387 --      {-----------------------------------+
16 2388 --      |   PostHandleEvent                 |
16 2389 --      +-----------------------------------}
16 2390 -- A    PROCEDURE TApplication.PostHandleEvent (VAR theEventInfo: EventInfo);
16 2391 --
16 2392 --      VAR
16 2393 --          sysWindowAct:       BOOLEAN;
16 2394 --          perm:               BOOLEAN;
16 2395 --
16 2396 0- A    BEGIN
16 2397 --          IF gRedrawMenuBar THEN          { application wants menus to be setup }
16 2398 --              SetupTheMenus
16 2399 --          ELSE
16 2400 --              gMenusAreSetup := FALSE;    { menus may need to be setup }
16 2401 --
16 2402 --          perm := PermAllocation(FALSE);
16 2403 --      {$IFC qDebug}
16 2404 --          IF perm THEN
16 2405 --              ProgramBreak('The permanent flag was left TRUE.');
16 2406 --      {$ENDC}
16 2407 --
16 2408 --          { See if a system window has been activated or deactivated. }
16 2409 --          sysWindowAct := IsDeskAccessory(FrontWindow);
16 2410 --
16 2411 --          IF sysWindowAct <> gSysWindowActive THEN
16 2412 1-              BEGIN
16 2413 --              gSysWindowActive := sysWindowAct;
16 2414 --
16 2415 --              IF gSysWindowActive THEN            { deactivating to sys window }
16 2416 2-                  BEGIN
16 2417 --                  AboutToLoseControl(TRUE);
16 2418 --                  gRedrawMenuBar := TRUE;
16 2419 -2                  END
16 2420 --              ELSE                                { coming back from sys window }
16 2421 --                  RegainControl(TRUE);
16 2422 -1              END;
16 2423 --
16 2424 -0 A    END { TApplication.PostHandleEvent };
16 2425 --
16 2426 --      {$S MAFinder}
16 2427 --      {-----------------------------------+
16 2428 --      |   PrintDocument                   |
16 2429 --      +-----------------------------------}
16 2430 -- A    FUNCTION  TApplication.PrintDocument(anAppFile: AppFile): BOOLEAN;
16 2431 --      VAR
16 2432 --          aDocument:      TDocument;
16 2433 --          aPrintHandler:  TPrintHandler;
16 2434 --          proceed:        BOOLEAN;
16 2435 --          fi:             FailInfo;
16 2436 --
16 2437 --          {------------------------------+
16 2438 --          |   HdlPrintDoc                 |
16 2439 --          +------------------------------}
16 2440 -- B        PROCEDURE HdlPrintDoc(error: INTEGER; message: LONGINT);
16 2441 0- B        BEGIN
16 2442 --              FreeObject(aDocument);
16 2443 -0 B        END;
16 2444 --
16 2445 0- A    BEGIN
16 2446 --          aDocument := NIL;
16 2447 --          CatchFailures(fi, HdlPrintDoc);
16 2448 --
16 2449 --          aDocument := DoMakeDocument(KindOfDocument(cFinderPrint, @anAppFile));
16 2450 --          aDocument.ReadFromFile(anAppFile, kForPrinting);
16 2451 --          aDocument.DoMakeViews(kForPrinting);
16 2452 --
16 2453 --          { Note that if we are finder printing, this segment will be resident }
16 2454 --          UnloadAllSegments;
16 2455 --
16 2456 --          aPrintHandler := aDocument.fDocPrintHandler;
16 2457 --          IF aPrintHandler <> NIL THEN
16 2458 1-              BEGIN
16 2459 --              proceed := aPrintHandler.SetupForFinder;
16 2460 --              IF proceed THEN
16 2461 --                  IF aPrintHandler.Print(cFinderPrint, proceed) <> gNoChanges THEN
16 2462 --      {$IFC qDebug}
16 2463 --                      ProgramBreak('TApplication.PrintDocument: Print return a real command.')
16 2464 --      {$ENDC}         ;
16 2465 -1              END
16 2466 --          ELSE
16 2467 1-              BEGIN
16 2468 --              proceed := TRUE;    { might as well try the next one }
16 2469 --      {$IFC qDebug}
16 2470 --              ProgramBreak('TApplication.PrintDocument: The document's fDocPrintHandler is NIL.');
16 2471 --      {$ENDC}
16 2472 -1              END;
16 2473 --
16 2474 --          UnloadAllSegments;
```

```
16 2475 --          Success(fi);
16 2476 --
16 2477 --          aDocument.Free;
16 2478 --          UnloadAllSegments;
16 2479 --          PrintDocument := proceed;
16 2480 -0 A    END;
16 2481 --
16 2482 --      {$S MAClipboard}
16 2483 --      {-----------------------------------+
16 2484 --      |   ReadFromDeskScrap               |
16 2485 --      +-----------------------------------}
16 2486 -- A    PROCEDURE TApplication.ReadFromDeskScrap;
16 2487 --      LABEL 1000;
16 2488 --      VAR
16 2489 --          aViewForClipboard:  TView;
16 2490 --          fi:                 FailInfo;
16 2491 --
16 2492 --          {-------------------------------+
16 2493 --          |   HdlMakeViewForAlienClipbd   |
16 2494 --          +-------------------------------}
16 2495 -- B        PROCEDURE HdlMakeViewForAlienClipbd(error: OSErr; message: LONGINT);
16 2496 0- B        BEGIN
16 2497 --
16 2498 --              aViewForClipboard := gClipOrphanage;
16 2499 --              IF message = 0 THEN
16 2500 --                  message := msgImportClipFailed;
16 2501 --              ShowError(error, message);
16 2502 --              GOTO 1000;
16 2503 -0 B        END;
16 2504 --
16 2505 0- A    BEGIN
16 2506 --          CatchFailures(fi, HdlMakeViewForAlienClipbd);
16 2507 --          aViewForClipboard := MakeViewForAlienClipboard;
16 2508 --          FailNil(aViewForClipboard);
16 2509 --          Success(fi);
16 2510 --      1000:
16 2511 --          ClaimClipboard(aViewForClipboard);
16 2512 -0 A    END;
16 2513 --
16 2514 --      {$S MARes}
16 2515 --      {-----------------------------------+
16 2516 --      |   RegainControl                   |
16 2517 --      +-----------------------------------}
16 2518 -- A    PROCEDURE TApplication.RegainControl(checkClipboard: BOOLEAN);
16 2519 0- A    BEGIN
16 2520 --          BusyActivate(TRUE);
16 2521 --          IF checkClipboard THEN
16 2522 --              CheckDeskScrap;
16 2523 -0 A    END;
16 2524 --
16 2525 --      {$IFC qDebug}
16 2526 --      {$S MADebug}
16 2527 --          { Debugging procedure: given an EventRecord, prints out information about the event. }
16 2528 --      {$IFC qTrace}{$D+}{$ENDC}
16 2529 --      {-----------------------------------+
16 2530 --      |   ReportEvent                     |
16 2531 --      +-----------------------------------}
16 2532 -- A    PROCEDURE TApplication.ReportEvent(VAR theEventInfo: EventInfo);
16 2533 --      VAR
16 2534 --          ch:     INTEGER;
16 2535 --          cap:    INTEGER;
16 2536 --          wName:  Str255;
16 2537 --          mods:   STRING[10];
16 2538 --
16 2539 0- A    BEGIN
16 2540 --          WITH theEventInfo, thePEvent^ DO
16 2541 1-              BEGIN
16 2542 --              Write('t = ', when);
16 2543 --              mods := '          ';
16 2544 --                      { 1234567890 };
16 2545 --              IF theControlKey THEN
16 2546 --                  mods[2] := 'C';
16 2547 --              IF theOptionKey THEN
16 2548 --                  mods[3] := 'O';
16 2549 --              IF theAlphaLock THEN
16 2550 --                  mods[4] := 'L';
16 2551 --              IF theShiftKey THEN
16 2552 --                  mods[5] := 'S';
16 2553 --              IF theCmdKey THEN
16 2554 --                  mods[6] := 'C';
16 2555 --              IF theBtnState THEN
16 2556 --                  mods[7] := 'M';
16 2557 --              IF what = activateEvt THEN
16 2558 --                  IF BAND(modifiers, activeFlag) <> 0 THEN
16 2559 --                      mods[8] := 'A'
16 2560 --                  ELSE
16 2561 --                      mods[8] := 'D';
16 2562 --              Write(mods);
16 2563 --
16 2564 2-              CASE what OF
16 2565 --                  nullEvent:
16 2566 --                      WriteLn('nullEvent   ');
16 2567 --                  mouseDown, mouseUp:
16 2568 3-                      BEGIN
16 2569 --                      IF what = mouseDown THEN
16 2570 --                          Write('mouseDown   ')
16 2571 --                      ELSE
16 2572 --                          Write('mouseUp     ');
16 2573 --                      WriteLn('@ (', where.h:1, ', ', where.v:1, ')');
```

```
16 2574 -3                  END;
16 2575 --                keyDown, autoKey, keyUp:
16 2576 3-                  BEGIN
16 2577 --                  IF what = keyDown THEN
16 2578 --                      Write('keyDown     ')
16 2579 --                  ELSE IF what = autoKey THEN
16 2580 --                      Write('autoKey     ')
16 2581 --                  ELSE
16 2582 --                      Write('keyUp       ');
16 2583 --
16 2584 --                  ch := BAND(message, charCodeMask);
16 2585 --                  cap := BSR(message, 8);
16 2586 --
16 2587 --                  IF (ch >= $20) AND (ch <= $D8) AND (ch <> $7F) THEN
16 2588 --                      Write('"', CHR(ch), '"')
16 2589 --                  ELSE
16 2590 --                      Write('   ');
16 2591 --
16 2592 --                  WriteLn('(', ch:1, '/', cap:1, ')');
16 2593 -3                  END;
16 2594 --                updateEvt, activateEvt:
16 2595 3-                  BEGIN
16 2596 --                  IF what = updateEvt THEN
16 2597 --                      Write('updateEvt   ')
16 2598 --                  ELSE
16 2599 --                      Write('activateEvt ');
16 2600 --
16 2601 --                  wName := WindowPeek(message)^.titleHandle^^;
16 2602 --                  WriteLn('"', wName, '"');
16 2603 -3                  END;
16 2604 --                diskEvt:
16 2605 --                  WriteLn('diskEvt     ', 'd = ', LoWord(message):1, ' e = ', HiWord(message):1);
16 2606 --                networkEvt:
16 2607 3-                  BEGIN
16 2608 --                  Write  ('networkEvt  ');
16 2609 --                  WritePtr(message);
16 2610 --                  Writeln;
16 2611 -3                  END;
16 2612 --                driverEvt:
16 2613 --                  WriteLn('driverEvt   ', message);
16 2614 --                appl Evt:
16 2615 --                  WriteLn('appl Evt    ', message);
16 2616 --                app2Evt:
16 2617 --                  WriteLn('app2Evt     ', message);
16 2618 --                app3Evt:
16 2619 --                  WriteLn('app3Evt     ', message);
16 2620 --                app4Evt:
16 2621 --                  IF BAND(message, $FF000000) = kSuspendOrResume THEN
16 2622 --                      IF ODD(message) THEN
16 2623 --                          Writeln('switch in   ', message)
16 2624 --                      ELSE
16 2625 --                          Writeln('switch out  ', message)
16 2626 --                  ELSE IF BAND(message, $FF000000) = kMouseMovedMessage THEN
16 2627 --                      WriteLn('mouse moved ', message)
16 2628 --                  ELSE
16 2629 --                      WriteLn('app4Evt     ', message);
16 2630 --                OTHERWISE
16 2631 --                  Writeln('??? unknown = ', what:1, '   ', message:1);
16 2632 -2                END;
16 2633 -1            END;
16 2634 -0 A   END;
16 2635 --     {$IFC qTrace}{$D++}{$ENDC}
16 2636 --     {$ENDC qDebug}
16 2637 --
16 2638 --     {$IFC qDebug}
16 2639 --     {$S MADebug}
16 2640 --     {$IFC qTrace}{$D+}{$ENDC}
16 2641 --         { Debugging procedure: given the result of FindWindow, WriteLn the location of the mouse. }
16 2642 --     {-----------------------------------+
16 2643 --     |   ReportMouseDown                 |
16 2644 --     +-----------------------------------}
16 2645 -- A   PROCEDURE TApplication.ReportMouseDown(where: INTEGER);
16 2646 --     VAR
16 2647 --         str:    Str255;
16 2648 0- A   BEGIN
16 2649 1-         CASE where OF
16 2650 --             inMenuBar:
16 2651 --                 str := 'inMenuBar';
16 2652 --             inSysWindow:
16 2653 --                 str := 'inSysWindow';
16 2654 --             inDrag:
16 2655 --                 str := 'inDrag';
16 2656 --             inGrow:
16 2657 --                 str := 'inGrow';
16 2658 --             inGoAway:
16 2659 --                 str := 'inGoAway';
16 2660 --             inContent:
16 2661 --                 str := 'inContent';
16 2662 --             inZoomIn:
16 2663 --                 str := 'inZoomIn';
16 2664 --             inZoomOut:
16 2665 --                 str := 'inZoomOut';
16 2666 --             OTHERWISE
16 2667 2-                 BEGIN
16 2668 --                 Writeln(' ':30, where:1);
16 2669 --                 str := 'Mouse clicked in an unknown place.'
16 2670 -2                 END;
16 2671 -1             END;  { CASE }
16 2672 --         WriteLn(' ':30, str);
```

```
16 2673 -0 A   END;
16 2674 --     {$IFC qTrace}{$D++}{$ENDC}
16 2675 --     {$ENDC qDebug}
16 2676 --
16 2677 --     {$S Main} { must be in main segment }
16 2678 --     {-----------------------------------+
16 2679 --     |   Run                             |
16 2680 --     +-----------------------------------}
16 2681 -- A   PROCEDURE TApplication.Run;
16 2682 --     VAR
16 2683 --         findSeg:    INTEGER;
16 2684 0- A   BEGIN
16 2685 --         UnloadAllSegments;
16 2686 --
16 2687 --         FailSpaceIsLow; { make sure we have enough memory to continue }
16 2688 --
16 2689 --         gInitialized := TRUE; { was set FALSE in InitToolBox }
16 2690 --         SetInitHandler(NIL);
16 2691 --
16 2692 --         IF NOT gFinderPrinting THEN
16 2693 --             LaunchClipboard;
16 2694 --         UnloadAllSegments;
16 2695 --
16 2696 --         findSeg := GetSegNumber(@FinderSegProc);
16 2697 --         IF gFinderPrinting THEN
16 2698 --             SetResidentSegment(findSeg, TRUE);
16 2699 --
16 2700 --         HandleFinderRequest;
16 2701 --
16 2702 --         IF gFinderPrinting THEN
16 2703 --             SetResidentSegment(findSeg, FALSE);
16 2704 --         UnloadAllSegments;
16 2705 --
16 2706 --         IF NOT gFinderPrinting THEN
16 2707 1-             BEGIN
16 2708 --             gRedrawMenuBar := TRUE;     { force setup/redraw of menu bar }
16 2709 --             SetupTheMenus;
16 2710 --
16 2711 --             MainEventLoop;
16 2712 --
16 2713 --             AboutToLoseControl(TRUE);
16 2714 -1             END
16 2715 --         ELSE
16 2716 --             Close;                      { Close is always called when quitting app }
16 2717 --
16 2718 --
16 2719 --     {$IFC qDebug}
16 2720 --         { See if previous max. resource usage has been exceeded by the termi-
16 2721 --             nation code and resources. }
16 2722 --         CheckRsrcUsage;
16 2723 --     {$ENDC}
16 2724 --
16 2725 --         { We must call CleanupMacApp here; if we wait to fall thru to the end of the
16 2726 --             main program, A5 has been invalidated and we can't refer to any globals. }
16 2727 --         CleanupMacApp;
16 2728 -0 A   END;
16 2729 --
16 2730 --     {$S MARes}
16 2731 --     {-----------------------------------+
16 2732 --     |   SelectWMgrWindow                |
16 2733 --     +-----------------------------------}
16 2734 -- A   PROCEDURE TApplication.SelectWMgrWindow (aWMgrWindow: WindowPtr);
16 2735 --
16 2736 0- A   BEGIN
16 2737 --         SelectWindow(aWMgrWindow);      { Simply call the toolbox to select it.     }
16 2738 --         gLastClickPart := inDesk;       { Make sure previous mouse clicks are not   }
16 2739 --                                         { are not considered part of a multi-click. }
16 2740 -0 A   END;
16 2741 --
16 2742 --     {$S MAClipboard}
16 2743 --     {-----------------------------------+
16 2744 --     |   SetClipView                     |
16 2745 --     +-----------------------------------}
16 2746 -- A   PROCEDURE TApplication.SetClipView(clipView: TView);
16 2747 --
16 2748 --     VAR
16 2749 --         theSuperView:       TView;
16 2750 --
16 2751 -- B       PROCEDURE RemoveView (aView: TView);
16 2752 --
16 2753 0- B       BEGIN
16 2754 --             theSuperView.RemoveSubView(aView);
16 2755 -0 B       END;
16 2756 --
16 2757 0- A   BEGIN
16 2758 --         IF gClipWindow <> NIL THEN
16 2759 1-             BEGIN
16 2760 --             IF gClipWindow.CountSubViews > 0 THEN
16 2761 --                 theSuperView := TView(gClipWindow.fSubViews.First)
16 2762 --             ELSE
16 2763 --                 theSuperView := gClipWindow;
16 2764 --             theSuperView.EachSubView(RemoveView);
16 2765 --             theSuperView.AddSubView(clipView);
16 2766 --             clipView.fSuperView := theSuperView;
16 2767 --             clipView.SuperViewChangedSize(gZeroVPt, FALSE);
16 2768 --             clipView.RevealTop(kDontRedraw);
16 2769 --             gClipWindow.ForceRedraw;
16 2770 --             gClipWindow.fTarget := gClipWindow;
16 2771 --             gClipWrittenToDeskScrap := clipView = gClipOrphanage;
```

```
16 2772 -1              END
16 2773 --          ELSE
16 2774 1-              BEGIN
16 2775 --      {$IFC qDebug}
16 2776 --              ProgramBreak('SetClipView in absence of gClipWindow');
16 2777 --      {$ENDC}
16 2778 -1              END;
16 2779 --
16 2780 --          gClipView := clipView;
16 2781 -0 A   END;
16 2782 --
16 2783 --      {$S MARes}
16 2784 --      {-----------------------------------+
16 2785 --      |   SetTarget                       |
16 2786 --      +-----------------------------------}
16 2787 -- A   PROCEDURE TApplication.SetTarget (newTarget: TEvtHandler);
16 2788 --
16 2789 0- A   BEGIN
16 2790 --          gTarget.InstallSelection(TRUE, FALSE);
16 2791 --          newTarget.InstallSelection(FALSE, TRUE);
16 2792 --          gTarget := newTarget;
16 2793 --          gIdleFreq := 0;             { Make sure we idle ASAP because there's a new target.  }
16 2794 --          SetEmptyRgn(gCursorRgn);    { Make sure cursor region gets set as well.             }
16 2795 -0 A   END { TApplication.SetTarget };
16 2796 --
16 2797 --      {$S MARes}
16 2798 --      {-----------------------------------+
16 2799 --      |   SetUndoText                     |
16 2800 --      +-----------------------------------}
16 2801 -- A   PROCEDURE TApplication.SetUndoText(cmdDone: BOOLEAN; aCmdNumber: CmdNumber);
16 2802 --      VAR
16 2803 --          newMenuState:   INTEGER;
16 2804 --          undoName:       Str255;
16 2805 --          cmdName:        Str255;
16 2806 --          preCmdName:     INTEGER;
16 2807 --          constChars:     INTEGER;
16 2808 0- A   BEGIN
16 2809 --          IF (gUndoState <> cmdDone) OR (gUndoCmd <> aCmdNumber) THEN
16 2810 1-              BEGIN
16 2811 --              IF aCmdNumber = cCantUndo THEN
16 2812 --                  newMenuState := bzCantUndo
16 2813 --              ELSE IF cmdDone THEN
16 2814 --                  newMenuState := bzUndo
16 2815 --              ELSE
16 2816 --                  newMenuState := bzRedo;
16 2817 --
16 2818 --              GetIndString(undoName, kIDBuzzString, newMenuState);
16 2819 --              IF ParseTitleTemplate(undoName, preCmdName, constChars) THEN
16 2820 2-                  BEGIN
16 2821 --                  IF (aCmdNumber = cNoCommand) OR (aCmdNumber = cCantUndo) THEN
16 2822 --                      cmdName := ''
16 2823 --                  ELSE
16 2824 --                      CmdToName(aCmdNumber, cmdName);
16 2825 --                  IF SubstituteInTitle(undoName, cmdName, preCmdName, constChars) THEN;
16 2826 -2                  END;
16 2827 --
16 2828 --              SetCmdName(cUndo, undoName);
16 2829 --
16 2830 --              gUndoState := cmdDone;
16 2831 --              gUndoCmd := aCmdNumber;
16 2832 -1              END;
16 2833 -0 A   END;
16 2834 --
16 2835 --      {$S MARes}
16 2836 --      {-----------------------------------+
16 2837 --      |   SetupTheMenus                   |
16 2838 --      +-----------------------------------}
16 2839 -- A   PROCEDURE TApplication.SetupTheMenus;
16 2840 --
16 2841 --          {-------------------------------+
16 2842 --          |   DoSetup                     |
16 2843 --          +-------------------------------}
16 2844 -- B       PROCEDURE DoSetup;
16 2845 --
16 2846 --          VAR
16 2847 --              appleMenu:      MenuHandle;
16 2848 --              undoState:      BOOLEAN;
16 2849 --              undoCmd:        CmdNumber;
16 2850 --
16 2851 0- B       BEGIN
16 2852 --              gGotClipType := FALSE;
16 2853 --
16 2854 --              gTarget.DoSetupMenus;                   { Setup menus relevent to target chain  }
16 2855 --
16 2856 --              { Set up the menu commands that are not dependent on the target chain… }
16 2857 --
16 2858 --              appleMenu := GetMHandle(mApple);
16 2859 --              WITH appleMenu^^ DO
16 2860 --                  IF ODD(enableFlags) = InModalState THEN
16 2861 1-                      BEGIN
16 2862 --                      enableFlags := BXOR(enableFlags, 1);
16 2863 --                      gRedrawMenuBar := TRUE;
16 2864 -1                      END;
16 2865 --
16 2866 --              undoState := kShowCantUndo;             { Set the Undo menu defaults.           }
16 2867 --              undoCmd := cCantUndo;
16 2868 --              IF gSysWindowActive THEN
16 2869 1-                  BEGIN
16 2870 --                  undoState := kShowUndo;
```

```
16 2871 --                  undoCmd := cNoCommand;
16 2872 -1                  END
16 2873 --              ELSE IF gLastCommand <> NIL THEN
16 2874 --                  WITH gLastCommand DO
16 2875 --                      IF fCanUndo THEN
16 2876 1-                          BEGIN
16 2877 --                          IF fCmdDone THEN
16 2878 --                              undoState := kShowUndo
16 2879 --                          ELSE
16 2880 --                              undoState := kShowRedo;
16 2881 --                          undoCmd := fCmdNumber;
16 2882 --
16 2883 --                          { Enable Undo only if the last command was not document-specific
16 2884 --                              or the document changed is the current document. }
16 2885 --                          Enable(cUndo, (fChangedDocument = NIL) OR (fChangedDocument = gDocument));
16 2886 -1                          END;
16 2887 --              SetUndoText(undoState, undoCmd);
16 2888 --
16 2889 --          {$IFC qDebug}
16 2890 --              EnableCheck(cExperimenting, TRUE, gExperimenting);
16 2891 --              EnableCheck(cReportEvt, TRUE, gReportEvt);
16 2892 --              EnableCheck(cDebugPrinting, TRUE, gDebugPrinting);
16 2893 --              EnableCheck(cReportMenuChoices, TRUE, gReportMenuChoices);
16 2894 --              EnableCheck(cIntenseDebugging, TRUE, gIntenseDebugging);
16 2895 --              Enable(cIdentifySoftware, TRUE);
16 2896 --
16 2897 --              IF gFrontWindow <> NIL THEN
16 2898 1-                  BEGIN
16 2899 --                  Enable(cModalToggle, TRUE);
16 2900 --                  SetMenuState(cModalToggle, kDebugBuzzStrings, bzMakeModal, bzMakeModeless,
16 2901 --                          gFrontWindow.fIsModal);
16 2902 --                  Enable(cRefreshFrontWindow, TRUE);
16 2903 --                  Enable(cDoFirstClick, TRUE);
16 2904 --                  SetMenuState(cDoFirstClick, kDebugBuzzStrings, bzDoFirstClick,
16 2905 --                          bzDontDoFirstClick, gFrontWindow.fDoFirstClick);
16 2906 -1                  END;
16 2907 --          {$ENDC}
16 2908 --          {$IFC qTrace}
16 2909 --              EnableCheck(cTraceSetupMenus, TRUE, gTraceSetupMenus);
16 2910 --              EnableCheck(cTraceIdle, TRUE, gTraceIdle);
16 2911 --          {$ENDC}
16 2912 --
16 2913 --              IF NOT gSysWindowActive THEN
16 2914 --                  Enable(cPaste, gGotClipType);
16 2915 -0 B       END { DoSetup };
16 2916 --
16 2917 0- A   BEGIN
16 2918 --          IF NOT gMenusAreSetup OR gRedrawMenuBar THEN
16 2919 --              PerformMenuSetup(DoSetup);
16 2920 -0 A   END;
16 2921 --
16 2922 --      {$S MAOpen}
16 2923 --      {-----------------------------------+
16 2924 --      |   SFGetParms                      |
16 2925 --      +-----------------------------------}
16 2926 -- A   PROCEDURE TApplication.SFGetParms(itsCmdNumber: CmdNumber; VAR dlgID: INTEGER;
16 2927 --              VAR where: Point; VAR fileFilter, dlgHook, filterProc: ProcPtr;
16 2928 --              typeList: HTypeList);
16 2929 --      CONST
16 2930 --          kStdSFWhere = 100 * $10000 + 100;
16 2931 --
16 2932 0- A   BEGIN
16 2933 --          dlgID := getDlgID;
16 2934 --          where := Point(kStdSFWhere);
16 2935 --          fileFilter := NIL;
16 2936 --          dlgHook := NIL;
16 2937 --          filterProc := NIL;
16 2938 --          SetHandleSize(Handle(typeList), 4);
16 2939 --          typeList^^[1] := gMainFileType;
16 2940 -0 A   END;
16 2941 --
16 2942 --      {$S MAError}
16 2943 --      {-----------------------------------+
16 2944 --      |   ShowError                       |
16 2945 --      +-----------------------------------}
16 2946 -- A   PROCEDURE TApplication.ShowError(error: OSErr; message: LONGINT);
16 2947 0- A   BEGIN
16 2948 --          ErrorAlert(error, message);
16 2949 -0 A   END;
16 2950 --
16 2951 --      {$S Main}
16 2952 --      {-----------------------------------+
16 2953 --      |   SpaceIsLow                      |
16 2954 --      +-----------------------------------}
16 2955 -- A   PROCEDURE TApplication.SpaceIsLow;
16 2956 --      VAR
16 2957 --          now:    LONGINT;
16 2958 0- A   BEGIN
16 2959 --          IF gEventLevel = 1 THEN     { Don't unload segs if nested event handling }
16 2960 --              UnloadAllSegments;
16 2961 --
16 2962 --          { Show 'space is low' alert only after ever gLowSpaceInterval ticks. }
16 2963 --          IF gLowSpaceInterval > 0 THEN
16 2964 1-              BEGIN
16 2965 --              now := TickCount;
16 2966 --              IF now > gNextSpaceMsg THEN
16 2967 2-                  BEGIN
16 2968 --                  StdAlert(phSpaceIsLow);
16 2969 --                  gNextSpaceMsg := now + gLowSpaceInterval;
```

```
16 2970 -2                  END;
16 2971 -1              END;
16 2972 -0 A    END;
16 2973 --
16 2974 --      {$S MAClipboard}
16 2975 --      {-----------------------------------+
16 2976 --      |   SwapClipViews                   |
16 2977 --      +-----------------------------------}
16 2978 -- A    PROCEDURE TApplication.SwapClipViews;
16 2979 --      VAR
16 2980 --          tempClipView: TView;
16 2981 0- A    BEGIN
16 2982 --          tempClipView := gClipUndoView;
16 2983 --          gClipUndoView := gClipView;
16 2984 --
16 2985 --          IF tempClipView <> NIL THEN
16 2986 --              SetClipView(tempClipView)   { Installs old Undo clipboard as current clipboard }
16 2987 --      {$IFC qDebug}
16 2988 --          ELSE
16 2989 --              ProgramBreak('SwapClipViews finds undo clipboard was NIL')
16 2990 --      {$ENDC}
16 2991 --                  ;
16 2992 -0 A    END;
16 2993 --
16 2994 --      {$S MARes}
16 2995 --      {-----------------------------------+
16 2996 --      |   TrackCursor                     |
16 2997 --      +-----------------------------------}
16 2998 -- A    FUNCTION  TApplication.TrackCursor: BOOLEAN;
16 2999 --
16 3000 --      VAR
16 3001 --          globalMouse:    Point;
16 3002 --          cursorIsSet:    BOOLEAN;
16 3003 --          aWMgrWindow:    WindowPtr;
16 3004 --          cursorWindow:   TWindow;
16 3005 --          cursorView:     TView;
16 3006 --          windowVPt:      VPoint;
16 3007 --          windowBounds:   Rect;
16 3008 --          r:              Rect;
16 3009 --          haveCursorRgn:  BOOLEAN;
16 3010 --
16 3011 0- A    BEGIN
16 3012 --          IF (gFrontWindow = NIL) OR gInBackground THEN
16 3013 --              EXIT(TrackCursor);
16 3014 --
16 3015 --          GetMouse(globalMouse);
16 3016 --          LocalToGlobal(globalMouse);
16 3017 --
16 3018 --          IF PtInRgn(globalMouse, gCursorRgn) THEN
16 3019 1-              BEGIN
16 3020 --      {$IFC qDebug}
16 3021 --              IF gIntenseDebugging AND gTraceIdle THEN
16 3022 --                  WRITELN('cursor is in cursor region');
16 3023 --      {$ENDC}
16 3024 --              EXIT(TrackCursor);
16 3025 -1              END;
16 3026 --
16 3027 --          IF (FindWindow(globalMouse, aWMgrWindow) = inContent) THEN
16 3028 1-              BEGIN
16 3029 --              { The cursor is in a window--for our purposes, it must be the front
16 3030 --                  window or the window must handle first clicks }
16 3031 --              cursorWindow := WMgrToWindow(aWMgrWindow);
16 3032 --              IF (cursorWindow <> gFrontWindow) & (NOT cursorWindow.fDoFirstClick) THEN
16 3033 --                  cursorWindow := NIL;
16 3034 -1              END
16 3035 --          ELSE
16 3036 --              cursorWindow := NIL;
16 3037 --
16 3038 --          SetEmptyRgn(gCursorRgn);
16 3039 --          haveCursorRgn := FALSE;
16 3040 --          cursorIsSet := FALSE;
16 3041 --          IF cursorWindow <> NIL THEN
16 3042 1-              BEGIN
16 3043 --              cursorWindow.GetGlobalBounds(windowBounds);
16 3044 --              windowVPt.h := globalMouse.h - windowBounds.left;
16 3045 --              windowVPt.v := globalMouse.v - windowBounds.top;
16 3046 --              cursorView := cursorWindow.HandleCursor(windowVPt, gCursorRgn);
16 3047 --              IF cursorView <> NIL THEN
16 3048 2-                  BEGIN
16 3049 --                  cursorIsSet := TRUE;
16 3050 --
16 3051 --                  IF (NOT EmptyRgn(gCursorRgn)) & (cursorView.CountSubViews = 0) THEN
16 3052 3-                      BEGIN
16 3053 --                      { Intersect with viewed rect }
16 3054 --                      cursorView.GetVisibleRect(r);
16 3055 --      {$IFC qDebug}
16 3056 --                      UseTempRgn('TApplication.TrackCursor');
16 3057 --      {$ENDC}
16 3058 --                      RectRgn(gTempRgn, r);
16 3059 --                      SectRgn(gTempRgn, gCursorRgn, gCursorRgn);
16 3060 --      {$IFC qDebug}
16 3061 --                      DoneWithTempRgn;
16 3062 --      {$ENDC}
16 3063 --                      { Convert gCursorRgn from view coords to global coords }
16 3064 --                      WITH thePort^.portRect DO
16 3065 --                          OffsetRgn(gCursorRgn, windowBounds.left - left,
16 3066 --                                                windowBounds.top - top);
16 3067 --                      haveCursorRgn := TRUE;
16 3068 -3                      END;
```

```
16 3069 -2                END;
16 3070 -1            END;
16 3071 --
16 3072 --        IF NOT haveCursorRgn THEN
16 3073 --            WITH globalMouse DO
16 3074 --                SetRectRgn(gCursorRgn, h, v, h + 1, v + 1);
16 3075 --
16 3076 --    {$IFC qDebug}
16 3077 --        IF gIntenseDebugging AND gTraceIdle THEN
16 3078 --            IF gCursorRgn = NIL THEN
16 3079 --                WRITELN('gCursorRgn is NIL')
16 3080 --            ELSE
16 3081 1-                BEGIN
16 3082 --                HLock(Handle(gCursorRgn));
16 3083 --                WrLblRect('gCursorRgn', gCursorRgn^^.RgnBBox);
16 3084 --                WRITELN;
16 3085 --                HUnlock(Handle(gCursorRgn));
16 3086 -1                END;
16 3087 --    {$ENDC}
16 3088 --
16 3089 --        IF NOT cursorIsSet THEN
16 3090 --            SetCursor(arrow);
16 3091 --        TrackCursor := cursorIsSet;
16 3092 -0 A    END;
16 3093 --
16 3094 --    {$S MADoCommand}
16 3095 --    {-----------------------------------+
16 3096 --    |   TrackMouse                      |
16 3097 --    +-----------------------------------}
16 3098 -- A    FUNCTION  TApplication.TrackMouse (globalMouse, hysteresis: Point; theCommand: TCommand): TCommand;
16 3099 --
16 3100 --    VAR
16 3101 --        tracker:            TCommand;
16 3102 --        view:               TView;
16 3103 --        scroller:           TScroller;
16 3104 --        gotATracker:        BOOLEAN;
16 3105 --        theQDMouse:         Point;
16 3106 --        theMouse:           VPoint;
16 3107 --        anchorPoint:        VPoint;
16 3108 --        previousPoint:      VPoint;
16 3109 --        peekEvent:          EventRecord;
16 3110 --        movedOnce:          BOOLEAN;
16 3111 --        amtMoved:           VPoint;
16 3112 --        didMove:            BOOLEAN;
16 3113 --        delta:              VPoint;
16 3114 --        didScroll:          BOOLEAN;
16 3115 --        currTranslation:    VPoint;
16 3116 --        viewExtent:         VRect;
16 3117 --        autoscrollLimit:    VRect;
16 3118 --
16 3119 --        {-------------------------------+
16 3120 --        |   SetupFocus                  |
16 3121 --        +-------------------------------}
16 3122 -- B      PROCEDURE SetupFocus;
16 3123 --        VAR
16 3124 --            wmgrPort:       GrafPtr;
16 3125 0- B      BEGIN
16 3126 --            IF view <> NIL THEN
16 3127 1-                BEGIN
16 3128 --                IF view.Focus THEN
16 3129 2-                    BEGIN
16 3130 --                    GetFocus;
16 3131 --                    IF scroller <> NIL THEN
16 3132 3-                        BEGIN
16 3133 --                        scroller.GetExtent(autoScrollLimit);
16 3134 --    {$IFC FALSE}
16 3135 --                        AddVPt(fAutoScrollMargin.topLeft, autoscrollLimit.topLeft);
16 3136 --                        AddVPt(fAutoScrollMargin.botRight, autoscrollLimit.botRight);
16 3137 --    {$ENDC}
16 3138 --                        currTranslation := scroller.fTranslation;
16 3139 -3                        END;
16 3140 -2                    END
16 3141 --    {$IFC qDebug}
16 3142 --                ELSE
16 3143 --                    ProgramBreak('TApplication.TrackMouse: Unable to focus view.')
16 3144 --    {$ENDC}
16 3145 --                    ;
16 3146 -1                END
16 3147 --            ELSE
16 3148 --                { focus on the desktop }
16 3149 1-                BEGIN
16 3150 --                GetWmgrPort(wmgrPort);
16 3151 --                SetPort(wmgrPort);
16 3152 --                ClipRect(GetGrayRgn^^.rgnBBox);
16 3153 --                gFocusedView := NIL;
16 3154 --                GetFocus;
16 3155 -1                END;
16 3156 -0 B      END;
16 3157 --
16 3158 --        {-------------------------------+
16 3159 --        |   DoFocus                     |
16 3160 --        +-------------------------------}
16 3161 -- B      PROCEDURE DoFocus;
16 3162 0- B      BEGIN
16 3163 --            IF (scroller <> NIL) & ((currTranslation.h <> scroller.fTranslation.h) OR
16 3164 --                                    (currTranslation.v <> scroller.fTranslation.v)) THEN
16 3165 --                SetupFocus
16 3166 --            ELSE
16 3167 --                SetFocus;
```

```
16 3168 -0 B        END;
16 3169 --
16 3170 --          {------------------------------+
16 3171 --          |   InstallTracker             |
16 3172 --          +------------------------------}
16 3173 -- B        PROCEDURE InstallTracker(newTracker: TCommand);
16 3174 0- B        BEGIN
16 3175 --              tracker := newTracker;
16 3176 --              gotATracker := (tracker <> gNoChanges);
16 3177 --              IF gotATracker THEN
16 3178 1-                  BEGIN
16 3179 --                  view := tracker.fView;
16 3180 --                  scroller := tracker.fScroller;
16 3181 --                  IF view <> NIL THEN
16 3182 --                      view.GetExtent(viewExtent);
16 3183 -1                  END;
16 3184 -0 B        END;
16 3185 --
16 3186 --          {------------------------------+
16 3187 --          |   FeedbackOnce               |
16 3188 --          +------------------------------}
16 3189 -- B        PROCEDURE FeedbackOnce(turnItOn, mouseDidMove: BOOLEAN);
16 3190 0- B        BEGIN
16 3191 --              IF gotATracker THEN
16 3192 1-                  BEGIN
16 3193 --                  PenNormal;
16 3194 --                  PenMode(PatXOR);
16 3195 --                  tracker.TrackFeedback(anchorPoint, previousPoint, turnItOn, mouseDidMove);
16 3196 -1                  END;
16 3197 -0 B        END;
16 3198 --
16 3199 --          {------------------------------+
16 3200 --          |   ConstrainOnce              |
16 3201 --          +------------------------------}
16 3202 -- B        PROCEDURE ConstrainOnce; { ??? fold this into TrackOnce ??? }
16 3203 0- B        BEGIN
16 3204 --              IF gotATracker THEN
16 3205 1-                  BEGIN
16 3206 --                  IF tracker.fViewConstrain & (view <> NIL) THEN
16 3207 --                      PinVRect(viewExtent, theMouse);
16 3208 --                  IF tracker.fConstrainsMouse THEN
16 3209 --                      tracker.TrackConstrain(anchorPoint, previousPoint, theMouse);
16 3210 -1                  END;
16 3211 -0 B        END;
16 3212 --
16 3213 --          {------------------------------+
16 3214 --          |   TrackOnce                  |
16 3215 --          +------------------------------}
16 3216 -- B        PROCEDURE TrackOnce(aTrackPhase: TrackPhase; didMouseMove: BOOLEAN);
16 3217 --          VAR
16 3218 --              newTracker: TCommand;
16 3219 0- B        BEGIN
16 3220 --      {$IFC qDebug}
16 3221 --              IF tracker = NIL THEN
16 3222 1-                  BEGIN
16 3223 --                  ProgramBreak('In TFrame.TrackInContent: tracker = NIL');
16 3224 --                  tracker := gNoChanges;
16 3225 --                  gotATracker := FALSE;
16 3226 -1                  END;
16 3227 --      {$ENDC}
16 3228 --
16 3229 --              IF gotATracker THEN
16 3230 1-                  BEGIN
16 3231 --                  newTracker := tracker.TrackMouse(aTrackPhase, anchorPoint,
16 3232 --                                                   previousPoint, theMouse, didMouseMove);
16 3233 --                  IF newTracker <> tracker THEN
16 3234 2-                      BEGIN
16 3235 --                      tracker.Free;
16 3236 --                      InstallTracker(newTracker);
16 3237 -2                      END
16 3238 --                  ELSE IF (newTracker <> gNoChanges) & (newTracker.fView <> view) THEN
16 3239 --                      InstallTracker(newTracker);
16 3240 -1                  END;
16 3241 -0 B         END;
16 3242 --
16 3243 0- A    BEGIN
16 3244 --          InstallTracker(theCommand);
16 3245 --
16 3246 --          SetupFocus;
16 3247 --
16 3248 --          theQDMouse := globalMouse;
16 3249 --          IF view <> NIL THEN
16 3250 1-              BEGIN
16 3251 --              GlobalToLocal(theQDMouse);
16 3252 --              view.QDToViewPt(theQDMouse, theMouse);
16 3253 -1              END
16 3254 --          ELSE
16 3255 --              PtToVPt(theQDMouse, theMouse);
16 3256 --          anchorPoint := theMouse;
16 3257 --          previousPoint := theMouse;
16 3258 --
16 3259 --          ConstrainOnce;
16 3260 --
16 3261 --          anchorPoint := theMouse;
16 3262 --          previousPoint := theMouse;      { in case Constrain changed the localPoint;
16 3263 --                                            guarantee that all 3 are the same on TrackPress }
16 3264 --
16 3265 --          TrackOnce(trackPress, TRUE);
16 3266 --          previousPoint := theMouse;      { in case TrackMouse changed nextPoint }
```

```
16 3267 --          FeedbackOnce(TRUE, TRUE);
16 3268 --
16 3269 --          movedOnce := FALSE;
16 3270 --
16 3271 --          WHILE StillDown DO
16 3272 1-              BEGIN
16 3273 --              DoFocus;
16 3274 --              GetMouse(theQDMouse);
16 3275 --              IF view <> NIL THEN
16 3276 --                  view.QDToViewPt(theQDMouse, theMouse)
16 3277 --              ELSE
16 3278 --                  PtToVPt(theQDMouse, theMouse);
16 3279 --
16 3280 --              IF NOT movedOnce THEN
16 3281 2-                  BEGIN
16 3282 --                  amtMoved := theMouse;
16 3283 --                  SubVPt(anchorPoint, amtMoved);
16 3284 --                  IF (Abs(amtMoved.h) >= hysteresis.h) OR
16 3285 --                     (Abs(amtMoved.v) >= hysteresis.v) THEN
16 3286 --                      movedOnce := TRUE;
16 3287 -2                 END;
16 3288 --
16 3289 --              IF (movedOnce | tracker.fTrackNonMovement) THEN
16 3290 2-                  BEGIN
16 3291 --                  SetVPt(delta, 0, 0);
16 3292 --
16 3293 --                  IF gotATracker THEN
16 3294 3-                      BEGIN
16 3295 --                          { ??? Problems with this:
16 3296 --                                  delta might be non-zero but scrolling can't take place
16 3297 --                                      because it is pinned at the end of the view
16 3298 --                                  also might want some slop before scrolling ??? }
16 3299 --                      IF scroller <> NIL THEN
16 3300 --                          IF NOT PtInVRect(theMouse, autoscrollLimit) THEN
16 3301 4-                              BEGIN
16 3302 --                              { !!! Need to convert theMouse to scroller coordinates }
16 3303 --                              scroller.AutoScroll(theMouse, delta);
16 3304 --                              AddVPt(delta, theMouse);
16 3305 -4                             END;
16 3306 --
16 3307 --                      ConstrainOnce;
16 3308 -3                     END;
16 3309 --
16 3310 --                  didScroll := (delta.h <> 0) OR (delta.v <> 0);
16 3311 --                  didMove := NOT EqualVPt(previousPoint, theMouse);
16 3312 --
16 3313 --                  FeedbackOnce(FALSE, didMove OR didScroll);
16 3314 --
16 3315 --                  { do the test here to avoid the method call, even though ScrollBy also checks }
16 3316 --                  IF didScroll THEN
16 3317 3-                      BEGIN
16 3318 --                      tracker.AutoScroll(delta.h, delta.v);
16 3319 --                      SetupFocus; { the focus changed }
16 3320 -3                     END;
16 3321 --
16 3322 --                  TrackOnce(trackMove, didMove); { ??? add OR didscroll ??? }
16 3323 --
16 3324 --                  previousPoint := theMouse;
16 3325 --                  FeedbackOnce(TRUE, didMove OR didScroll);
16 3326 -2                 END;
16 3327 -1              END;
16 3328 --
16 3329 --          DoFocus;
16 3330 --
16 3331 --          IF NOT movedOnce THEN
16 3332 --              theMouse := previousPoint { normally same as original mouse down; we don't use
16 3333 --                                  anchorPoint in case someone has changed that -- it is more
16 3334 --                                  likely that an app would change anchorPoint than previousPoint }
16 3335 --
16 3336 --          ELSE IF EventAvail(mUpMask, peekEvent) THEN
16 3337 1-              BEGIN
16 3338 --              theQDMouse := peekEvent.where;
16 3339 --              IF view <> NIL THEN
16 3340 2-                  BEGIN
16 3341 --                  GlobalToLocal(theQDMouse);
16 3342 --                  view.QDToViewPt(theQDMouse, theMouse);
16 3343 -2                 END
16 3344 --              ELSE
16 3345 --                  PtToVPt(theQDMouse, theMouse);
16 3346 --              ConstrainOnce;
16 3347 -1              END;
16 3348 --          { ELSE we use the last known mouse position }
16 3349 --
16 3350 --          FeedbackOnce(FALSE, TRUE);
16 3351 --          TrackOnce(trackRelease, TRUE);
16 3352 --
16 3353 --          TrackMouse := tracker;
16 3354 --
16 3355 -0 A    END { TApplication.TrackMouse };
16 3356 --
16 3357 --      {$S MARes}
16 3358 --      {-----------------------------------+
16 3359 --      |   UpdateAllWindows                |
16 3360 --      +-----------------------------------}
16 3361 -- A    PROCEDURE TApplication.UpdateAllWindows;
16 3362 --
16 3363 --      VAR
16 3364 --          anEvent:        EventRecord;
16 3365 --
```

```
16 3366 0- A   BEGIN
16 3367 --         WHILE GetEvent(updateMask, 0, NIL, anEvent) DO
16 3368 --             HandleEvent(anEvent);
16 3369 -0 A   END;
16 3370 --
16 3371 --
16 3372 --     {-----------------------------------+
16 3373 --     |   WMgrToWindow                    |
16 3374 --     +-----------------------------------}
16 3375 -- A   FUNCTION TApplication.WMgrToWindow (aWMgrWindow: WindowPtr): TWindow;
16 3376 --
16 3377 0- A   BEGIN
16 3378 --         IF (aWMgrWindow <> NIL) & (NOT IsDeskAccessory(aWMgrWindow))
16 3379 --     {$IFC qDebug}
16 3380 --            & (aWMgrWindow <> gDebugWindowPtr)
16 3381 --     {$ENDC}
16 3382 --            THEN
16 3383 --             WMgrToWindow := TWindow(GetWRefCon(aWMgrWindow))
16 3384 --         ELSE
16 3385 --             WMgrToWindow := NIL;
16 3386 -0 A   END;
13 3004 --     {$I UMacApp.TDocument.p}
```

```
17    1 --      {$P}
17    2 --      { UMacApp.TDocument.p }
17    3 --      { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
17    4 --
17    5 --
17    6 --      {$S MAOpen}
17    7 --      {-----------------------------------+
17    8 --      |   IDocument                       |
17    9 --      +-----------------------------------}
17   10 -- A    PROCEDURE TDocument.IDocument(itsFileType, itsCreator: OSType;
17   11 --                                    usesDataFork, usesRsrcFork: BOOLEAN;
17   12 --                                    keepsDataOpen, keepsRsrcOpen: BOOLEAN);
17   13 --      VAR
17   14 --          fi: FailInfo;
17   15 --
17   16 --          {-------------------------------+
17   17 --          |   HdlIDocument                |
17   18 --          +-------------------------------}
17   19 -- B      PROCEDURE HdlIDocument(error: OSErr; message: LONGINT);
17   20 0- B      BEGIN
17   21 --            Free;
17   22 -0 B      END;
17   23 --
17   24 0- A    BEGIN
17   25 --          fWindowList := NIL;
17   26 --          fViewList := NIL;
17   27 --          fPrintInfo := NIL;
17   28 --          fDocPrintHandler := NIL;
17   29 --
17   30 --          fDataRefnum := kNoFileRefnum;
17   31 --          fRsrcRefnum := kNoFileRefnum;
17   32 --
17   33 --          fTitle := '';
17   34 --          fPrintInfo := NIL;
17   35 --          fSavePrintInfo := FALSE;
17   36 --          fSharePrintInfo := TRUE;
17   37 --
17   38 --          fVolRefNum := 0;
17   39 --          fReopenAlert := TRUE;
17   40 --          fSaveExists := FALSE;
17   41 --          fCommitOnSave := TRUE;
17   42 --          fModDate := 0;
17   43 --
17   44 --          fFileType := itsFileType;
17   45 --          fCreator := itsCreator;
17   46 --
17   47 --          fUsesDataFork := usesDataFork;
17   48 --          fUsesRsrcFork := usesRsrcFork;
17   49 --          fDataOpen := keepsDataOpen;
17   50 --          fRsrcOpen := keepsRsrcOpen;
17   51 --
17   52 --      {$IFC qDebug}
17   53 --          IF fDataOpen THEN
17   54 --              IF NOT fUsesDataFork THEN
17   55 1-                  BEGIN
17   56 --                  Writeln('In TDocument.IDocument:  fDataOpen AND NOT fUsesDataFork;');
17   57 --                  Writeln('In TDocument.IDocument:  fUsesDataFork := TRUE;');
17   58 --                  fUsesDataFork := TRUE;
17   59 -1                  END;
17   60 --          IF fRsrcOpen THEN
17   61 --              IF NOT fUsesRsrcFork THEN
17   62 1-                  BEGIN
17   63 --                  Writeln('In TDocument.IDocument:  fRsrcOpen AND NOT fUsesRsrcFork;');
17   64 --                  Writeln('In TDocument.IDocument:  fUsesRsrcFork := TRUE;');
17   65 --                  fUsesRsrcFork := TRUE;
17   66 -1                  END;
17   67 --      {$ENDC}
17   68 --
17   69 --          IF keepsDataOpen OR keepsRsrcOpen THEN
17   70 --              fSaveInPlace := sipNever
17   71 --          ELSE
17   72 --              fSaveInPlace := sipAskUser;
17   73 --
17   74 --          fDataPerm := fsRdPerm;
17   75 --          fRsrcPerm := fsRdPerm;
17   76 --
17   77 --          IEvtHandler(gApplication);
17   78 --
17   79 --          CatchFailures(fi, HdlIDocument);
17   80 --
17   81 --          fChangeCount := 0;
17   82 --
17   83 --          fWindowList := NewList;
17   84 --      {$IFC qDebug}
17   85 --          fWindowList.SetEltType('TWindow');
17   86 --      {$ENDC}
17   87 --
17   88 --          fViewList := NewList;
17   89 --      {$IFC qDebug}
17   90 --          fViewList.SetEltType('TView');
17   91 --      {$ENDC}
17   92 --
17   93 --          Success(fi);
17   94 -0 A    END;
17   95 --
17   96 --
17   97 --      {$S MAClose}
17   98 --      {-----------------------------------+
```

```
17   99 --      |   Free                              |
17  100 --      +-------------------------------------}
17  101 -- A    PROCEDURE TDocument.Free; OVERRIDE;
17  102 --
17  103 0- A    BEGIN
17  104 --          gApplication.DeleteDocument(SELF);
17  105 --
17  106 --          FreeFile;
17  107 --
17  108 --          IF fWindowList <> NIL THEN
17  109 --              fWindowList.FreeList;
17  110 --
17  111 --          IF fViewList <> NIL THEN
17  112 --              fViewList.FreeList;
17  113 --
17  114 --          IF fSharePrintInfo AND (fPrintInfo <> NIL)  THEN
17  115 --              DisposHandle(fPrintInfo);
17  116 --
17  117 --          INHERITED Free;
17  118 -0 A    END;
17  119 --
17  120 --
17  121 --      {$S MAOpen}
17  122 --      {-------------------------------------+
17  123 --      |   AddView                           |
17  124 --      +-------------------------------------}
17  125 -- A    PROCEDURE TDocument.AddView(aView: TView);
17  126 0- A    BEGIN
17  127 --          fViewList.InsertLast(aView);
17  128 -0 A    END;
17  129 --
17  130 --
17  131 --      {$S MAWriteFile}
17  132 --      {-------------------------------------+
17  133 --      |   AboutToSave                       |
17  134 --      +-------------------------------------}
17  135 -- A    PROCEDURE TDocument.AboutToSave(itsCmd: CmdNumber;
17  136 --                                      VAR newName: Str255; VAR newVolRefnum: INTEGER;
17  137 --                                      VAR makingCopy: BOOLEAN);
17  138 0- A    BEGIN
17  139 -0 A    END;
17  140 --
17  141 --
17  142 --      {$S MAOpen}
17  143 --      {-------------------------------------+
17  144 --      |   AddWindow                         |
17  145 --      +-------------------------------------}
17  146 -- A    PROCEDURE TDocument.AddWindow(aWindow: TWindow);
17  147 0- A    BEGIN
17  148 --          fWindowList.InsertLast(aWindow);
17  149 -0 A    END;
17  150 --
17  151 --
17  152 --      {$S MAFile}
17  153 --      {-------------------------------------+
17  154 --      |   CheckDiskFile                     |
17  155 --      +-------------------------------------}
17  156 -- A    PROCEDURE TDocument.CheckDiskFile(rsrcId, rsrcIndex: INTEGER; reverting: BOOLEAN);
17  157 --          VAR err:    OSErr;
17  158 --              name:   Str255;
17  159 --              s:      Str255;
17  160 0- A    BEGIN
17  161 --          err := DiskFileChanged(reverting); {don't care about the file type if
17  162 --                                                 saving}
17  163 --          IF err = errFileChanged THEN
17  164 1-              BEGIN
17  165 --              name := fTitle;
17  166 --              GetIndString(s, rsrcId, rsrcIndex);
17  167 --              ParamText(name, s, '', '');
17  168 --              IF MacAppAlert(phFileChanged, NIL) = cancel THEN
17  169 --                  Failure(noErr, msgCancelled);
17  170 -1              END
17  171 --                  {if reverting then we signal an error}
17  172 --          ELSE IF (err <> noErr) AND reverting THEN
17  173 --              Failure(err, 0);
17  174 -0 A    END;
17  175 --
17  176 --
17  177 --      {$S MAClose}
17  178 --      {-------------------------------------+
17  179 --      |   Close                             |
17  180 --      +-------------------------------------}
17  181 -- A    PROCEDURE TDocument.Close;
17  182 --          {Must never be called for a document related to a view in the Clipboard.
17  183 --              Why is this???}
17  184 --
17  185 --          {-------------------------------+
17  186 --          |   CloseAWindow                |
17  187 --          +-------------------------------}
17  188 -- B        PROCEDURE CloseAWindow(aWindow: TWindow);
17  189 0- B        BEGIN
17  190 --              aWindow.Close
17  191 -0 B        END;
17  192 --
17  193 0- A    BEGIN
17  194 --      {$IFC qDebug}
17  195 --          IF gClipWindow.fDocument = SELF THEN
17  196 --              ProgramBreak('TDocument.Close:  Attempt to close clipboard document');
17  197 --      {$ENDC}
```

```
17  198 --
17  199 --          IF fChangeCount <> 0 THEN
17  200 1-              CASE PoseSaveDialog OF
17  201 --                  cancel:
17  202 --                      Failure(noErr, msgCancelled);
17  203 --                  kYesButton:
17  204 --                      {Will fail if unable to save}
17  205 --                      Save(cClose, {askForFilename:} NOT fSaveExists, kSwitchToTarget);
17  206 -1                 END;
17  207 --
17  208 --          IF gLastCommand <> NIL THEN
17  209 --              IF gLastCommand.fChangedDocument = SELF THEN
17  210 --                  gApplication.CommitLastCommand;
17  211 --
17  212 --          ForAllWindowsDo(CloseAWindow);
17  213 --
17  214 --          Free;
17  215 -0 A    END;
17  216 --
17  217 --
17  218 --      {$S MAClose}
17  219 --      {-----------------------------------+
17  220 --      |   DeleteView                      |
17  221 --      +-----------------------------------}
17  222 -- A    PROCEDURE TDocument.DeleteView(viewToDelete: TView);
17  223 0- A    BEGIN
17  224 --          fViewList.Delete(viewToDelete);
17  225 -0 A    END;
17  226 --
17  227 --
17  228 --      {$S MAClose}
17  229 --      {-----------------------------------+
17  230 --      |   DeleteWindow                    |
17  231 --      +-----------------------------------}
17  232 -- A    PROCEDURE TDocument.DeleteWindow(windowToDelete: TWindow);
17  233 0- A    BEGIN
17  234 --          fWindowList.Delete(windowToDelete);
17  235 -0 A    END;
17  236 --
17  237 --
17  238 --      {$S MAFile}
17  239 --      {-----------------------------------+
17  240 --      |   DiskFileChanged                 |
17  241 --      +-----------------------------------}
17  242 -- A    FUNCTION  TDocument.DiskFileChanged(checkType: BOOLEAN): OSErr;
17  243 --          VAR pb:     HParamBlockRec;
17  244 --              err:    OSErr;
17  245 0- A    BEGIN
17  246 --          IF fSaveExists THEN
17  247 1-              BEGIN
17  248 --              err := GetFileInfo(fTitle, fVolRefnum, pb);
17  249 --              IF err = noErr THEN
17  250 --                  IF checkType AND (pb.ioFlFndrInfo.fdType <> fFileType) THEN
17  251 --                      err := errFTypeChanged
17  252 --                  ELSE IF pb.ioFlMdDat <> fModDate THEN
17  253 --                      err := errFileChanged;
17  254 --              DiskFileChanged := err;
17  255 -1              END
17  256 --          ELSE
17  257 --              DiskFileChanged := noErr;
17  258 -0 A    END;
17  259 --
17  260 --
17  261 --      {$S MAOpen}
17  262 --      {-----------------------------------+
17  263 --      |   DoInitialState                  |
17  264 --      +-----------------------------------}
17  265 -- A    PROCEDURE TDocument.DoInitialState;
17  266 --              {Called for 'New' & 'Revert' [to blank] commands & for default open tool icon}
17  267 0- A    BEGIN
17  268 -0 A    END;
17  269 --
17  270 --
17  271 --      {$S MANever}
17  272 --      {-----------------------------------+
17  273 --      |   DoMakeViews                     |
17  274 --      +-----------------------------------}
17  275 -- A    PROCEDURE TDocument.DoMakeViews(forPrinting: BOOLEAN);
17  276 --      (*  VAR aYOURView: TYOURView; *)
17  277 0- A    BEGIN
17  278 --          (*
17  279 --          NEW(aYOURView);
17  280 --          aYOURView.IYOURView(SELF, YOURExtentRect);
17  281 --          DoMakeView := aYOURView;
17  282 --          *)
17  283 -0 A    END;
17  284 --
17  285 --
17  286 --      {$S MARes}
17  287 --      {-----------------------------------+
17  288 --      |   DoMakeWindows                   |
17  289 --      +-----------------------------------}
17  290 -- A    PROCEDURE TDocument.DoMakeWindows;
17  291 0- A    BEGIN
17  292 -0 A    END;
17  293 --
17  294 --
17  295 --      {$S MASelCommand}
17  296 --      {-----------------------------------+
```

```
17  297 --      |   DoMenuCommand                   |
17  298 --      +-----------------------------------}
17  299 -- A    FUNCTION  TDocument.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
17  300 --          VAR name:           Str255;
17  301 --              info:           HParamBlockRec;
17  302 --              err:            OSErr;
17  303 --              fi:             FailInfo;
17  304 --
17  305 --          {-------------------------------+
17  306 --          |   HdlRevertCmd                |
17  307 --          +-------------------------------}
17  308 -- B        PROCEDURE HdlRevertCmd(error: OSErr; message: LONGINT);
17  309 0- B       BEGIN
17  310 --              ShowReverted; {make sure screen is updated}
17  311 -0 B       END;
17  312 --
17  313 0- A    BEGIN
17  314 --          DoMenuCommand := gNoChanges;
17  315 --
17  316 1-          CASE aCmdNumber OF
17  317 --
17  318 --              cPrFileBase..cPrFileMax:
17  319 --                  IF fDocPrintHandler <> NIL THEN
17  320 --                      DoMenuCommand := fDocPrintHandler.DoMenuCommand(aCmdNumber);
17  321 --
17  322 --              cSave, cSaveAs, cSaveCopy:
17  323 2-                  BEGIN
17  324 --                  Save(aCmdNumber,
17  325 --                          {askForFilename:}   NOT fSaveExists OR (aCmdNumber <> cSave),
17  326 --                          {makingCopy:}       aCmdNumber = cSaveCopy);
17  327 -2                  END;
17  328 --
17  329 --              cRevert:
17  330 2-                  BEGIN
17  331 --                  name := fTitle;
17  332 --                  ParamText(name, '', '', '');
17  333 --                  IF MacAppAlert(phRevert, NIL) = kYesButton THEN
17  334 3-                      BEGIN
17  335 --                      CatchFailures(fi, HdlRevertCmd);
17  336 --                      Revert;
17  337 --                      Success(fi);
17  338 --                      ShowReverted;
17  339 -3                      END;
17  340 -2                  END;
17  341 --
17  342 --              OTHERWISE
17  343 --                  DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
17  344 -1              END; {CASE}
17  345 -0 A    END;
17  346 --
17  347 --
17  348 --      {$S MAWriteFile}
17  349 --      {-----------------------------------+
17  350 --      |   DoNeedDiskSpace                 |
17  351 --      +-----------------------------------}
17  352 -- A    PROCEDURE TDocument.DoNeedDiskSpace(VAR dataForkBytes, rsrcForkBytes: LONGINT);
17  353 0- A    BEGIN
17  354 --          IF fSavePrintInfo THEN
17  355 --              dataForkBytes := dataForkBytes + kPrintInfoSize;
17  356 --          IF fUsesRsrcFork THEN
17  357 --              rsrcForkBytes := rsrcForkBytes + kRsrcFileOverhead;
17  358 -0 A    END;
17  359 --
17  360 --
17  361 --      {$S MAReadFile}
17  362 --      {-----------------------------------+
17  363 --      |   DoRead                          |
17  364 --      +-----------------------------------}
17  365 -- A    PROCEDURE TDocument.DoRead(aRefNum: INTEGER;
17  366 --                                          rsrcExists, forPrinting: BOOLEAN);
17  367 --          VAR count:  LONGINT;
17  368 --
17  369 0- A    BEGIN
17  370 --          IF fSavePrintInfo THEN
17  371 1-              BEGIN
17  372 --              IF fPrintInfo = NIL THEN
17  373 2-                  BEGIN
17  374 --                  fPrintInfo := NewPermHandle(kPrintInfoSize);
17  375 --                  FailNIL(fPrintInfo);
17  376 -2                  END;
17  377 --
17  378 --              count := kPrintInfoSize;
17  379 --              FailOSErr(FSRead(aRefNum, count, fPrintInfo^));
17  380 -1              END;
17  381 -0 A    END;
17  382 --
17  383 --
17  384 --      {$S MARes}
17  385 --      {-----------------------------------+
17  386 --      |   DoSetupMenus                    |
17  387 --      +-----------------------------------}
17  388 -- A    PROCEDURE TDocument.DoSetupMenus;
17  389 0- A    BEGIN
17  390 --          INHERITED DoSetupMenus;
17  391 --
17  392 --          Enable(cSaveAs,TRUE);
17  393 --          Enable(cSaveCopy,TRUE);
17  394 --          IF fChangeCount <> 0 THEN
17  395 1-              BEGIN
```

```
17  396 --              Enable(cSave, TRUE);
17  397 --              Enable(cRevert, TRUE);
17  398 -1              END;
17  399 --
17  400 --          IF fDocPrintHandler <> NIL THEN
17  401 --              IF NOT gTarget.HandlesPrintingCommands THEN
17  402 --                  fDocPrintHandler.DoSetUpMenus;
17  403 -0 A     END;
17  404 --
17  405 --
17  406 --      {$S MAWriteFile}
17  407 --      {-----------------------------------+
17  408 --      |   DoWrite                         |
17  409 --      +-----------------------------------}
17  410 -- A    PROCEDURE TDocument.DoWrite(aRefNum: INTEGER; makingCopy: BOOLEAN);
17  411 --          VAR count:  LONGINT;
17  412 0- A    BEGIN
17  413 --          IF fSavePrintInfo THEN
17  414 1-              BEGIN
17  415 --              IF fPrintInfo = NIL THEN
17  416 2-                  BEGIN
17  417 --      {$IFC qDebug}
17  418 --                  ProgramBreak('no print record in document')  {???}
17  419 --      {$ENDC}
17  420 -2                  END
17  421 --              ELSE
17  422 2-                  BEGIN
17  423 --                  count := kPrintInfoSize;
17  424 --                  FailOSErr(FSWrite(aRefNum, count, fPrintInfo^));
17  425 -2                  END;
17  426 -1              END;
17  427 -0 A     END;
17  428 --
17  429 --
17  430 --          {This is specified to work only on views currently in frames.
17  431 --              If you want all views use fViewList.Each.}
17  432 --      {$S MARes}
17  433 --      {-----------------------------------+
17  434 --      |   ForAllViewsDo                   |
17  435 --      +-----------------------------------}
17  436 -- A    PROCEDURE TDocument.ForAllViewsDo(PROCEDURE DoToView(aView: TView));
17  437 --
17  438 0- A    BEGIN
17  439 --          IF gFinderPrinting THEN
17  440 1-              BEGIN
17  441 --              IF fDocPrintHandler <> NIL THEN
17  442 --                  IF fDocPrintHandler.fView <> NIL THEN
17  443 --                      DoToView(fDocPrintHandler.fView);
17  444 -1              END
17  445 --          ELSE
17  446 --              fViewList.Each(DoToView);
17  447 -0 A     END;
17  448 --
17  449 --
17  450 --      {$S MARes}
17  451 --      {-----------------------------------+
17  452 --      |   ForAllWindowsDo                 |
17  453 --      +-----------------------------------}
17  454 -- A    PROCEDURE TDocument.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
17  455 0- A    BEGIN
17  456 --          fWindowList.Each(DoToWind);
17  457 -0 A     END;
17  458 --
17  459 --
17  460 --      {$S MARes}
17  461 --      {-----------------------------------+
17  462 --      |   FreeData                        |
17  463 --      +-----------------------------------}
17  464 -- A    PROCEDURE TDocument.FreeData;
17  465 0- A    BEGIN
17  466 -0 A     END;
17  467 --
17  468 --
17  469 --      {$S MAWriteFile}
17  470 --      {-----------------------------------+
17  471 --      |   FreeFile                        |
17  472 --      +-----------------------------------}
17  473 -- A    PROCEDURE TDocument.FreeFile;
17  474 --          VAR err:    OSErr;
17  475 0- A    BEGIN
17  476 --          IF fDataOpen OR fRsrcOpen THEN
17  477 1-              BEGIN
17  478 --              err := CloseFile(fDataRefnum, fRsrcRefnum);
17  479 --              {$IFC qDebug}
17  480 --              IF err <> noErr THEN
17  481 --                  Writeln('In TDocument.FreeFile: error from CloseFile = ', err:1);
17  482 --              {$ENDC}
17  483 -1              END;
17  484 -0 A     END;
17  485 --
17  486 --
17  487 --      {$S MAClipboard}
17  488 --      {-----------------------------------+
17  489 --      |   FreeFromClipboard               |
17  490 --      +-----------------------------------}
17  491 -- A    PROCEDURE TDocument.FreeFromClipboard;
17  492 0- A    BEGIN
17  493 --          fWindowList.Delete(gClipWindow);
17  494 --          Free;
```

```
17  495 -0 A   END;
17  496 --
17  497 --
17  498 --     {$IFC qInspector}
17  499 --     {$S MAInspector}
17  500 --     {-----------------------------------+
17  501 --     |   GetInspectorName                |
17  502 --     +-----------------------------------}
17  503 -- A   PROCEDURE TDocument.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
17  504 --
17  505 0- A   BEGIN
17  506 --         inspectorName := fTitle;
17  507 -0 A   END;
17  508 --     {$ENDC}
17  509 --
17  510 --
17  511 --     {$S MAWriteFile}
17  512 --     {-----------------------------------+
17  513 --     |   GetSaveInfo                     |
17  514 --     +-----------------------------------}
17  515 -- A   FUNCTION  TDocument.GetSaveInfo(itsCmdNumber: CmdNumber;
17  516 --                                        copyFInfo: BOOLEAN;
17  517 --                                        VAR cInfo: CInfoPBRec): BOOLEAN;
17  518 --         VAR currName:   Str255;
17  519 --             err:        OSErr;
17  520 0- A   BEGIN
17  521 --         IF fSaveExists AND copyFInfo THEN
17  522 1-             BEGIN
17  523 --             currName := fTitle;
17  524 --             WITH cInfo DO
17  525 2-                 BEGIN
17  526 --                 ioNamePtr := @currName;
17  527 --                 ioVRefnum := fVolRefnum;
17  528 --                 ioFVersNum := 0;
17  529 --                 ioFDirIndex := 0;
17  530 --                 ioDirID := 0;
17  531 -2                 END;
17  532 --             IF gConfiguration.hasHFS THEN
17  533 2-                 BEGIN
17  534 --                 err := FillInDirID(@cInfo);
17  535 --                 IF err = noErr THEN
17  536 --                     err := PBGetCatInfo(@cInfo, FALSE);
17  537 -2                 END
17  538 --             ELSE
17  539 --                 err := PBGetFInfo(@cInfo, FALSE);
17  540 --
17  541 --             cInfo.ioNamePtr := NIL; {since ptr is invalid when we exit}
17  542 --
17  543 --                 {set the type and creator in case it has changed;
17  544 --                     the file might be on a file server and someone else
17  545 --                     could have changed the document}
17  546 --             cInfo.ioFlFndrInfo.fdCreator := fCreator;
17  547 --             cInfo.ioFlFndrInfo.fdType := fFileType;
17  548 -1             END
17  549 --         ELSE
17  550 --             err := fnfErr; {fake error}
17  551 --
17  552 --         IF err = noErr THEN
17  553 --             GetSaveInfo := TRUE
17  554 --         ELSE
17  555 1-             BEGIN
17  556 --             WITH cInfo.ioFlFndrInfo DO
17  557 2-                 BEGIN
17  558 --                 fdCreator := fCreator;
17  559 --                 fdType := fFileType;
17  560 -2                 END;
17  561 --
17  562 --             GetSaveInfo := FALSE
17  563 -1             END;
17  564 -0 A   END;
17  565 --
17  566 --
17  567 --     {$S MAFile}
17  568 --     {-----------------------------------+
17  569 --     |   GetTempName                     |
17  570 --     +-----------------------------------}
17  571 -- A   PROCEDURE TDocument.GetTempName(VAR filename: Str255);
17  572 --
17  573 --     CONST
17  574 --         maxName =       31;         {maximum name size to generate}
17  575 --         maxNumber =     10;         {maximum # digits of the random number}
17  576 --         maxPrefix =     maxName - maxNumber;
17  577 --
17  578 --     VAR
17  579 --         s:          Str255;
17  580 --         apRefnum:   INTEGER;
17  581 --         apParam:    Handle;
17  582 --         time:       LONGINT;
17  583 --
17  584 0- A   BEGIN
17  585 --         {If the document is untitled, use the application name.}
17  586 --         IF fTitle = '' THEN
17  587 --             GetAppParms(filename, apRefnum, apParam)
17  588 --         ELSE
17  589 --             filename := fTitle;
17  590 --
17  591 --         IF Length(filename) > maxPrefix THEN
17  592 --             filename := Copy(filename, 1, maxPrefix);
17  593 --
```

```
17  594 --          {append a pseudo-random number}
17  595 --          GetDateTime(time);
17  596 --          NumToString(ABS(BXOR(time, BRotR(TickCount, 16))), s);
17  597 --          filename := CONCAT(filename, s);
17  598 -0 A     END;
17  599 --
17  600 --
17  601 --      {$S MARes}
17  602 --      {-----------------------------------+
17  603 --      |   HandlesPrintingCommands         |
17  604 --      +-----------------------------------}
17  605 -- A    FUNCTION  TDocument.HandlesPrintingCommands: BOOLEAN; OVERRIDE;
17  606 0- A    BEGIN
17  607 --          HandlesPrintingCommands := FALSE;
17  608 -0 A    END;
17  609 --
17  610 --
17  611 --      {$S MAWriteFile}
17  612 --      {-----------------------------------+
17  613 --      |   MakeNewCopy                     |
17  614 --      +-----------------------------------}
17  615 -- A    PROCEDURE TDocument.MakeNewCopy(makingCopy: BOOLEAN;
17  616 --                                      validFInfo: BOOLEAN;
17  617 --                                                  VAR cInfo: CInfoPBRec);
17  618 --           {This routine changes the following fields of cInfo:
17  619 --
17  620 --            The ioNamePtr, and ioVRefnum fields must be set to indicate
17  621 --              the desired file; this routine sets ioDirID to 0.
17  622 --
17  623 --           IF setFInfo is TRUE then all the fields of cInfo
17  624 --              should be set up.  Otherwise only the file type and creator
17  625 --              need to be set up.}
17  626 --
17  627 --          VAR err:        OSErr;
17  628 --              dataRefnum: INTEGER;
17  629 --              rsrcRefnum: INTEGER;
17  630 --              oldVRefnum: INTEGER;
17  631 --              fi:         FailInfo;
17  632 --
17  633 --          {-------------------------------+
17  634 --          |   HdlMkNewCopy                |
17  635 --          +-------------------------------}
17  636 -- B        PROCEDURE HdlMkNewCopy(error: OSErr; message: LONGINT);
17  637 --              VAR err:    OSErr;
17  638 0- B        BEGIN
17  639 --              err := CloseFile(dataRefnum, rsrcRefnum);
17  640 --              {$IFC qDebug}
17  641 --              IF err <> noErr THEN
17  642 --                  Writeln('In HdlMkNewCopy: error from CloseFile is ', err: 1);
17  643 --              {$ENDC}
17  644 --
17  645 --              err := DeleteFile(cInfo.ioNamePtr, cInfo.ioVRefnum);
17  646 --              {$IFC qDebug}
17  647 --              IF err <> noErr THEN
17  648 --                  IF err <> fnfErr THEN
17  649 --                      Writeln('In HdlMkNewCopy: error from DeleteFile is ', err: 1);
17  650 --              {$ENDC}
17  651 -0 B        END;
17  652 --
17  653 0- A    BEGIN
17  654 --          IF fUsesDataFork OR fUsesRsrcFork THEN
17  655 1-              BEGIN
17  656 --              cInfo.ioDirID := 0;
17  657 --
17  658 --                  {Initalize these in case we fail before the call to OpenAFile.}
17  659 --              dataRefnum := kNoFileRefnum;
17  660 --              rsrcRefnum := kNoFileRefnum;
17  661 --
17  662 --                  {Create the file with the desired creator/type,
17  663 --                      in case we are not going to set the file info
17  664 --                      below.}
17  665 --              FailOSErr(Create(cInfo.ioNamePtr^, cInfo.ioVRefnum,
17  666 --                               cInfo.ioFlFndrInfo.fdCreator,
17  667 --                               cInfo.ioFlFndrInfo.fdType));
17  668 --
17  669 --              CatchFailures(fi, HdlMkNewCopy);
17  670 --
17  671 --              IF fUsesRsrcFork THEN
17  672 2-                  BEGIN
17  673 --                  FailOSErr(GetVol(NIL, oldVRefnum));
17  674 --                  FailOSErr(SetVol(NIL, cInfo.ioVRefnum));
17  675 --
17  676 --                  CreateResFile(cInfo.ioNamePtr^);
17  677 --
17  678 --                      {Do this here to ensure that the current volume is
17  679 --                          reset, even if we fail.}
17  680 --                  FailOSErr(SetVol(NIL, oldVRefnum));
17  681 --
17  682 --                  FailResError; {this checks the call to CreateResFile}
17  683 -2                  END;
17  684 --
17  685 --              IF validFInfo THEN
17  686 --                  IF NOT makingCopy THEN
17  687 2-                      BEGIN
17  688 --                      IF gConfiguration.hasHFS THEN
17  689 --                          err := PBSetCatInfo(@cInfo, FALSE) {??? PMSP ???}
17  690 --                      ELSE
17  691 3-                          BEGIN
17  692 --                              {NOTE: We can use the cInfo for this call since the
```

```
17  693 --                                fields required by PBSetInfo are a subset of
17  694 --                                those in the CInfoPBRec; except we set the
17  695 --                                ioFVersNum field (just in case)}
17  696 --                          cInfo.ioFVersNum := 0;
17  697 --                          err := PBSetFInfo(@cInfo, FALSE);
17  698 -3                          END;
17  699 --                      FailOSErr(err);
17  700 -2                      END;
17  701 --
17  702 --              FailOSErr(OpenAFile(cInfo.ioNamePtr^,   cInfo.ioVRefnum,
17  703 --                                  fUsesDataFork,      fUsesRsrcFork,
17  704 --                                  fsRdWrPerm,         fsRdWrPerm,
17  705 --                                  dataRefnum,         rsrcRefnum));
17  706 --
17  707 --              DoWrite(dataRefnum, makingCopy);
17  708 --
17  709 --              Success(fi);
17  710 --
17  711 --              FailOSErr(CloseFile(dataRefnum, rsrcRefnum));
17  712 -1              END;
17  713 -0 A    END;
17  714 --
17  715 --
17  716 --      {$S MAFile}
17  717 --      {-----------------------------------+
17  718 --      |   OpenAFile                       |
17  719 --      +-----------------------------------}
17  720 -- A    FUNCTION TDocument.OpenAFile(name: Str255; volRefnum: INTEGER;
17  721 --              openData, openRsrc: BOOLEAN; dataPerm, rsrcPerm: INTEGER;
17  722 --              VAR dataRefnum, rsrcRefnum: INTEGER): OSErr;
17  723 --
17  724 0- A    BEGIN
17  725 --          OpenAFile := OpenFile(name, volRefnum, openData, openRsrc, dataPerm, rsrcPerm,
17  726 --                                dataRefnum, rsrcRefnum);
17  727 -0 A    END;
17  728 --
17  729 --
17  730 --      {$S MAOpen}
17  731 --      {-----------------------------------+
17  732 --      |   OpenAgain                       |
17  733 --      +-----------------------------------}
17  734 -- A    PROCEDURE TDocument.OpenAgain(itsCmdNumber: CmdNumber; openingDoc: TDocument);
17  735 --          VAR window: TWindow;
17  736 --              s:      Str255;
17  737 0- A    BEGIN
17  738 --          s := fTitle;                { because ParamText allocates memory }
17  739 --          ParamText(s, '', '', '');
17  740 --
17  741 --          IF fReopenAlert THEN
17  742 --              StdAlert(phReopenDoc);
17  743 --
17  744 --          window := TWindow(fWindowList.First);
17  745 --          window.Select;
17  746 --          Failure(0, 0);
17  747 -0 A    END;
17  748 --
17  749 --
17  750 --      {$S MAClose}
17  751 --      {-----------------------------------+
17  752 --      |   PoseSaveDialog                  |
17  753 --      +-----------------------------------}
17  754 -- A    FUNCTION  TDocument.PoseSaveDialog: INTEGER;
17  755 --
17  756 --      VAR
17  757 --          idx:        INTEGER;
17  758 --          name:       Str255;
17  759 --          reason:     Str255;
17  760 --
17  761 0- A    BEGIN
17  762 --          IF fChangeCount <> 0 THEN
17  763 1-              BEGIN
17  764 --              IF gAppDone THEN
17  765 --                  idx := bzQuitting
17  766 --              ELSE
17  767 --                  idx := bzClosing;
17  768 --
17  769 --              GetIndString(reason, kIDBuzzString, idx);
17  770 --              name := fTitle;         { ParamText can compact heap }
17  771 --              ParamText(name, reason, '', '');
17  772 --
17  773 --              PoseSaveDialog := MacAppAlert(phSaveChanges, NIL);
17  774 -1              END
17  775 --          ELSE
17  776 --              PoseSaveDialog := kNoButton;
17  777 -0 A    END {TDocument.PoseSaveDialog};
17  778 --
17  779 --      {$IFC FALSE}
17  780 --          {$S MAFinder}
17  781 --          {-------------------------------+
17  782 --          |   Print                       |
17  783 --          +-------------------------------}
17  784 -- A        FUNCTION  TDocument.Print(itsCmdNumber: CmdNumber): BOOLEAN;
17  785 --                  {??? Only relevant for print from Finder at the present, so aCommand will always be
17  786 --                          gNoChanges, and hence need not have anything done to it.  May change ???}
17  787 --              VAR proceed:    BOOLEAN;
17  788 --                  aCommand:   TCommand;
17  789 0- A        BEGIN
17  790 --              aCommand := fDocPrintHandler.Print(itsCmdNumber, proceed);
17  791 --              Print := proceed;
```

```
17  792 -0 A       END;
17  793 --      {$ENDC}
17  794 --
17  795 --      {$S MAReadFile}
17  796 --      {-----------------------------------+
17  797 --      |   ReadFromFile                    |
17  798 --      +-----------------------------------}
17  799 -- A   PROCEDURE TDocument.ReadFromFile(VAR anAppFile: AppFile; forPrinting: BOOLEAN);
17  800 --          VAR fi:             FailInfo;
17  801 --              dataRefnum:     INTEGER;
17  802 --              rsrcRefnum:     INTEGER;
17  803 --              isRevert:       BOOLEAN;
17  804 --              shouldOpenData: BOOLEAN;
17  805 --              shouldOpenRsrc: BOOLEAN;
17  806 --
17  807 --          {-------------------------------+
17  808 --          |   HdlRead                     |
17  809 --          +-------------------------------}
17  810 -- B       PROCEDURE HdlRead(error: INTEGER; message: LONGINT);
17  811 --              VAR err:    INTEGER;
17  812 0- B       BEGIN
17  813 --              err := CloseFile(dataRefnum, rsrcRefnum);
17  814 --              {$IFC qDebug}
17  815 --              IF err <> noErr THEN
17  816 --                  Writeln('In HdlOpen: error from CloseFile is ', err:1);
17  817 --              {$ENDC}
17  818 --
17  819 --              {caller should set the message as appropriate}
17  820 -0 B       END;
17  821 --
17  822 0- A   BEGIN
17  823 --          isRevert := (anAppFile.fName = ''); {signal to re-read file}
17  824 --          IF isRevert THEN
17  825 1-              BEGIN
17  826 --              anAppFile.fName := fTitle;
17  827 --              anAppFile.vRefnum := fVolRefnum;
17  828 -1              END
17  829 --          ELSE
17  830 1-              BEGIN
17  831 --              fTitle := anAppFile.fName;
17  832 --              fVolRefnum := anAppFile.vRefnum;
17  833 -1              END;
17  834 --
17  835 --          {Don't attempt to open the fork(s) again if they're already open.
17  836 --              We do all this rather than close and reopen so that we need not
17  837 --              search the directory again, an expensive operation on large MFS
17  838 --              disks.}
17  839 --          shouldOpenData := fUsesDataFork AND NOT (fDataOpen AND isRevert);
17  840 --          shouldOpenRsrc := fUsesRsrcFork AND NOT (fRsrcOpen AND isRevert);
17  841 --
17  842 --          {Make sure CloseFile operates properly if OpenAFile fails.}
17  843 --          dataRefnum := kNoFileRefnum;
17  844 --          rsrcRefnum := kNoFileRefnum;
17  845 --
17  846 --          CatchFailures(fi, HdlRead);
17  847 --
17  848 --          FailOSErr(OpenAFile(anAppFile.fName, anAppFile.vRefnum,
17  849 --                              shouldOpenData,  shouldOpenRsrc,
17  850 --                              fDataPerm,       fRsrcPerm,
17  851 --                              dataRefnum,      rsrcRefnum));
17  852 --
17  853 --          fSaveExists := TRUE;
17  854 --
17  855 --          {If the file is already open, use the refnums we already have.
17  856 --              Make sure that the document's data fork is positioned at TOF.
17  857 --              Make sure that the document's resource file is on top.}
17  858 --          IF fDataOpen AND NOT shouldOpenData THEN
17  859 1-              BEGIN
17  860 --              dataRefNum := fDataRefNum;
17  861 --              FailOSErr(SetFPos(dataRefNum, fsFromStart, 0));
17  862 -1              END;
17  863 --          IF fRsrcOpen AND NOT shouldOpenRsrc THEN
17  864 1-              BEGIN
17  865 --              rsrcRefNum := fRsrcRefNum;
17  866 --              UseResFile(rsrcRefNum);
17  867 -1              END;
17  868 --
17  869 --          DoRead(dataRefnum, {rsrcExists:} rsrcRefnum <> kNoFileRefnum, forPrinting);
17  870 --
17  871 --          fChangeCount := 0;
17  872 --
17  873 --          Success(fi);
17  874 --
17  875 --          IF fDataOpen THEN
17  876 1-              BEGIN
17  877 --              fDataRefnum := dataRefnum;      {save valid refnum}
17  878 --              dataRefnum := kNoFileRefnum;    {make sure CloseFile doesn't close it}
17  879 -1              END
17  880 --          ELSE
17  881 --              fDataRefnum := kNoFileRefnum;   {save the no file refnum}
17  882 --
17  883 --          IF fRsrcOpen THEN
17  884 1-              BEGIN
17  885 --              fRsrcRefnum := rsrcRefnum;      {save valid refnum}
17  886 --              rsrcRefnum := kNoFileRefnum;    {make sure CloseFile doesn't close it}
17  887 -1              END
17  888 --          ELSE
17  889 --              fRsrcRefnum := kNoFileRefnum;   {save the no file refnum}
17  890 --
```

```
17  891 --          FailOSErr(CloseFile(dataRefnum, rsrcRefnum));
17  892 --
17  893 --          fModDate := FileModDate(anAppFile.fName, anAppFile.vRefnum);
17  894 -0 A    END;
17  895 --
17  896 --
17  897 --      {$S MAWriteFile}
17  898 --      {-----------------------------------+
17  899 --      |   RequestFileName                 |
17  900 --      +-----------------------------------}
17  901 -- A    PROCEDURE TDocument.RequestFileName (itsCmdNumber:  CmdNumber;
17  902 --                                           makingCopy:    BOOLEAN;
17  903 --                                           VAR filename:  Str255;
17  904 --                                           VAR volRefNum: INTEGER);
17  905 --
17  906 --      VAR
17  907 --          reply:          SFReply;
17  908 --          dlgID:          INTEGER;
17  909 --          prompt:         Str255;
17  910 --          dlgLoc:         Point;
17  911 --          dlgHook:        ProcPtr;
17  912 --          filterProc:     ProcPtr;
17  913 --          otherDoc:       TDocument;
17  914 --          err:            OSErr;
17  915 --
17  916 0- A    BEGIN
17  917 --          filename := fTitle;
17  918 --          SFPutParms(itsCmdNumber, dlgID, dlgLoc, filename, prompt, dlgHook, filterProc);
17  919 --
17  920 --      {$IFC qDebug}
17  921 --          gRsrcCheck := 0; {force immediate check}
17  922 --      {$ENDC}
17  923 --
17  924 --          {Update all the windows to avoid a bug in Standard File in which
17  925 --              you can't mount a disk correctly when window updates are pending.}
17  926 --          gApplication.UpdateAllWindows;
17  927 --
17  928 --          SFPPutFile(dlgLoc, prompt, filename, dlgHook, reply, dlgID, filterProc);
17  929 --
17  930 --          IF reply.good THEN
17  931 1-              BEGIN
17  932 --              filename := reply.fName;
17  933 --              volRefnum := reply.vRefnum;
17  934 --
17  935 --              {See if there is an open document with the same name.  If there
17  936 --                  is, tell it we're trying to save it again, which will
17  937 --                  ordinarily result in failure.}
17  938 --              otherDoc := gApplication.AlreadyOpen(filename, volRefnum);
17  939 --              IF otherDoc <> NIL THEN
17  940 --                  otherDoc.SaveAgain(itsCmdNumber, makingCopy, SELF);
17  941 --
17  942 --              {User has already confirmed deleting target in this case,
17  943 --                  so trash file and get maximum disk space.}
17  944 --              err := DeleteFile(@filename, volRefnum);
17  945 --              IF (err <> noErr) AND (err <> fnfErr) THEN
17  946 --                  Failure(err, 0);
17  947 -1              END
17  948 --          ELSE
17  949 --              Failure(noErr, msgCancelled); {user cancelled}
17  950 -0 A    END {TDocument.RequestFileName};
17  951 --
17  952 --
17  953 --      {$S MAReadFile}
17  954 --      {-----------------------------------+
17  955 --      |   Revert                          |
17  956 --      +-----------------------------------}
17  957 -- A    PROCEDURE TDocument.Revert;
17  958 --          VAR anAppFile:  AppFile;
17  959 --              fi:         FailInfo;
17  960 --
17  961 --          {-------------------------------+
17  962 --          |   HdlRevert                   |
17  963 --          +-------------------------------}
17  964 -- B        PROCEDURE HdlRevert(error: OSErr; message: LONGINT);
17  965 0- B        BEGIN
17  966 --              IF error = fnfErr THEN
17  967 --                  error := errRevertFNF;
17  968 --              IF message = 0 THEN
17  969 --                  gErrorParm3 := fTitle;
17  970 --              FailNewMessage(error, message, msgRevertFailed);
17  971 -0 B        END;
17  972 --
17  973 --          {-------------------------------+
17  974 --          |   ResetPrintHandler           |
17  975 --          +-------------------------------}
17  976 -- B        PROCEDURE ResetPrintHandler(view: TView);
17  977 0- B        BEGIN
17  978 --              IF view.fPrintHandler <> NIL THEN
17  979 --                  view.fPrintHandler.Reset;
17  980 -0 B        END;
17  981 --
17  982 0- A    BEGIN
17  983 --          CatchFailures(fi, HdlRevert);
17  984 --
17  985 --          CheckDiskFile(kIDBuzzString, bzRevertAnyways, {reverting:}TRUE);
17  986 --
17  987 --          IF gLastCommand <> NIL THEN
17  988 --              IF gLastCommand.fChangedDocument = SELF THEN
17  989 --                  gApplication.CommitLastCommand;
```

```
17  990 --
17  991 --          FreeData;
17  992 --
17  993 --          IF fSaveExists THEN
17  994 1-              BEGIN
17  995 --              anAppFile.fName := ''; {signal that we are reverting}
17  996 --              ReadFromFile(anAppFile, kForDisplay);
17  997 -1              END
17  998 --          ELSE
17  999 1-              BEGIN
17 1000 --              fViewList.Each(ResetPrintHandler);
17 1001 --              DoInitialState;
17 1002 -1              END;
17 1003 --
17 1004 --          fChangeCount := 0;
17 1005 --
17 1006 --          Success(fi);
17 1007 -0 A    END;
17 1008 --
17 1009 --
17 1010 --          {Notes:
17 1011 --              Parameter combinations:
17 1012 --                  ask?    copy?
17 1013 --                   T        F     Save As... or Save of Untitled doc
17 1014 --                   T        T     Save a Copy In...
17 1015 --                   F        F     Save of a titled doc
17 1016 --
17 1017 --              Caller is responsible for passing askForFilename = TRUE
17 1018 --                  if the document is currently untitled.
17 1019 --
17 1020 --              If askForFilename is FALSE, we force switchToTarget to TRUE.
17 1021 --
17 1022 --              We call SaveInPlace only if there is insufficient disk space
17 1023 --                  to save a temporary copy and we are allowed to save in
17 1024 --                  place (according to the fSaveInPlace field).
17 1025 --
17 1026 --              If askForFilename is TRUE, SaveInPlace should not be called
17 1027 --                  since we delete the target immediately.  (If there is not
17 1028 --                  enough room to save via a temporary file, we would not
17 1029 --                  be able to save in place either.)
17 1030 --          }
17 1031 --
17 1032 --      {$S MAWriteFile}
17 1033 --      {---------------------------------+
17 1034 --      |   Save                          |
17 1035 --      +---------------------------------}
17 1036 -- A    PROCEDURE TDocument.Save(itsCmdNumber: CmdNumber;
17 1037 --                               askForFilename, makingCopy: BOOLEAN);
17 1038 --
17 1039 --      VAR
17 1040 --          name:           Str255;
17 1041 --          volRefnum:      INTEGER;
17 1042 --
17 1043 --          hPB:            HParamBlockRec;
17 1044 --
17 1045 --          dataBytes:      LONGINT;
17 1046 --          rsrcBytes:      LONGINT;
17 1047 --          neededBlks:     LONGINT;
17 1048 --          usedBlks:       LONGINT;
17 1049 --          freeBlks:       LONGINT;
17 1050 --          blkSize:        LONGINT;
17 1051 --
17 1052 --          copyFInfo:      BOOLEAN;
17 1053 --          canSaveInPlace: BOOLEAN;
17 1054 --          oldFlag:        BOOLEAN;
17 1055 --          fi:             FailInfo;
17 1056 --          err:            OSErr;
17 1057 --
17 1058 --          otherDoc:       TDocument;
17 1059 --
17 1060 --          {-------------------------------+
17 1061 --          |   HdlSave                     |
17 1062 --          +-------------------------------}
17 1063 -- B        PROCEDURE HdlSave(error: INTEGER; message: LONGINT);
17 1064 --              VAR newMsg: LONGINT;
17 1065 0- B        BEGIN
17 1066 --              err := FlushVol(NIL, volRefnum);
17 1067 --
17 1068 --              IF message = 0 THEN
17 1069 --                  gErrorParm3 := name;
17 1070 --
17 1071 --              IF NOT askForFilename THEN
17 1072 --                  newMsg := msgSaveFailed
17 1073 --              ELSE IF makingCopy THEN
17 1074 --                  newMsg := msgSaveCopyFailed
17 1075 --              ELSE
17 1076 --                  newMsg := msgSaveAsFailed;
17 1077 --
17 1078 --              FailNewMessage(error, message, newMsg);
17 1079 -0 B        END;
17 1080 --
17 1081 0- A    BEGIN
17 1082 --          CatchFailures(fi, HdlSave);
17 1083 --
17 1084 --      {Step 1: Get the target of the save}
17 1085 --          {Caller should set askForFilename if this is an Untitled document}
17 1086 --          IF askForFilename THEN
17 1087 --              RequestFileName(itsCmdNumber, makingCopy, name, volRefnum)
17 1088 --          ELSE
```

```
17 1089 1-            BEGIN
17 1090 --            name := fTitle;
17 1091 --            volRefnum := fVolRefnum;
17 1092 -1            END;
17 1093 --
17 1094 --
17 1095 --      {Step 2: Decide whether to save with a temporary file or in place,
17 1096 --            and call appropriate method (SaveViaTemp or SaveInPlace).}
17 1097 --
17 1098 --        copyFInfo := NOT (askForFilename OR makingCopy);
17 1099 --
17 1100 --        IF copyFInfo THEN
17 1101 --            CheckDiskFile(kIDBuzzString, bzSaveAnyways, {reverting:}FALSE);
17 1102 --
17 1103 --        AboutToSave(itsCmdNumber, name, volRefnum, makingCopy);
17 1104 --
17 1105 --        IF fCommitOnSave OR NOT makingCopy THEN
17 1106 --            IF gLastCommand <> NIL THEN
17 1107 --                IF gLastCommand.fChangedDocument = SELF THEN
17 1108 --                    gApplication.CommitLastCommand;
17 1109 --
17 1110 --            {Get information about the volume saving to}
17 1111 --        WITH hPB DO
17 1112 1-            BEGIN
17 1113 --            ioNamePtr := NIL;
17 1114 --            ioVRefnum := volRefnum;
17 1115 --            ioVolIndex := 0;
17 1116 -1            END;
17 1117 --        FailOSErr(PBHGetVInfo(@hPB, FALSE));
17 1118 --
17 1119 --            {on HFS ioVFrBlk is an unsigned INTEGER; on MFS it is
17 1120 --                limited to a positive signed INTEGER}
17 1121 --        freeBlks := BAND(hPB.ioVFrBlk, $0000FFFF) - 1; {-1 for some slop -- don't try to fill up the disk completely}
17 1122 --
17 1123 --            {compute size needed to save document}
17 1124 --        blkSize := hPB.ioVAlBlkSiz;
17 1125 --
17 1126 --        dataBytes := 0;
17 1127 --        rsrcBytes := 0;
17 1128 --        DoNeedDiskSpace(dataBytes, rsrcBytes);
17 1129 --        neededBlks := NumBlocks(rsrcBytes, blkSize) + NumBlocks(dataBytes, blkSize);
17 1130 --
17 1131 --        IF freeBlks >= neededBlks THEN
17 1132 --                {enough disk space to create a second copy of document}
17 1133 --            SaveViaTemp(itsCmdNumber, makingCopy, copyFInfo, name, volRefnum)
17 1134 --        ELSE
17 1135 1-            BEGIN       {cannot make a duplicate of document}
17 1136 --            {Check to see if we can save the file in place.}
17 1137 --
17 1138 --            canSaveInPlace := FALSE;    {default value}
17 1139 --
17 1140 --            IF fSaveInPlace <> sipNever THEN
17 1141 2-                BEGIN
17 1142 --                {See if target exists, if the disk space it uses is
17 1143 --                    enough to allow us to save the file after deleting it.}
17 1144 --                err := GetFileInfo(name, volRefnum, hPB);
17 1145 --
17 1146 --                IF err = noErr THEN
17 1147 3-                    BEGIN
17 1148 --                        {compute # block used by target}
17 1149 --                    usedBlks := NumBlocks(hPB.ioFlRPyLen, blkSize) +
17 1150 --                                NumBlocks(hPB.ioFlPyLen, blkSize);
17 1151 --
17 1152 --                    IF neededBlks <= usedBlks + freeBlks THEN
17 1153 --                            {we could save if target is deleted first}
17 1154 4-                        BEGIN
17 1155 --                        IF fSaveInPlace = sipAskUser THEN
17 1156 5-                            BEGIN
17 1157 --                            ParamText(name, '', '', '');
17 1158 --                            IF MacAppAlert(phPurgeOld, NIL) =   kYesButton THEN
17 1159 --                                canSaveInPlace := TRUE
17 1160 --                            ELSE
17 1161 --                                Failure(noErr, msgCancelled);
17 1162 -5                            END
17 1163 --                        ELSE    {we know fSaveInPlace <> sipNever;
17 1164 --                                    it must be sipAlways}
17 1165 --                            canSaveInPlace := TRUE;
17 1166 -4                        END;
17 1167 -3                    END
17 1168 --                ELSE IF err <> fnfErr THEN
17 1169 --                    Failure(err, 0);
17 1170 -2                END;
17 1171 --
17 1172 --            IF canSaveInPlace THEN
17 1173 --                SaveInPlace(itsCmdNumber, makingCopy, copyFInfo, name, volRefnum)
17 1174 --            ELSE
17 1175 --                Failure(dskFulErr, 0);
17 1176 -1            END;
17 1177 --
17 1178 --        Success(fi);            {??? Put later in proc ???}
17 1179 --
17 1180 --    {$IFC qDebug}
17 1181 --        err := GetFileInfo(name, volRefnum, hPB);
17 1182 --        IF err = noErr THEN
17 1183 1-            BEGIN
17 1184 --            usedBlks := NumBlocks(hPB.ioFlRPyLen, blkSize) + NumBlocks(hPB.ioFlPyLen, blkSize);
17 1185 --            IF usedBlks <> neededBlks THEN
17 1186 2-                BEGIN
17 1187 --                Writeln('In TDocument.Save: DoNeedDiskSpace estimated disk space incorrectly.');
```

```
17 1188 --                Writeln('estimated # disk blocks = ', neededBlks:1);
17 1189 --                Writeln('   actual # disk blocks = ', usedBlks:1);
17 1190 -2                END;
17 1191 -1            END;
17 1192 --      {$ENDC}
17 1193 --
17 1194 --      {Step 3: Tell the document that the save was successful.}
17 1195 --          IF NOT makingCopy THEN
17 1196 --              SavedOn(name, volRefnum);
17 1197 --
17 1198 --          err := FlushVol(NIL, volRefnum);
17 1199 -0 A    END;
17 1200 --
17 1201 --
17 1202 --      {$S MAWriteFile}
17 1203 --      {-----------------------------------+
17 1204 --      |   SaveAgain                       |
17 1205 --      +-----------------------------------}
17 1206 -- A    PROCEDURE TDocument.SaveAgain(itsCmdNumber: CmdNumber; makingCopy: BOOLEAN;
17 1207 --                                savingDoc: TDocument);
17 1208 0- A    BEGIN
17 1209 --          {Don't save the file if another one of the same name is already open.}
17 1210 --          IF savingDoc <> SELF THEN
17 1211 --              Failure(errSaveAgain, 0);
17 1212 -0 A    END;
17 1213 --
17 1214 --
17 1215 --      {$S MAWriteFile}
17 1216 --      {-----------------------------------+
17 1217 --      |   SavedOn                         |
17 1218 --      +-----------------------------------}
17 1219 -- A    PROCEDURE TDocument.SavedOn(VAR fileName: Str255; volRefnum: INTEGER);
17 1220 --      VAR
17 1221 --          dataRefnum: INTEGER;
17 1222 --          rsrcRefnum: INTEGER;
17 1223 --
17 1224 0- A    BEGIN
17 1225 --          fChangeCount := 0;
17 1226 --          fSaveExists := TRUE;
17 1227 --
17 1228 --          IF fTitle <> fileName THEN
17 1229 --              SetTitle(fileName);
17 1230 --          fVolRefnum := volRefnum;
17 1231 --
17 1232 --          fModDate := FileModDate(fileName, volRefnum);
17 1233 --
17 1234 --          FailOSErr(OpenAFile(fileName,   volRefnum,
17 1235 --                              fDataOpen,  fRsrcOpen,
17 1236 --                              fDataPerm,  fRsrcPerm,
17 1237 --                              dataRefnum, rsrcRefnum));
17 1238 --          fDataRefnum := dataRefnum;
17 1239 --          fRsrcRefnum := rsrcRefnum;
17 1240 -0 A    END;
17 1241 --
17 1242 --
17 1243 --      {$S MAWriteFile}
17 1244 --      {-----------------------------------+
17 1245 --      |   SaveInPlace                     |
17 1246 --      +-----------------------------------}
17 1247 -- A    PROCEDURE TDocument.SaveInPlace(itsCmdNumber: CmdNumber;
17 1248 --                              makingCopy, copyFInfo: BOOLEAN;
17 1249 --                              VAR fileName: Str255; volRefnum: INTEGER);
17 1250 --          {fileName is VAR only to avoid copying}
17 1251 --      VAR cInfo:      CInfoPBRec;
17 1252 --          validInfo:  BOOLEAN;
17 1253 --          err:        OSErr;
17 1254 0- A    BEGIN
17 1255 --          IF NOT (fDataOpen OR fRsrcOpen) THEN
17 1256 1-              BEGIN
17 1257 --              validInfo := GetSaveInfo(itsCmdNumber, copyFInfo, cInfo);
17 1258 --
17 1259 --                  {Tell document that the file is going away.}
17 1260 --              FreeFile;
17 1261 --
17 1262 --                  {Delete the current file.}
17 1263 --              err := DeleteFile(@fileName, volRefnum);
17 1264 --              IF err <> noErr THEN
17 1265 --                  IF err <> fnfErr THEN
17 1266 --                      Failure(err, 0);
17 1267 --
17 1268 --                  {Save a new copy.}
17 1269 --              cInfo.ioNamePtr := @fileName;
17 1270 --              cInfo.ioVRefnum := volRefnum;
17 1271 --
17 1272 --              MakeNewCopy(makingCopy, validInfo, cInfo);
17 1273 -1              END
17 1274 --          ELSE
17 1275 1-              BEGIN
17 1276 --      {$IFC qDebug}
17 1277 --              ProgramBreak('You must override TDocument.SaveInPlace for a disk-based document.');
17 1278 --      {$ENDC}
17 1279 -1              END;
17 1280 -0 A    END;
17 1281 --
17 1282 --
17 1283 --      {$S MAWriteFile}
17 1284 --      {-----------------------------------+
17 1285 --      |   SaveViaTemp                     |
17 1286 --      +-----------------------------------}
```

```
17 1287 -- A   PROCEDURE TDocument.SaveViaTemp(itsCmdNumber: CmdNumber;
17 1288 --                                     makingCopy, copyFInfo: BOOLEAN;
17 1289 --                                     VAR fileName: Str255; volRefnum: INTEGER);
17 1290 --         {fileName is VAR only to avoid copying}
17 1291 --       VAR cInfo:      CInfoPBRec;
17 1292 --           validInfo:  BOOLEAN;
17 1293 --           tmpName:    Str255;
17 1294 --           fi:         FailInfo;
17 1295 --           err:        OSErr;
17 1296 --
17 1297 --       {-------------------------------+
17 1298 --       |   HdlSvTemp                   |
17 1299 --       +-------------------------------}
17 1300 -- B     PROCEDURE HdlSvTemp(error: OSErr; message: LONGINT);
17 1301 --           VAR err:    OSErr;
17 1302 0- B     BEGIN
17 1303 --           err := DeleteFile(@tmpName, volRefnum);
17 1304 --           {$IFC qDebug}
17 1305 --           IF err <> noErr THEN
17 1306 --               IF err <> fnfErr THEN
17 1307 --                   Writeln('In HdlSvTemp: error from DeleteFile is ', err:1);
17 1308 --           {$ENDC}
17 1309 -0 B     END;
17 1310 --
17 1311 --       {-------------------------------+
17 1312 --       |   HdlSaveFailed               |
17 1313 --       +-------------------------------}
17 1314 -- B     PROCEDURE HdlSaveFailed(error: OSErr; message: LONGINT);
17 1315 --       VAR
17 1316 --           dataRefnum: INTEGER;
17 1317 --           rsrcRefnum: INTEGER;
17 1318 --           fi:         FailInfo;
17 1319 --
17 1320 --           {If reopen attempt fails, make sure original error gets through.}
17 1321 --           {-------------------------------+
17 1322 --           |   HdlFailFailed               |
17 1323 --           +-------------------------------}
17 1324 -- C         PROCEDURE HdlFailFailed(newError: OSErr; newMessage: LONGINT);
17 1325 0- C         BEGIN
17 1326 --               Failure(error, message);
17 1327 -0 C         END;
17 1328 --
17 1329 0- B     BEGIN
17 1330 --           HdlSvTemp(error, message);
17 1331 --           IF fSaveExists AND NOT makingCopy THEN
17 1332 1-               BEGIN
17 1333 --               CatchFailures(fi, HdlFailFailed);
17 1334 --               FailOSErr(OpenAFile(fTitle,         fVolRefNum,
17 1335 --                                   fDataOpen,      fRsrcOpen,
17 1336 --                                   fDataPerm,      fRsrcPerm,
17 1337 --                                   dataRefnum,     rsrcRefnum));
17 1338 --               Success(fi);
17 1339 --               fDataRefnum := dataRefnum;
17 1340 --               fRsrcRefnum := rsrcRefnum;
17 1341 -1                END;
17 1342 -0 B     END;
17 1343 --
17 1344 0- A   BEGIN
17 1345 --       validInfo := GetSaveInfo(itsCmdNumber, copyFInfo, cInfo);
17 1346 --
17 1347 --       GetTempName(tmpName);
17 1348 --
17 1349 --       cInfo.ioNamePtr := @tmpName;
17 1350 --       cInfo.ioVRefnum := volRefnum;
17 1351 --
17 1352 --       MakeNewCopy(makingCopy, validInfo, cInfo);
17 1353 --
17 1354 --       CatchFailures(fi, HdlSvTemp);
17 1355 --
17 1356 --           {Tell document that the file is going away}
17 1357 --       IF NOT makingCopy THEN
17 1358 --           FreeFile;
17 1359 --
17 1360 --       Success(fi);
17 1361 --
17 1362 --       CatchFailures(fi, HdlSaveFailed);
17 1363 --
17 1364 --           {Delete the old copy if it exists.}
17 1365 --       err := DeleteFile(@fileName, volRefnum);
17 1366 --       IF err <> noErr THEN
17 1367 --           IF err <> fnfErr THEN
17 1368 --               Failure(err, 0);
17 1369 --
17 1370 --       FailOSErr(Rename(tmpName, volRefnum, fileName));
17 1371 --
17 1372 --       Success(fi);
17 1373 -0 A   END;
17 1374 --
17 1375 --
17 1376 --     {$S MARes}
17 1377 --     {---------------------------------+
17 1378 --     |   SetTitle                      |
17 1379 --     +---------------------------------}
17 1380 -- A   PROCEDURE TDocument.SetTitle(aTitle: Str255);
17 1381 --
17 1382 --       {-------------------------------+
17 1383 --       |   InstallTitle                |
17 1384 --       +-------------------------------}
17 1385 -- B     PROCEDURE InstallTitle(aWindow: TWindow);
```

```
17 1386 0- B       BEGIN
17 1387 --             aWindow.SetTitleForDoc(aTitle);
17 1388 -0 B       END;
17 1389 0- A   BEGIN
17 1390 --         fTitle := aTitle;
17 1391 --         ForAllWindowsDo(InstallTitle);
17 1392 -0 A   END;
17 1393 --
17 1394 --
17 1395 --     {$S MAWriteFile}
17 1396 --     {-----------------------------------+
17 1397 --     |   SFPutParms                      |
17 1398 --     +-----------------------------------}
17 1399 -- A   PROCEDURE TDocument.SFPutParms(itsCmdNumber: CmdNumber; VAR dlgID: INTEGER;
17 1400 --             VAR where: Point; VAR defaultName, prompt: Str255; VAR dlgHook, filterProc: ProcPtr);
17 1401 --         CONST   kStdSFWhere = 100 * $10000 + 100;
17 1402 --         VAR idx:    INTEGER;
17 1403 --
17 1404 0- A   BEGIN
17 1405 --         dlgID := putDlgID;                      {putDlgID defined by Standard File}
17 1406 --         where := Point(kStdSFWhere);
17 1407 --
17 1408 1-         CASE itsCmdNumber OF
17 1409 --             cSave, cSaveAs: idx := bzSaveAs;
17 1410 --             cSaveCopy:      idx := bzSaveCopy;
17 1411 --             OTHERWISE       idx := 0;
17 1412 -1             END;
17 1413 --
17 1414 --         IF idx = 0 THEN
17 1415 --             prompt := ''
17 1416 --         ELSE
17 1417 --             GetIndString(prompt, kIDBuzzString, idx);
17 1418 --
17 1419 --         dlgHook := NIL;
17 1420 --         filterProc := NIL;
17 1421 -0 A   END;
17 1422 --
17 1423 --
17 1424 --     {$S MAReadFile}
17 1425 --     {-----------------------------------+
17 1426 --     |   ShowReverted                    |
17 1427 --     +-----------------------------------}
17 1428 -- A   PROCEDURE TDocument.ShowReverted;
17 1429 --         {-------------------------------+
17 1430 --         |   RevertView                  |
17 1431 --         +-------------------------------}
17 1432 -- B       PROCEDURE RevertView(aView: TView);
17 1433 0- B       BEGIN
17 1434 --             aView.ShowReverted;
17 1435 -0 B       END;
17 1436 0- A   BEGIN
17 1437 --         ForAllViewsDo(RevertView);
17 1438 -0 A   END;
17 1439 --
17 1440 --
17 1441 --     {$S MAOpen}
17 1442 --     {-----------------------------------+
17 1443 --     |   ShowWindows                     |
17 1444 --     +-----------------------------------}
17 1445 -- A   PROCEDURE TDocument.ShowWindows;
17 1446 --         {-------------------------------+
17 1447 --         |   ShowAWindow                 |
17 1448 --         +-------------------------------}
17 1449 -- B       PROCEDURE ShowAWindow(aWindow: TWindow);
17 1450 0- B       BEGIN
17 1451 --             IF aWindow.fOpenInitially THEN
17 1452 --                 aWindow.Open;
17 1453 -0 B       END;
17 1454 0- A   BEGIN
17 1455 --         ForAllWindowsDo(ShowAWindow);
17 1456 -0 A   END;
17 1457 --
17 1458 --
17 1459 --     {$S MAOpen}
17 1460 --     {-----------------------------------+
17 1461 --     |   UntitledName                    |
17 1462 --     +-----------------------------------}
17 1463 -- A   PROCEDURE TDocument.UntitledName(VAR noName: Str255);
17 1464 --         VAR preInsert:  INTEGER;
17 1465 --             constChars: INTEGER;
17 1466 --             num:        Str255;
17 1467 0- A   BEGIN
17 1468 --         GetIndString(noName, kIDBuzzString, bzUntitled);
17 1469 --         IF ParseTitleTemplate(noName, preInsert, constChars) THEN
17 1470 1-             BEGIN
17 1471 --             NumToString(gNumUntitled, num);
17 1472 --
17 1473 --             IF SubstituteInTitle(noName, num, preInsert, constChars) THEN
17 1474 --                 gNumUntitled := gNumUntitled + 1;
17 1475 -1             END;
17 1476 -0 A   END;
17 1477 --
17 1478 --
17 1479 --     {$IFC qDebug}
17 1480 --         {$S MAFields}
17 1481 --         {-----------------------------------+
17 1482 --         |   Fields                          |
17 1483 --         +-----------------------------------}
17 1484 -- A       PROCEDURE TDocument.Fields (PROCEDURE DoToField (fieldName: Str255;
```

```
17 1485 --                                                      fieldAddr: Ptr;
17 1486 --                                                      fieldType: INTEGER));
17 1487 --
17 1488 0- A        BEGIN
17 1489 --              DoToField('TDocument', NIL, bClass);
17 1490 --              DoToField('fWindowList', @fWindowList, bObject);
17 1491 --              DoToField('fViewList', @fViewList, bObject);
17 1492 --              DoToField('fChangeCount', @fChangeCount, bLongInt);
17 1493 --              DoToField('fDocPrintHandler', @fDocPrintHandler, bObject);
17 1494 --              DoToField('fSavePrintInfo', @fSavePrintInfo, bBoolean);
17 1495 --              DoToField('fSharePrintInfo', @fSharePrintInfo, bBoolean);
17 1496 --              DoToField('fPrintInfo', @fPrintInfo, bHandle);
17 1497 --              DoToField('fTitle', @fTitle, bString);
17 1498 --              DoToField('fFileType', @fFileType, bOSType);
17 1499 --              DoToField('fCreator', @fCreator, bOSType);
17 1500 --              DoToField('fVolRefNum', @fVolRefNum, bInteger);
17 1501 --              DoToField('fModDate', @fModDate, bLongInt);
17 1502 --              DoToField('fReopenAlert', @fReopenAlert, bBoolean);
17 1503 --              DoToField('fSaveExists', @fSaveExists, bBoolean);
17 1504 --              DoToField('fCommitOnSave', @fCommitOnSave, bBoolean);
17 1505 --              DoToField('fUsesDataFork', @fUsesDataFork, bBoolean);
17 1506 --              DoToField('fUsesRsrcFork', @fUsesRsrcFork, bBoolean);
17 1507 --              DoToField('fDataOpen', @fDataOpen, bBoolean);
17 1508 --              DoToField('fRsrcOpen', @fRsrcOpen, bBoolean);
17 1509 --              DoToField('fDataPerm', @fDataPerm, bInteger);
17 1510 --              DoToField('fRsrcPerm', @fRsrcPerm, bInteger);
17 1511 --              DoToField('fDataRefnum', @fDataRefnum, bInteger);
17 1512 --              DoToField('fRsrcRefNum', @fRsrcRefNum, bInteger);
17 1513 --              DoToField('fSaveInPlace', @fSaveInPlace, bByte);
17 1514 --              INHERITED Fields(DoToField);
17 1515 -0 A        END {TDocument.Fields};
17 1516 --      {$ENDC}
13 3005 --      {$I UMacApp.TView.p}
```

```
18    1 --      {$P}
18    2 --      { UMacApp.TView.p }
18    3 --      { Copyright © 1984-1988 Apple Computer Inc.  All rights reserved. }
18    4 --
18    5 --      (*****************************************************************************************)
18    6 --      (*      G L O B A L   R O U T I N E S                                                    *)
18    7 --      (*****************************************************************************************)
18    8 --
18    9 --      {$IFC qDebug}
18   10 --      {$S MADebug}
18   11 --      {-----------------------------------+
18   12 --      |   WriteFocus                      |
18   13 --      +-----------------------------------}
18   14 -- A    PROCEDURE WriteFocus;
18   15 0- A    BEGIN
18   16 --          WrLblVPt( '  gLongOffset', gLongOffset);
18   17 --          WRITELN;
18   18 --          WrLblRect('     portRect', thePort^.portRect);
18   19 --          WRITELN;
18   20 --          WrLblRect('       visRgn', thePort^.visRgn^^.rgnBox);
18   21 --          WRITELN;
18   22 --          WrLblRect('      clipRgn', thePort^.clipRgn^^.rgnBox);
18   23 --          WRITELN;
18   24 -0 A    END {WriteFocus};
18   25 --      {$ENDC}
18   26 --
18   27 --      {$S MAWriteRes}
18   28 --      {-----------------------------------+
18   29 --      |   StripLong                       |
18   30 --      +-----------------------------------}
18   31 -- A    FUNCTION  StripLong (address: UNIV Ptr): LONGINT;
18   32 --
18   33 0- A    BEGIN
18   34 --          StripLong := LONGINT(StripAddress(address));
18   35 -0 A    END;
18   36 --
18   37 --      {$S MAWriteRes}
18   38 --      {-----------------------------------+
18   39 --      |   NewViewRsrc                     |   Create a new view resource
18   40 --      +-----------------------------------}
18   41 -- A    FUNCTION  NewViewRsrc (VAR p: UNIV Ptr): ViewRsrcHndl;
18   42 --
18   43 --      VAR
18   44 --          aHandle:        ViewRsrcHndl;
18   45 --
18   46 0- A    BEGIN
18   47 --          { ??? Should this be doing a NewPermHandle here ??? }
18   48 --          aHandle := ViewRsrcHndl(NewHandle(kViewRsrcExpandAmt));
18   49 --          MoveHHi(Handle(aHandle));
18   50 --          HLock(Handle(aHandle));
18   51 --          FailMemError;
18   52 --          WITH aHandle^^ DO
18   53 1-              BEGIN
18   54 --              numViews := 0;
18   55 --              p := @theViews;
18   56 -1              END;
18   57 --          NewViewRsrc := aHandle;
18   58 -0 A    END;
18   59 --
18   60 --      {$S MAWriteRes}
18   61 --      {-----------------------------------+
18   62 --      |   DoneViewRsrc                    |   Size view resource handle back to real size
18   63 --      +-----------------------------------}
18   64 -- A    PROCEDURE DoneViewRsrc (viewRsrc: UNIV Handle; lastPtr: UNIV LONGINT);
18   65 --
18   66 0- A    BEGIN
18   67 --          HUnlock(viewRsrc);
18   68 --          SetHandleSize(viewRsrc, StripLong(lastPtr) - StripLong(viewRsrc^));
18   69 -0 A    END;
18   70 --
18   71 --      {$S MAOpen}
18   72 --      {-----------------------------------+
18   73 --      |   OffsetPtr                       |
18   74 --      +-----------------------------------}
18   75 -- A    PROCEDURE OffsetPtr (VAR p: UNIV LONGINT; offset: LONGINT);
18   76 --
18   77 0- A    BEGIN
18   78 --          p := p + offset;
18   79 --          IF ODD(p) THEN
18   80 --              p := p + 1;
18   81 -0 A    END {OffsetPtr};
18   82 --
18   83 --      {$S MAOpen}
18   84 --      {-----------------------------------+
18   85 --      |   OffsetPtrWStr                   |
18   86 --      +-----------------------------------}
18   87 -- A    PROCEDURE OffsetPtrWStr (VAR p: UNIV LONGINT; offset: LONGINT);
18   88 --
18   89 0- A    BEGIN
18   90 --          OffsetPtr(p, offset - 255 + LENGTH(StringPtr(p + offset - 256)^));
18   91 -0 A    END {OffsetPtrWStr};
18   92 --
18   93 --      {$S MAWriteRes}
18   94 --      {-----------------------------------+
18   95 --      |   ExpandPtr                       |
18   96 --      +-----------------------------------}
18   97 -- A    FUNCTION  ExpandPtr (viewRsrc: UNIV Handle; VAR p: UNIV LONGINT; offset: LONGINT): Ptr;
18   98 --
```

```
18   99 --      VAR
18  100 --          oldOffset:      LONGINT;
18  101 --          rsrcSize:       Size;
18  102 --          desiredEnd:     LONGINT;
18  103 --          rsrcBase:       LONGINT;
18  104 --          currentPtr:     LONGINT;
18  105 --
18  106 0- A    BEGIN
18  107 --          rsrcSize := GetHandleSize(viewRsrc);
18  108 --          rsrcBase := StripLong(viewRsrc^);
18  109 --          currentPtr := StripLong(p);
18  110 --          IF ODD(offset) THEN
18  111 --              offset := offset + 1;
18  112 --          desiredEnd := currentPtr + offset + SIZEOF(INTEGER);
18  113 --
18  114 --          IF desiredEnd >= rsrcBase + rsrcSize THEN
18  115 1-              BEGIN
18  116 --              { This appropriation logic might need some re-examination.  If the size of the added
18  117 --                  template is larger than the minimum amount, then simply the size is added.  If
18  118 --                  the handle is already near to being full, this won't help for the next allocation.
18  119 --                  Maybe it should use a hystersis?… }
18  120 --              oldOffset := currentPtr - rsrcBase;
18  121 --              HUnlock(viewRsrc);
18  122 --              SetHandleSize(viewRsrc, rsrcSize + MAX(kViewRsrcExpandAmt, offset));
18  123 --              FailMemError;
18  124 --              MoveHHi(viewRsrc);
18  125 --              HLock(viewRsrc);
18  126 --              p := LONGINT(viewRsrc^) + oldOffset;
18  127 -1              END;
18  128 --          ExpandPtr := Ptr(p);
18  129 --          OffsetPtr(p, offset);
18  130 -0 A    END;
18  131 --
18  132 --      {$S MAWriteRes}
18  133 --      {-----------------------------------+
18  134 --      |   ExpandPtrWStr                   |
18  135 --      +-----------------------------------}
18  136 -- A    FUNCTION  ExpandPtrWStr (viewRsrc: UNIV Handle; VAR p: UNIV LONGINT;
18  137 --                                           offset, len: LONGINT): Ptr;
18  138 0- A    BEGIN
18  139 --          ExpandPtrWStr := ExpandPtr(viewRsrc, p, offset - 255 + len);
18  140 -0 A    END;
18  141 --
18  142 --      {$S MAMain}
18  143 --      {-----------------------------------+
18  144 --      |   PtIsVisible                     |
18  145 --      +-----------------------------------}
18  146 -- A    FUNCTION  PtIsVisible (pt: Point): BOOLEAN;
18  147 0- A    BEGIN
18  148 --          PtIsVisible := PtInRgn(pt, thePort^.visRgn) & PtInRgn(pt, thePort^.clipRgn);
18  149 -0 A    END {PtIsVisible};
18  150 --
18  151 --      {$S MARes}
18  152 --      {-----------------------------------+
18  153 --      |   VisibleRect                     |
18  154 --      +-----------------------------------}
18  155 -- A    PROCEDURE VisibleRect (VAR r: Rect);
18  156 --
18  157 0- A    BEGIN
18  158 --          IF NOT gDrawingPictScrap THEN
18  159 1-              BEGIN
18  160 --      {$IFC qDebug}
18  161 --              UseTempRgn('VisibleRect');
18  162 --      {$ENDC}
18  163 --              RectRgn(gTempRgn, r);
18  164 --              SectRgn(gTempRgn, thePort^.clipRgn, gTempRgn);
18  165 --              r := gTempRgn^^.rgnBBox;
18  166 --      {$IFC qDebug}
18  167 --              DoneWithTempRgn;
18  168 --      {$ENDC}
18  169 -1              END;
18  170 -0 A    END {VisibleRect};
18  171 --
18  172 --      (*****************************************************************************************)
18  173 --      (*      T V i e w                                                                        *)
18  174 --      (*****************************************************************************************)
18  175 --      {-----------------------------------+
18  176 --      |   IView                           |
18  177 --      +-----------------------------------}
18  178 --      {$S MAOpen}
18  179 -- A    PROCEDURE TView.IView (itsDocument: TDocument; itsSuperview: TView; itsLocation: VPoint; itsSize: VPoint;
18  180 --                  itsHSizeDet, itsVSizeDet: SizeDeterminer);
18  181 --
18  182 --      VAR
18  183 --          fi:         FailInfo;
18  184 --
18  185 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
18  186 0- B       BEGIN
18  187 --              Free;
18  188 -0 B       END;
18  189 --
18  190 0- A    BEGIN
18  191 --          fSuperview := itsSuperview;
18  192 --          fSubViews := NIL;
18  193 --          fDocument := itsDocument;
18  194 --          fLocation := itsLocation;
18  195 --          fSize := itsSize;
18  196 --          fSizeDeterminer[h] := itsHSizeDet;
18  197 --          fSizeDeterminer[v] := itsVSizeDet;
```

```
18  198 --          fHLDesired := hlOff;
18  199 --          fIdentifier := kNoIdentifier;
18  200 --          fShown := TRUE;
18  201 --          fViewEnabled := TRUE;
18  202 --          fPrintHandler := NIL;
18  203 --
18  204 --          IEvtHandler(itsSuperView);
18  205 --
18  206 --          CatchFailures(fi, HandleFailure);
18  207 --          IF itsSuperView <> NIL THEN
18  208 --              itsSuperView.AddSubView(SELF);
18  209 --          IF itsDocument <> NIL THEN
18  210 --              itsDocument.AddView(SELF);
18  211 --          Success(fi);
18  212 -0 A    END {TView.IView};
18  213 --
18  214 --
18  215 --      {-----------------------------------+
18  216 --      |   IRes                            |
18  217 --      +-----------------------------------}
18  218 --      {$S MAOpen}
18  219 -- A    PROCEDURE TView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
18  220 --
18  221 0- A    BEGIN
18  222 --          WITH ViewTemplatePtr(itsParams)^ DO
18  223 1-              BEGIN
18  224 --              IView(itsDocument, itsSuperView, itsLocation, itsSize,
18  225 --                    SizeDeterminer(itsHSizeDet), SizeDeterminer(itsVSizeDet));
18  226 --              fShown := isShown;
18  227 --              fViewEnabled := isEnabled;
18  228 --              fHLDesired := hlOff;
18  229 --              fIdentifier := thisViewID;
18  230 --              OffsetPtrWStr(itsParams, SIZEOF(ViewTemplate));
18  231 -1              END;
18  232 -0 A    END;
18  233 --
18  234 --
18  235 --      {-----------------------------------+
18  236 --      |   WRes                            |
18  237 --      +-----------------------------------}
18  238 --      {$S MAWriteRes}
18  239 -- A    PROCEDURE TView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr);
18  240 --      VAR
18  241 --          vwPtr:          ViewTemplatePtr;
18  242 --
18  243 0- A    BEGIN
18  244 --          vwPtr := ViewTemplatePtr(ExpandPtrWStr(theResource, itsParams,
18  245 --            SIZEOF(ViewTemplate), LENGTH(gWResType)));
18  246 --
18  247 --          WITH vwPtr^ DO
18  248 1-              BEGIN
18  249 --              IF fSuperView <> NIL THEN
18  250 --                  itsParentID := fSuperView.fIdentifier
18  251 --              ELSE
18  252 --                  itsParentID := kNoIdentifier;
18  253 --              thisViewID := fIdentifier;
18  254 --              itsLocation := fLocation;
18  255 --              itsSize := fSize;
18  256 --              itsVSizeDet := Integer8(fSizeDeterminer[v]);
18  257 --              itsHSizeDet := Integer8(fSizeDeterminer[h]);
18  258 --              isShown := fShown;
18  259 --              isEnabled := fViewEnabled;
18  260 --              itsSignature := gWResSignature;
18  261 --              (* itsType := gWResType; *)
18  262 --              CopyStr255(gWResType, PRStr(itsType));
18  263 -1              END;
18  264 --
18  265 --          { Everybody gets this far - bump number of views in resource by 1 }
18  266 --          WITH theResource^^ DO
18  267 --              numViews := numViews + 1;
18  268 -0 A    END;
18  269 --
18  270 --
18  271 --      {-----------------------------------+
18  272 --      |   WriteRes                        |
18  273 --      +-----------------------------------}
18  274 --      {$S MAWriteRes}
18  275 -- A    PROCEDURE TView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr);
18  276 0- A    BEGIN
18  277 --          gWResSignature := 'view';   gWResType := 'TView';
18  278 --          WRes(theResource, itsParams);
18  279 -0 A    END;
18  280 --
18  281 --
18  282 --      {-----------------------------------+
18  283 --      |   Free                            |
18  284 --      +-----------------------------------}
18  285 --      {$S MAClose}
18  286 -- A    PROCEDURE TView.Free;
18  287 --
18  288 0- A    BEGIN
18  289 --          IF fSubViews <> NIL THEN
18  290 --              fSubViews.FreeList;
18  291 --          fSubViews := NIL;
18  292 --          IF fSuperView <> NIL THEN
18  293 --              fSuperView.RemoveSubView(SELF);
18  294 --          IF fDocument <> NIL THEN
18  295 --              fDocument.DeleteView(SELF);
18  296 --          IF fPrintHandler <> NIL THEN
```

```
18  297 --              fPrintHandler.Free;
18  298 --
18  299 --          INHERITED Free;
18  300 -0 A    END {TView.Free};
18  301 --
18  302 --      {-----------------------------------+
18  303 --      |   Activate                        |
18  304 --      +-----------------------------------}
18  305 --      {$S MAActivate}
18  306 -- A    PROCEDURE TView.Activate (entering: BOOLEAN);
18  307 --
18  308 --          {-------------------------------+
18  309 --          |   ActivateSubView             |
18  310 --          +-------------------------------}
18  311 -- B        PROCEDURE ActivateSubView (theSubView: TView);
18  312 0- B        BEGIN
18  313 --              theSubView.Activate(entering);
18  314 -0 B        END;
18  315 --
18  316 0- A    BEGIN
18  317 --          IF Focus THEN ;
18  318 --          IF entering THEN
18  319 1-              BEGIN
18  320 --              DoHighlightSelection(fHLDesired, hlOn);
18  321 --              fHLDesired := hlOn;
18  322 -1              END
18  323 --          ELSE
18  324 1-              BEGIN
18  325 --              DoHighlightSelection(fHLDesired, hlDim);
18  326 --              fHLDesired := hlDim;
18  327 -1              END;
18  328 --          EachSubView(ActivateSubView);
18  329 -0 A    END {TView.Activate};
18  330 --
18  331 --      {-----------------------------------+
18  332 --      |   AddSubView                      |
18  333 --      +-----------------------------------}
18  334 --      {$S MAOpen}
18  335 -- A    PROCEDURE TView.AddSubView (theSubView: TView);
18  336 --      VAR
18  337 --          itsGrafPort:        GrafPtr;
18  338 --
18  339 0- A    BEGIN
18  340 --          IF theSubView <> NIL THEN
18  341 1-              BEGIN
18  342 --              IF fSubViews = NIL THEN
18  343 2-                  BEGIN
18  344 --                  fSubViews := NewList;
18  345 --      {$IFC qDebug}
18  346 --                  fSubViews.SetEltType('TView');
18  347 --      {$ENDC}
18  348 -2                  END;
18  349 --      {$IFC FALSE}
18  350 --              IF fSubViews.GetItemNumber(theSubView) <> 0 THEN
18  351 --                  ProgramBreak('Attempting to duplicate a subview in AddSubView.');
18  352 --      {$ENDC}
18  353 --
18  354 --              fSubViews.InsertLast(theSubView);
18  355 --
18  356 --              theSubView.fSuperView := SELF;
18  357 --              IF theSubView.fNextHandler = NIL THEN
18  358 --                  theSubView.fNextHandler := SELF;
18  359 --
18  360 --              itsGrafPort := GetGrafPort;
18  361 --              IF itsGrafPort <> NIL THEN
18  362 --                  theSubView.BeInPort(itsGrafPort);
18  363 --      {$IFC FALSE}
18  364 --              theSubView.SuperViewChangedSize(fSize);
18  365 --      {$ENDC}
18  366 --              IF theSubView.fNextHandler = NIL THEN       { If necessary,                          }
18  367 --                  theSubView.fNextHandler := SELF;        { …hook it up to an event hander chain. }
18  368 -1              END;
18  369 -0 A    END {TView.AddSubView};
18  370 --
18  371 --      {$S MAView}
18  372 --      {-----------------------------------+
18  373 --      |   AdjustSize                      |
18  374 --      +-----------------------------------}
18  375 --      {$S MARes}
18  376 -- A    PROCEDURE TView.AdjustSize;
18  377 --      VAR
18  378 --          newSize:    VPoint;
18  379 --
18  380 0- A    BEGIN
18  381 --          newSize := fSize;
18  382 --          ComputeSize(newSize);
18  383 --          IF (newSize.h <> fSize.h) OR (newSize.v <> fSize.v) THEN
18  384 1-              BEGIN
18  385 --              Resize(newSize.h, newSize.v, TRUE);
18  386 --              DoPagination;
18  387 -1              END;
18  388 -0 A    END {TView.AdjustSize};
18  389 --
18  390 --      {-----------------------------------+
18  391 --      |   Adorn                           |
18  392 --      +-----------------------------------}
18  393 --      {$S MARes}
18  394 -- A    PROCEDURE TView.Adorn (area: Rect; itsPenSize: Point; itsAdornment: CntlAdornment);
18  395 --      CONST
```

```
18  396 --          CntlFrame =     [adnLineTop, adnLineLeft, adnLineBottom, adnLineRight];
18  397 --      VAR
18  398 --          shadowed:       BOOLEAN;
18  399 --          savedPenState:  PenState;
18  400 --
18  401 --          {-------------------------------+
18  402 --          |   XFrameOval                  |
18  403 --          +-------------------------------}
18  404 -- B       PROCEDURE XFrameOval;
18  405 0- B       BEGIN
18  406 --              FrameOval(area);
18  407 -0 B       END;
18  408 --
18  409 --          {-------------------------------+
18  410 --          |   XFrameRRect                 |
18  411 --          +-------------------------------}
18  412 -- B       PROCEDURE XFrameRRect;
18  413 0- B       BEGIN
18  414 --              FrameRoundRect(area, 16, 16);
18  415 -0 B       END;
18  416 --
18  417 --          {-------------------------------+
18  418 --          |   DrawWithShadow              |
18  419 --          +-------------------------------}
18  420 -- B       PROCEDURE DrawWithShadow (PROCEDURE DrawCmd);
18  421 --          LABEL
18  422 --              1;
18  423 --          VAR
18  424 --              drawRgn:    RgnHandle;
18  425 --              shadowRgn:  RgnHandle;
18  426 --              fi:         FailInfo;
18  427 --
18  428 --              {---------------------------+
18  429 --              |   HandleFailure           |
18  430 --              +---------------------------}
18  431 -- C           PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
18  432 0- C           BEGIN
18  433 --                  { If we fail allocating memory for the regions, dispose of the regions if
18  434 --                      they exist and continue}
18  435 --                  GOTO 1;
18  436 -0 C           END;
18  437 --
18  438 0- B       BEGIN
18  439 --              IF shadowed THEN
18  440 1-                  BEGIN
18  441 --                  shadowRgn := NIL;               {In case we creating drawRgn fails}
18  442 --                  CatchFailures(fi, HandleFailure);
18  443 --                  drawRgn := MakeNewRgn;
18  444 --                  shadowRgn := MakeNewRgn;
18  445 --
18  446 --                  OpenRgn;
18  447 --                  DrawCmd;
18  448 --                  CloseRgn(drawRgn);
18  449 --
18  450 --                  CopyRgn(drawRgn, shadowRgn);
18  451 --                  OffsetRgn(shadowRgn, itsPenSize.h, itsPenSize.v);
18  452 --                  DiffRgn(shadowRgn, drawRgn, shadowRgn);
18  453 --                  PaintRgn(shadowRgn);
18  454 --                  Success(fi);
18  455 --              1:
18  456 --                  IF drawRgn <> NIL THEN
18  457 --                      DisposeRgn(drawRgn);
18  458 --                  IF shadowRgn <> NIL THEN
18  459 --                      DisposeRgn(shadowRgn);
18  460 -1              END;
18  461 --              DrawCmd;
18  462 -0 B       END;
18  463 --
18  464 --          {-------------------------------+
18  465 --          |   DrawLine                    |
18  466 --          +-------------------------------}
18  467 -- B       PROCEDURE DrawLine(vhs: VHSelect; lh, lv, lto: INTEGER);
18  468 0- B       BEGIN
18  469 --              MoveTo(lh, lv);
18  470 1-              CASE vhs OF
18  471 --                  h:  Line(lto-lh, 0);
18  472 --                  v:  Line(0, lto-lv);
18  473 -1                  END;
18  474 -0 B       END;
18  475 --
18  476 0- A   BEGIN
18  477 --          IF itsAdornment <> [] THEN              { Only do this if adorning is asked for         }
18  478 1-              BEGIN
18  479 --              GetPenState(savedPenState);
18  480 --              PenNormal;
18  481 --              PenSize(itsPenSize.h, itsPenSize.v);
18  482 --
18  483 --              shadowed := adnShadow IN itsAdornment;
18  484 --              IF shadowed THEN
18  485 --                  SubPt(itsPenSize, area.botRight);
18  486 --
18  487 --              IF adnOval IN itsAdornment THEN
18  488 --                  DrawWithShadow(XFrameOval);
18  489 --
18  490 --              IF adnRRect IN itsAdornment THEN
18  491 --                  DrawWithShadow(XFrameRRect);
18  492 --
18  493 --              IF itsAdornment >= CntlFrame THEN   { Special case for whole rect!                  }
18  494 2-                  BEGIN
```

```
18  495 --                FrameRect(area);
18  496 --                IF shadowed THEN
18  497 --                    WITH area DO
18  498 3-                        BEGIN
18  499 --                        DrawLine(h, left + itsPenSize.h, bottom, right);
18  500 --                        DrawLine(v, right, top + itsPenSize.v, bottom);
18  501 -3                        END;
18  502 -2                END
18  503 --            ELSE
18  504 --                WITH area DO
18  505 2-                    BEGIN                       { No shadows for single lines                   }
18  506 --                    IF adnLineTop IN itsAdornment THEN
18  507 --                        DrawLine(h, left, top, right);
18  508 --                    IF adnLineLeft IN itsAdornment THEN
18  509 --                        DrawLine(v, left, top, bottom);
18  510 --                    IF adnLineBottom IN itsAdornment THEN
18  511 --                        DrawLine(h, left, bottom - 1, right);
18  512 --                    IF adnLineRight IN itsAdornment THEN
18  513 --                        DrawLine(v, right - 1, top, bottom);
18  514 -2                    END;
18  515 --            SetPenState(savedPenState);
18  516 -1            END;
18  517 -0 A   END {TView.Adorn};
18  518 --
18  519 --      {-----------------------------------+
18  520 --      |   AttachPrintHandler              |
18  521 --      +-----------------------------------}
18  522 --      {$S MARes}
18  523 -- A    PROCEDURE TView.AttachPrintHandler (itsPrintHandler: TPrintHandler);
18  524 0- A   BEGIN
18  525 --          fPrintHandler := itsPrintHandler;
18  526 --          DoCheckPrinter;
18  527 -0 A   END {TView.AttachPrintHandler};
18  528 --
18  529 --      {-----------------------------------+
18  530 --      |   BeInPort                        |
18  531 --      +-----------------------------------}
18  532 --      {$S MANonRes}
18  533 -- A    PROCEDURE TView.BeInPort (itsPort:  GrafPtr);
18  534 --
18  535 --          {-------------------------------+
18  536 --          |   NotifySubView               |
18  537 --          +-------------------------------}
18  538 -- B        PROCEDURE NotifySubView (theSubView: TView);
18  539 0- B        BEGIN
18  540 --              theSubView.BeInPort(itsPort);
18  541 -0 B        END;
18  542 --
18  543 0- A   BEGIN
18  544 --          IF fPrintHandler <> NIL THEN
18  545 --              DoCheckPrinter;
18  546 --          EachSubView(NotifySubView);
18  547 -0 A   END {TView.BeInPort};
18  548 --
18  549 --      {-----------------------------------+
18  550 --      |   BeInScroller                    |
18  551 --      +-----------------------------------}
18  552 --      {$S MANonRes}
18  553 -- A    PROCEDURE TView.BeInScroller (itsScroller: TScroller);
18  554 --      VAR
18  555 --          scrollLimits:       VPoint;
18  556 --
18  557 0- A   BEGIN
18  558 --          IF itsScroller <> NIL THEN
18  559 1-              BEGIN
18  560 --              scrollLimits := fSize;
18  561 --              AddVPt(fLocation, scrollLimits);
18  562 --              itsScroller.SetScrollLimits(scrollLimits, TRUE);
18  563 -1              END;
18  564 -0 A   END {TView.BeInScroller};
18  565 --
18  566 --      {-----------------------------------+
18  567 --      |   CalcMinSize                     |
18  568 --      +-----------------------------------}
18  569 --      {$S MARes}
18  570 -- A    PROCEDURE TView.CalcMinSize (VAR minSize: VPoint);
18  571 0- A   BEGIN
18  572 --          minSize := fSize;
18  573 -0 A   END {TView.CalcMinSize};
18  574 --
18  575 --      {-----------------------------------+
18  576 --      |   ClipFurtherTo                   |
18  577 --      +-----------------------------------}
18  578 --      {$S MARes}
18  579 -- A    PROCEDURE TView.ClipFurtherTo(r: Rect; hDeltaOrg, vDeltaOrg: INTEGER);
18  580 --          { Set clipping to: (<current clipping> INTERSECT r) OFFSET-BY (hDeltaOrg, vDeltaOrg) }
18  581 --
18  582 0- A   BEGIN
18  583 --          {$IFC qDebug}
18  584 --          UseTempRgn('ClipFurtherTo');
18  585 --          {$ENDC}
18  586 --
18  587 --          RectRgn(gTempRgn, r);
18  588 --          SectRgn(thePort^.clipRgn, gTempRgn, gTempRgn);
18  589 --          OffsetRgn(gTempRgn, hDeltaOrg, vDeltaOrg);
18  590 --          SetClip(gTempRgn);
18  591 --
18  592 --          {$IFC qDebug}
18  593 --          DoneWithTempRgn;
```

```
18  594 --          {$ENDC}
18  595 -0 A    END {TView.ClipFurtherTo};
18  596 --
18  597 --      {-----------------------------------+
18  598 --      |   Close                           |
18  599 --      +-----------------------------------}
18  600 --      {$S MAClose}
18  601 -- A    PROCEDURE TView.Close;
18  602 --
18  603 --          {---------------------------------+
18  604 --          |   CloseSubView                  |
18  605 --          +---------------------------------}
18  606 -- B        PROCEDURE CloseSubView (theSubView: TView);
18  607 0- B        BEGIN
18  608 --              theSubView.Close;
18  609 -0 B        END;
18  610 --
18  611 0- A    BEGIN
18  612 --          EachSubView(CloseSubView);
18  613 -0 A    END {TView.Close};
18  614 --
18  615 --      {-----------------------------------+
18  616 --      |   ComputeSize                     |
18  617 --      +-----------------------------------}
18  618 --      {$S MARes}
18  619 -- A    PROCEDURE TView.ComputeSize (VAR newSize: VPoint);
18  620 --      VAR
18  621 --          dimension:      VCoordinate;
18  622 --          vhs:            VHSelect;
18  623 --          minSize:        VPoint;
18  624 --
18  625 0- A    BEGIN
18  626 --          CalcMinSize(minSize);
18  627 --
18  628 --          { Adjust view size based on size determiners. }
18  629 --
18  630 --          FOR vhs := v TO h DO
18  631 1-              BEGIN
18  632 2-              CASE fSizeDeterminer[vhs] OF
18  633 --                  sizeVariable:
18  634 --                      dimension := minSize.vh[vhs];
18  635 --                  sizeFixed:
18  636 --                      dimension := fSize.vh[vhs];
18  637 --                  sizeRelSuperView:
18  638 --                      dimension := newSize.vh[vhs];   { newSize set in SuperViewChangedSize }
18  639 --                  sizeSuperView:
18  640 --                      IF fSuperView <> NIL THEN
18  641 --                          dimension := fSuperView.fSize.vh[vhs];
18  642 --                  sizePage:
18  643 --                      IF fPrintHandler <> NIL THEN
18  644 --                          dimension := fPrintHandler.fViewPerPage.vh[vhs]
18  645 --      {$IFC qDebug}
18  646 --                      ELSE
18  647 --                          ProgramBreak('Size determiner of sizePage, but no print handler.')
18  648 --      {$ENDC}
18  649 --                      ;
18  650 --                  sizeFillPages:
18  651 --                      IF fPrintHandler <> NIL THEN
18  652 --                          dimension := Min(
18  653 --                              RoundUp(minSize.vh[vhs], fPrintHandler.fViewPerPage.vh[vhs]),
18  654 --                                      kMaxCoord)
18  655 --      {$IFC qDebug}
18  656 --                      ELSE
18  657 --                          ProgramBreak('Size determiner of sizePage, but no print handler.')
18  658 --      {$ENDC}
18  659 --                      ;
18  660 -2                  END;
18  661 --              newSize.vh[vhs] := dimension;
18  662 -1              END;
18  663 -0 A    END {TView.ComputeSize};
18  664 --
18  665 --      {-----------------------------------+
18  666 --      |   ContainsClipType                |
18  667 --      +-----------------------------------}
18  668 --      {$S MARes}
18  669 -- A    FUNCTION  TView.ContainsClipType (aType: ResType): BOOLEAN;
18  670 --      VAR
18  671 --          offset: LONGINT;
18  672 --
18  673 0- A    BEGIN
18  674 --          ContainsClipType := (GetScrap(NIL, aType, offset) > 0)
18  675 -0 A    END {TView.ContainsClipType};
18  676 --
18  677 --      {-----------------------------------+
18  678 --      |   ContainsMouse                   |
18  679 --      +-----------------------------------}
18  680 --      {$S MARes}
18  681 -- A    FUNCTION  TView.ContainsMouse (theMouse: VPoint): BOOLEAN;
18  682 --
18  683 --      VAR
18  684 --          extent:     VRect;
18  685 --
18  686 0- A    BEGIN
18  687 --          GetExtent(extent);
18  688 --          ContainsMouse := IsShown AND PtInVRect(theMouse, extent);
18  689 -0 A    END {TView.ContainsMouse};
18  690 --
18  691 --      {-----------------------------------+
18  692 --      |   CountSubViews                   |
```

```
18  693 --      +-----------------------------------}
18  694 --      {$S MARes}
18  695 -- A    FUNCTION  TView.CountSubViews: INTEGER;
18  696 0- A    BEGIN
18  697 --          IF fSubViews <> NIL THEN
18  698 --              CountSubViews := fSubViews.fSize
18  699 --          ELSE
18  700 --              CountSubViews := 0;
18  701 -0 A    END {TView.CountSubViews};
18  702 --
18  703 --      {-----------------------------------+
18  704 --      |   DoBreakFollowing                |
18  705 --      +-----------------------------------}
18  706 --      {$S MARes}
18  707 -- A    FUNCTION  TView.DoBreakFollowing (vhs: VHSelect; prevBreak: VCoordinate;
18  708 --                                        VAR automatic: BOOLEAN): VCoordinate;
18  709 0- A    BEGIN
18  710 --          DoBreakFollowing := fPrintHandler.BreakFollowing(vhs, prevBreak, automatic);
18  711 -0 A    END {TView.DoBreakFollowing};
18  712 --
18  713 --      {-----------------------------------+
18  714 --      |   DoCalcPageStrips                |
18  715 --      +-----------------------------------}
18  716 --      {$S MANonRes}
18  717 -- A    PROCEDURE TView.DoCalcPageStrips (VAR pageStrips: Point);
18  718 0- A    BEGIN
18  719 --          fPrintHandler.CalcPageStrips(pageStrips);
18  720 -0 A    END {TView.DoCalcPageStrips};
18  721 --
18  722 --      {$S MANonRes}
18  723 --      {-----------------------------------+
18  724 --      |   DoCalcViewPerPage               |
18  725 --      +-----------------------------------}
18  726 --      {$S MANonRes}
18  727 -- A    PROCEDURE TView.DoCalcViewPerPage (VAR viewPerPage: VPoint);
18  728 0- A    BEGIN
18  729 --          viewPerPage := fSize;   {in case fPrintHandler is vacuous}
18  730 --          fPrintHandler.CalcViewPerPage(viewPerPage);
18  731 -0 A    END {TView.DoCalcViewPerPage};
18  732 --
18  733 --      {-----------------------------------+
18  734 --      |   DoCheckPrinter                  |
18  735 --      +-----------------------------------}
18  736 --      {$S MARes}
18  737 -- A    PROCEDURE TView.DoCheckPrinter;
18  738 0- A    BEGIN
18  739 --          fPrintHandler.CheckPrinter
18  740 -0 A    END {TView.DoCheckPrinter};
18  741 --
18  742 --      {-----------------------------------+
18  743 --      |   DoChoice                        |
18  744 --      +-----------------------------------}
18  745 --      {$S MARes}
18  746 -- A    PROCEDURE TView.DoChoice (origView: TView; itsChoice: INTEGER);
18  747 0- A    BEGIN
18  748 --          IF fSuperView <> NIL THEN
18  749 --              fSuperView.DoChoice (origView, itsChoice);
18  750 -0 A    END;
18  751 --
18  752 --      {-----------------------------------+
18  753 --      |   DoDrawPrintFeedback             |
18  754 --      +-----------------------------------}
18  755 --      {$S MARes}
18  756 -- A    PROCEDURE TView.DoDrawPrintFeedback (area: Rect);
18  757 0- A    BEGIN
18  758 --          IF fPrintHandler <> NIL THEN
18  759 --              fPrintHandler.DrawPrintFeedback(area);
18  760 -0 A    END {TView.DoDrawPrintFeedback};
18  761 --
18  762 --      {-----------------------------------+
18  763 --      |   DoDrawPageBreak                 |
18  764 --      +-----------------------------------}
18  765 --      {$S MARes}
18  766 -- A    PROCEDURE TView.DoDrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
18  767 --                                      loc: VCoordinate; automatic: BOOLEAN);
18  768 0- A    BEGIN
18  769 --          fPrintHandler.DrawPageBreak(vhs, whichBreak, loc, automatic);
18  770 -0 A    END {TView.DoDrawPageBreak};
18  771 --
18  772 --      {-----------------------------------+
18  773 --      |   DoHighlightSelection            |
18  774 --      +-----------------------------------}
18  775 --      {$S MARes}
18  776 -- A    PROCEDURE TView.DoHighlightSelection (fromHL, toHL: HLState);
18  777 0- A    BEGIN
18  778 -0 A    END {TView.DoHighlightSelection};
18  779 --
18  780 --      {-----------------------------------+
18  781 --      |   DoMenuCommand                   |
18  782 --      +-----------------------------------}
18  783 --      {$S MASelCommand}
18  784 -- A    FUNCTION  TView.DoMenuCommand (aCmdNumber: CmdNumber): TCommand;
18  785 0- A    BEGIN
18  786 1-          CASE aCmdNumber OF
18  787 --              cPrViewBase..cPrViewMax,
18  788 --              cPrFileBase..cPrFileMax:
18  789 --                  IF fPrintHandler <> NIL THEN
18  790 --                      DoMenuCommand := fPrintHandler.DoMenuCommand(aCmdNumber)
18  791 --                  ELSE
```

```
18  792 --                      DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
18  793 --              OTHERWISE
18  794 --                  DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
18  795 -1              END;
18  796 -0 A    END {TView.DoMenuCommand};
18  797 --
18  798 --      {-----------------------------------+
18  799 --      |   DoMouseCommand                  |
18  800 --      +-----------------------------------}
18  801 --      {$S MASelCommand}
18  802 -- A    FUNCTION  TView.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
18  803 --                                      VAR hysteresis: Point): TCommand;
18  804 0- A    BEGIN
18  805 --          DoMouseCommand := gNoChanges;
18  806 -0 A    END {TView.DoMouseCommand};
18  807 --
18  808 --      {-----------------------------------+
18  809 --      |   DoPagination                    |
18  810 --      +-----------------------------------}
18  811 --      {$S MANonRes}
18  812 -- A    PROCEDURE TView.DoPagination;
18  813 0- A    BEGIN
18  814 --          IF fPrintHandler <> NIL THEN
18  815 --              fPrintHandler.RedoPageBreaks
18  816 -0 A    END {TView.DoPagination};
18  817 --
18  818 --      {-----------------------------------+
18  819 --      |   DoPrinterChanged                |
18  820 --      +-----------------------------------}
18  821 --      {$S MANonRes}
18  822 -- A    PROCEDURE TView.DoPrinterChanged;
18  823 0- A    BEGIN
18  824 --          IF fPrintHandler <> NIL THEN
18  825 --              fPrintHandler.PrinterChanged;
18  826 -0 A    END {TView.DoPrinterChanged};
18  827 --
18  828 --      {-----------------------------------+
18  829 --      |   DoSetPageOffset                 |
18  830 --      +-----------------------------------}
18  831 --      {$S MAPrint}
18  832 -- A    PROCEDURE TView.DoSetPageOffset (coord: VPoint);
18  833 0- A    BEGIN
18  834 --          fPrintHandler.SetPageOffset(coord);
18  835 -0 A    END {TView.DoSetPageOffset};
18  836 --
18  837 --      {-----------------------------------+
18  838 --      |   DoSetCursor                     |
18  839 --      +-----------------------------------}
18  840 --      {$S MARes}
18  841 -- A    FUNCTION  TView.DoSetCursor (localPoint: Point;
18  842 --                                   cursorRgn: RgnHandle): BOOLEAN;
18  843 0- A    BEGIN
18  844 --          DoSetCursor := FALSE;
18  845 -0 A    END {TTEView.DoSetCursor};
18  846 --
18  847 --      {-----------------------------------+
18  848 --      |   DoSetupMenus                    |
18  849 --      +-----------------------------------}
18  850 --      {$S MARes}
18  851 -- A    PROCEDURE TView.DoSetupMenus;
18  852 0- A    BEGIN
18  853 --          INHERITED DoSetupMenus;
18  854 --
18  855 --          IF fPrintHandler <> NIL THEN
18  856 --              fPrintHandler.DoSetupMenus;
18  857 -0 A    END;
18  858 --
18  859 --      {-----------------------------------+
18  860 --      |   Draw                            |
18  861 --      +-----------------------------------}
18  862 --      {$S MARes}
18  863 -- A    PROCEDURE TView.Draw (area: Rect);
18  864 0- A    BEGIN
18  865 -0 A    END {TView.Draw};
18  866 --
18  867 --      {-----------------------------------+
18  868 --      |   DrawContents                    |
18  869 --      +-----------------------------------}
18  870 --      {$S MARes}
18  871 -- A    PROCEDURE TView.DrawContents;
18  872 --      VAR
18  873 --          longOffset:         VPoint;
18  874 --          visRect:            Rect;
18  875 --          savedClip:          RgnHandle;
18  876 --          savedOrigin:        Point;
18  877 --          savedLongOffset:    VPoint;
18  878 --          displaying:         BOOLEAN;
18  879 --
18  880 --          {-------------------------------+
18  881 --          |   DrawSubView                 |
18  882 --          +-------------------------------}
18  883 -- B        PROCEDURE DrawSubView (theSubView: TView);
18  884 --          VAR
18  885 --              itsFrame:       VRect;
18  886 --              itsQDFrame:     Rect;
18  887 0- B        BEGIN
18  888 --              IF theSubView.IsShown THEN
18  889 1-                BEGIN
18  890 --                  SetOrigin(savedOrigin.h, savedOrigin.v);    { Restore superview's focus.        }
```

```
18  891 --                  SetClip(savedClip);
18  892 --                  gLongOffset := savedLongOffset;
18  893 --                  gFocusedView := SELF;
18  894 --
18  895 --                  theSubView.GetFrame(itsFrame);              { See if this subview is visible.   }
18  896 --                  OffsetVRect(itsFrame, -longOffset.h, -longOffset.v);
18  897 --                  VRectToRect(itsFrame, itsQDFrame);
18  898 --                  IF (NOT displaying) | RectInRgn(itsQDFrame, thePort^.visRgn) THEN
18  899 --                      theSubView.DrawContents;
18  900 -1                  END;
18  901 -0 B        END;
18  902 --
18  903 0- A    BEGIN
18  904 --          IF Focus THEN
18  905 1-              BEGIN
18  906 --              GetVisibleRect(visRect);
18  907 --              displaying := NOT (gPrinting OR gDrawingPictScrap);
18  908 --              IF (NOT displaying) | SectRect(visRect, thePort^.visRgn^^.rgnbBox, visRect) THEN
18  909 2-                  BEGIN
18  910 --                  Draw(visRect);
18  911 --
18  912 --                  IF displaying THEN
18  913 3-                      BEGIN
18  914 --                      DoHighlightSelection(hlOff, fHLDesired);
18  915 --                      IF fPrintHandler <> NIL THEN
18  916 --                          DoDrawPrintFeedback(visRect);
18  917 -3                      END;
18  918 --
18  919 --                  IF CountSubViews > 0 THEN
18  920 3-                      BEGIN
18  921 --                      longOffset := gLongOffset;
18  922 --                      savedClip := MakeNewRgn;                { Save the superview's focus.       }
18  923 --                      savedOrigin := thePort^.portRect.topLeft;
18  924 --                      savedLongOffset := gLongOffset;
18  925 --                      GetClip(savedClip);
18  926 --                      EachSubView(DrawSubView);
18  927 --                      DisposeRgn(savedClip);
18  928 -3                      END;
18  929 -2                  END;
18  930 -1              END;
18  931 -0 A    END {TView.DrawContents};
18  932 --
18  933 --      {-----------------------------------+
18  934 --      |   EachSubView                     |
18  935 --      +-----------------------------------}
18  936 --      {$S MARes}
18  937 -- A    PROCEDURE TView.EachSubView (PROCEDURE DoToSubView (theSubView: TView));
18  938 0- A    BEGIN
18  939 --          IF fSubViews <> NIL THEN
18  940 --              fSubViews.Each(DoToSubView);
18  941 -0 A    END {TView.EachSubView};
18  942 --
18  943 --      {-----------------------------------+
18  944 --      |   FindSubView                     |
18  945 --      +-----------------------------------}
18  946 --      {$S MARes}
18  947 -- A    FUNCTION  TView.FindSubView (itsIdentifier: IDType): TView;
18  948 --      VAR
18  949 --          foundView:      TView;
18  950 --          dummyView:      TView;
18  951 --
18  952 --          {-------------------------------+
18  953 --          |   TestSubID                   |
18  954 --          +-------------------------------}
18  955 -- B        FUNCTION  TestSubID (theSubView: TView): BOOLEAN;
18  956 --
18  957 0- B        BEGIN
18  958 --              IF LONGINT(theSubView.fIdentifier) = LONGINT(itsIdentifier) THEN
18  959 --                  foundView := theSubView
18  960 --              ELSE
18  961 --                  dummyView := theSubView.FirstSubViewThat(TestSubID);
18  962 --              TestSubID := foundView <> NIL;
18  963 -0 B        END;
18  964 --
18  965 0- A    BEGIN
18  966 --          IF LONGINT(fIdentifier) = LONGINT(itsIdentifier) THEN
18  967 --              foundView := SELF
18  968 --          ELSE
18  969 1-              BEGIN
18  970 --              foundView := NIL;
18  971 --              dummyView := FirstSubViewThat(TestSubID);
18  972 -1              END;
18  973 --      {$IFC qDebug}
18  974 --          IF foundView = NIL THEN
18  975 1-              BEGIN
18  976 --              WRITELN('Unable to find subview: ''', itsIdentifier, '''');
18  977 --              IF gIntenseDebugging THEN
18  978 --                  ProgramBreak('!');
18  979 -1              END;
18  980 --      {$ENDC}
18  981 --              FindSubView := foundView;
18  982 -0 A    END;
18  983 --
18  984 --      {-----------------------------------+
18  985 --      |   FirstSubViewThat                |
18  986 --      +-----------------------------------}
18  987 --      {$S MARes}
18  988 -- A    FUNCTION  TView.FirstSubViewThat (FUNCTION TestSubView (theSubView: TView): BOOLEAN): TView;
18  989 0- A    BEGIN
```

```
18  990 --          IF fSubViews <> NIL THEN
18  991 --              FirstSubViewThat := TView(fSubViews.FirstThat(TestSubView))
18  992 --          ELSE
18  993 --              FirstSubViewThat := NIL;
18  994 -0 A    END {TView.FirstSubViewThat};
18  995 --
18  996 --      {-----------------------------------+
18  997 --      |   Focus                           |
18  998 --      +-----------------------------------}
18  999 --      {$S MARes}
18 1000 -- A    FUNCTION  TView.Focus: BOOLEAN;
18 1001 --      VAR
18 1002 --          relOrigin:      VPoint;
18 1003 --          qdFrame:        Rect;
18 1004 --          aRect:          Rect;
18 1005 --          viewRect:       VRect;
18 1006 --          visRect:        VRect;
18 1007 --          myFrame:        VRect;
18 1008 --          vhs:            VHSelect;
18 1009 --
18 1010 0- A    BEGIN
18 1011 --          Focus := TRUE;
18 1012 --
18 1013 --          IF gFocusedView = SELF THEN
18 1014 --              EXIT(Focus);
18 1015 --
18 1016 --          IF (gCurrPrintHandler <> NIL) & (gCurrPrintHandler.fView = SELF) THEN
18 1017 1-              BEGIN
18 1018 --              gCurrPrintHandler.FocusOnInterior;
18 1019 --              gFocusedView := SELF;
18 1020 --              EXIT(Focus);
18 1021 -1              END;
18 1022 --
18 1023 --          IF fShown & FocusOnSuperView THEN
18 1024 1-              BEGIN
18 1025 --              aRect := thePort^.clipRgn^^.rgnBBox;
18 1026 --              QDToViewRect(aRect, viewRect);
18 1027 --              {$H-}
18 1028 --              GetFrame(myFrame);
18 1029 --              IF SectVRect(myFrame, viewRect, visRect) THEN
18 1030 --              {$H+}
18 1031 2-                  BEGIN
18 1032 --                  ViewToQDRect(myFrame, qdFrame); {Must be done before changing gLongOffset}
18 1033 --                  FOR vhs := v TO h DO
18 1034 --                      IF fSize.vh[vhs] > kMaxCoord THEN
18 1035 3-                          BEGIN
18 1036 --                          relOrigin.vh[vhs] := viewRect.topLeft.vh[vhs] - visRect.topLeft.vh[vhs];
18 1037 --                          gLongOffset.vh[vhs] := gLongOffset.vh[vhs] - fLocation.vh[vhs];
18 1038 -3                          END
18 1039 --                      ELSE
18 1040 3-                          BEGIN
18 1041 --                          relOrigin.vh[vhs] := -fLocation.vh[vhs] + gLongOffset.vh[vhs];
18 1042 --                          gLongOffset.vh[vhs] := 0;
18 1043 -3                          END;
18 1044 --
18 1045 --                  SetOrigin(thePort^.portRect.left + relOrigin.h,
18 1046 --                            thePort^.portRect.top + relOrigin.v);
18 1047 --                  ClipFurtherTo(qdFrame, relOrigin.h, relOrigin.v);
18 1048 --
18 1049 --                  gFocusedView := SELF;
18 1050 --                  EXIT(Focus);
18 1051 -2                  END;
18 1052 -1              END;
18 1053 --
18 1054 --          { If we get this far, then we can't focus }
18 1055 --          ClipRect(gZeroRect);
18 1056 --          gFocusedView := NIL;
18 1057 --          Focus := FALSE;
18 1058 -0 A    END {TView.Focus};
18 1059 --
18 1060 --      {-----------------------------------+
18 1061 --      |   FocusOnSuperView                |
18 1062 --      +-----------------------------------}
18 1063 --      {$S MARes}
18 1064 -- A    FUNCTION  TView.FocusOnSuperView: BOOLEAN;
18 1065 0- A    BEGIN
18 1066 --          IF fSuperView <> NIL THEN
18 1067 --              FocusOnSuperView := fSuperView.Focus
18 1068 --          ELSE
18 1069 --              FocusOnSuperView := FALSE;
18 1070 -0 A    END {TView.FocusOnSuperView};
18 1071 --
18 1072 --      {-----------------------------------+
18 1073 --      |   ForceRedraw                     |
18 1074 --      +-----------------------------------}
18 1075 --      {$S MARes}
18 1076 -- A    PROCEDURE TView.ForceRedraw;
18 1077 --      VAR
18 1078 --          extent:     VRect;
18 1079 --
18 1080 0- A    BEGIN
18 1081 --          GetExtent(extent);
18 1082 --          InvalidVRect(extent);
18 1083 -0 A    END {TView.ForceRedraw};
18 1084 --
18 1085 --      {-----------------------------------+
18 1086 --      |   Free                            |
18 1087 --      +-----------------------------------}
18 1088 --      {$S MAClipboard}
```

```
18 1089 -- A   PROCEDURE TView.FreeFromClipboard;
18 1090 --
18 1091 0- A   BEGIN
18 1092 --         IF fDocument <> NIL THEN
18 1093 --             fDocument.FreeFromClipboard
18 1094 --         ELSE
18 1095 --             Free;
18 1096 -0 A   END {TView.FreeFromClipboard};
18 1097 --
18 1098 --     {$IFC qInspector}
18 1099 --     {-----------------------------------+
18 1100 --     |   GetInspectorName                |
18 1101 --     +-----------------------------------}
18 1102 --     {$S MAInspector}
18 1103 -- A   PROCEDURE TView.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
18 1104 --     VAR
18 1105 --         itsWindow:      TWindow;
18 1106 --
18 1107 0- A   BEGIN
18 1108 --         IF IsObject(fSuperView) THEN
18 1109 1-             BEGIN
18 1110 --             itsWindow := GetWindow;
18 1111 --             IF IsObject(itsWindow) & (NOT ODD(ORD4(itsWindow.fWMgrWindow))) THEN
18 1112 2-                 BEGIN
18 1113 --                 GetWTitle(itsWindow.fWMgrWindow, inspectorName);
18 1114 --                 inspectorName := CONCAT('In ', inspectorName);
18 1115 -2                 END;
18 1116 -1             END;
18 1117 -0 A   END {TView.GetInspectorName};
18 1118 --     {$ENDC}
18 1119 --
18 1120 --     {-----------------------------------+
18 1121 --     |   GetDialogView                   |   Actually, returns a TDialogView…
18 1122 --     +-----------------------------------}
18 1123 --     {$S MARes}
18 1124 -- A   FUNCTION  TView.GetDialogView: TView;
18 1125 0- A   BEGIN
18 1126 --         IF fSuperView <> NIL THEN
18 1127 --             GetDialogView := fSuperView.GetDialogView
18 1128 --         ELSE
18 1129 --             GetDialogView := NIL;
18 1130 -0 A   END {TView.GetDialogView};
18 1131 --
18 1132 --     {-----------------------------------+
18 1133 --     |   GetExtent                       |
18 1134 --     +-----------------------------------}
18 1135 --     {$S MARes}
18 1136 -- A   PROCEDURE TView.GetExtent (VAR itsExtent: VRect);
18 1137 0- A   BEGIN
18 1138 --         itsExtent.topLeft := gZeroVPt;
18 1139 --         itsExtent.botRight := fSize;
18 1140 -0 A   END {TView.GetExtent};
18 1141 --
18 1142 --     {-----------------------------------+
18 1143 --     |   GetFrame                        |
18 1144 --     +-----------------------------------}
18 1145 --     {$S MARes}
18 1146 -- A   PROCEDURE TView.GetFrame (VAR itsFrame: VRect);
18 1147 0- A   BEGIN
18 1148 --         itsFrame.topLeft := fLocation;
18 1149 --         itsFrame.right := itsFrame.left + fSize.h;
18 1150 --         itsFrame.bottom := itsFrame.top + fSize.v;
18 1151 -0 A   END {TView.GetFrame};
18 1152 --
18 1153 --     {-----------------------------------+
18 1154 --     |   GetGrafPort                     |
18 1155 --     +-----------------------------------}
18 1156 --     {$S MARes}
18 1157 -- A   FUNCTION  TView.GetGrafPort: GrafPtr;
18 1158 0- A   BEGIN
18 1159 --         IF gPrinting THEN
18 1160 --             GetGrafPort := thePort      {thePort assumed to be set by print handler}
18 1161 --         ELSE IF fSuperView <> NIL THEN
18 1162 --             GetGrafPort := fSuperView.GetGrafPort
18 1163 --         ELSE
18 1164 --             GetGrafPort := NIL;
18 1165 -0 A   END {TView.GetGrafPort};
18 1166 --
18 1167 --     {-----------------------------------+
18 1168 --     |   GetPrintExtent                  |
18 1169 --     +-----------------------------------}
18 1170 --     {$S MANonRes}
18 1171 -- A   PROCEDURE TView.GetPrintExtent (VAR printExtent: VRect);
18 1172 0- A   BEGIN
18 1173 --         GetExtent(printExtent);
18 1174 -0 A   END {TView.GetPrintExtent};
18 1175 --
18 1176 --     {-----------------------------------+
18 1177 --     |   GetQDExtent                     |
18 1178 --     +-----------------------------------}
18 1179 --     {$S MARes}
18 1180 -- A   PROCEDURE TView.GetQDExtent (VAR qdExtent: Rect);
18 1181 0- A   BEGIN
18 1182 --         qdExtent.topLeft := ViewToQDPt(gZeroVPt);
18 1183 --         qdExtent.botRight := ViewToQDPt(fSize);
18 1184 -0 A   END {TView.GetQDExtent};
18 1185 --
18 1186 --     {-----------------------------------+
18 1187 --     |   GetScroller                     |
```

```
18 1188 --      +-----------------------------------}
18 1189 --      {$S MARes}
18 1190 -- A    FUNCTION TView.GetScroller (immediateSuperView: BOOLEAN): TScroller;
18 1191 --
18 1192 --      VAR
18 1193 --          aScroller:      TScroller;
18 1194 --
18 1195 0- A    BEGIN
18 1196 --          GetScroller := NIL;
18 1197 --          IF fSuperView <> NIL THEN
18 1198 1-              BEGIN
18 1199 --              aScroller := fSuperView.GetScroller(immediateSuperView);
18 1200 --              IF (aScroller = fSuperView) | (NOT immediateSuperView) THEN
18 1201 --                  GetScroller := aScroller;
18 1202 -1              END;
18 1203 -0 A    END {TView.GetScroller};
18 1204 --
18 1205 --      {-----------------------------------+
18 1206 --      |   GetVisibleRect                  |
18 1207 --      +-----------------------------------}
18 1208 --      {$S MARes}
18 1209 -- A    PROCEDURE TView.GetVisibleRect (VAR visQDRect: Rect);
18 1210 --
18 1211 --      VAR
18 1212 --          extent:     VRect;
18 1213 --
18 1214 0- A    BEGIN
18 1215 --          {!!! Need to make GetQDExtent work for scrollers}
18 1216 --          GetExtent(extent);
18 1217 --          ViewToQDRect(extent, visQDRect);
18 1218 --          VisibleRect(visQDRect);
18 1219 -0 A    END {TView.GetVisibleRect};
18 1220 --
18 1221 --      {-----------------------------------+
18 1222 --      |   GetWindow                       |
18 1223 --      +-----------------------------------}
18 1224 --      {$S MARes}
18 1225 -- A    FUNCTION  TView.GetWindow: TWindow;
18 1226 0- A    BEGIN
18 1227 --          IF fSuperView <> NIL THEN
18 1228 --              GetWindow := fSuperView.GetWindow
18 1229 --          ELSE
18 1230 --              GetWindow := NIL;
18 1231 -0 A    END {TView.GetWindow};
18 1232 --
18 1233 --      {-----------------------------------+
18 1234 --      |   GivePasteData                   |
18 1235 --      +-----------------------------------}
18 1236 --      {$S MAClipboard}
18 1237 -- A    FUNCTION  TView.GivePasteData (aDataHandle: Handle; dataType: ResType): LONGINT;
18 1238 --      VAR
18 1239 --          offset:     LONGINT;
18 1240 --          err:        LONGINT;
18 1241 --          savedPerm:  BOOLEAN;
18 1242 --
18 1243 0- A    BEGIN
18 1244 --          {Make sure the scrap handle is grown out of permanent memory.  If it
18 1245 --              was grown out of temporary memory it would eat away at our
18 1246 --              code space.}
18 1247 --          savedPerm := PermAllocation(TRUE);
18 1248 --          err := GetScrap(aDataHandle, dataType, offset);
18 1249 --          savedPerm := PermAllocation(savedPerm);
18 1250 --
18 1251 --      {$IFC qDebug}
18 1252 --          IF err < 0 THEN
18 1253 --              WriteLn('TView.GivePasteData got error code # ', err:1, ' from GetScrap for type "',
18 1254 --                  dataType, '"');
18 1255 --      {$ENDC}
18 1256 --          GivePasteData := err;
18 1257 -0 A    END {TView.GivePasteData};
18 1258 --
18 1259 --      {-----------------------------------+
18 1260 --      |   HandleCursor                    |
18 1261 --      +-----------------------------------}
18 1262 --      {$S MARes}
18 1263 -- A    FUNCTION TView.HandleCursor (theMouse: VPoint; cursorRgn: RgnHandle): TView;
18 1264 --
18 1265 --      VAR
18 1266 --          dummyView,
18 1267 --          viewThatHandledCursor:  TView;
18 1268 --
18 1269 -- B        FUNCTION  TestMouse (theSubView: TView): BOOLEAN;
18 1270 --          VAR
18 1271 --              subViewPt:      VPoint;
18 1272 --
18 1273 0- B        BEGIN
18 1274 --              subViewPt := theMouse;
18 1275 --              theSubView.SuperToLocal(subViewPt);
18 1276 --              IF theSubView.ContainsMouse(subViewPt) THEN
18 1277 --                  viewThatHandledCursor := theSubView.HandleCursor(subViewPt, cursorRgn);
18 1278 --              TestMouse := viewThatHandledCursor <> NIL;
18 1279 -0 B        END {TestMouse};
18 1280 --
18 1281 0- A    BEGIN
18 1282 --          {Assume if we got here then we already know the mouse is in this view}
18 1283 --          viewThatHandledCursor := NIL;
18 1284 --          dummyView := LastSubViewThat(TestMouse);
18 1285 --
18 1286 --          IF viewThatHandledCursor = NIL THEN
```

```
18 1287 --              IF IsViewEnabled & Focus & DoSetCursor(ViewToQDPt(theMouse), cursorRgn) THEN
18 1288 --                  viewThatHandledCursor := SELF;
18 1289 --
18 1290 --          HandleCursor := viewThatHandledCursor;
18 1291 -0 A    END {TView.HandleCursor};
18 1292 --
18 1293 --      {-----------------------------------+
18 1294 --      |   HandleMouseDown                 |
18 1295 --      +-----------------------------------}
18 1296 --      {$S MASelCommand}
18 1297 -- A    FUNCTION  TView.HandleMouseDown (theMouse:          VPoint;
18 1298 --                                       VAR info:         EventInfo;
18 1299 --                                       VAR hysteresis:   Point;
18 1300 --                                       VAR theCommand:   TCommand): BOOLEAN;
18 1301 --      VAR
18 1302 --          viewThatHandledMouse:   TView;
18 1303 --
18 1304 --          {-----------------------------------+
18 1305 --          |   TestMouse                       |
18 1306 --          +-----------------------------------}
18 1307 -- B       FUNCTION  TestMouse (theSubView: TView): BOOLEAN;
18 1308 --          VAR
18 1309 --              subViewPt:      VPoint;
18 1310 --
18 1311 0- B       BEGIN
18 1312 --              subViewPt := theMouse;
18 1313 --              theSubView.SuperToLocal(subViewPt);
18 1314 --              IF theSubView.ContainsMouse(subViewPt) THEN
18 1315 --                  TestMouse := theSubView.HandleMouseDown(subViewPt,
18 1316 --                                                          info, hysteresis, theCommand)
18 1317 --              ELSE
18 1318 --                  TestMouse := FALSE;
18 1319 -0 B       END {TestMouse};
18 1320 --
18 1321 0- A    BEGIN
18 1322 --          HandleMouseDown := FALSE;
18 1323 --          theCommand := gNoChanges;
18 1324 --
18 1325 --          {Assume if we got here then we already know the mouse is in this view}
18 1326 --
18 1327 --          viewThatHandledMouse := LastSubViewThat(TestMouse);
18 1328 --
18 1329 --          IF viewThatHandledMouse <> NIL THEN     {a subview handled it}
18 1330 --              HandleMouseDown := TRUE
18 1331 --          ELSE IF IsViewEnabled & Focus THEN      {see if we can handle it}
18 1332 1-              BEGIN
18 1333 --              theCommand := DoMouseCommand(ViewToQDPt(theMouse), info, hysteresis);
18 1334 --              HandleMouseDown := TRUE;
18 1335 -1              END;
18 1336 -0 A    END {TView.HandleMouseDown};
18 1337 --
18 1338 --      {-----------------------------------+
18 1339 --      |   InvalidRect                     |
18 1340 --      +-----------------------------------}
18 1341 --      {$S MARes}
18 1342 -- A    PROCEDURE TView.InvalidRect (r: Rect);
18 1343 0- A    BEGIN
18 1344 --          IF Focus THEN
18 1345 1-              BEGIN
18 1346 --              VisibleRect(r);
18 1347 --              InvalRect(r);
18 1348 -1              END;
18 1349 -0 A    END {TView.InvalidRect};
18 1350 --
18 1351 --      {-----------------------------------+
18 1352 --      |   InvalidVRect                    |
18 1353 --      +-----------------------------------}
18 1354 --      {$S MARes}
18 1355 -- A    PROCEDURE TView.InvalidVRect (viewRect: VRect);
18 1356 --      VAR
18 1357 --          r:      Rect;
18 1358 --
18 1359 0- A    BEGIN
18 1360 --          IF Focus THEN
18 1361 1-              BEGIN
18 1362 --              ViewToQDRect(viewRect, r);
18 1363 --              VisibleRect(r);
18 1364 --              InvalRect(r);
18 1365 -1              END;
18 1366 -0 A    END {TView.InvalidVRect};
18 1367 --
18 1368 --      {-----------------------------------+
18 1369 --      |   IsShown                         |
18 1370 --      +-----------------------------------}
18 1371 --      {$S MARes}
18 1372 -- A    FUNCTION  TView.IsShown: BOOLEAN;
18 1373 0- A    BEGIN
18 1374 --          IsShown := fShown;
18 1375 -0 A    END;
18 1376 --
18 1377 --      {-----------------------------------+
18 1378 --      |   IsViewEnabled                   |
18 1379 --      +-----------------------------------}
18 1380 --      {$S MARes}
18 1381 -- A    FUNCTION  TView.IsViewEnabled: BOOLEAN;
18 1382 0- A    BEGIN
18 1383 --          IsViewEnabled := fViewEnabled;
18 1384 -0 A    END;
18 1385 --
```

```
18 1386 --      {-----------------------------------+
18 1387 --      |   LastSubViewThat                 |
18 1388 --      +-----------------------------------}
18 1389 --      {$S MARes}
18 1390 -- A    FUNCTION  TView.LastSubViewThat (FUNCTION TestSubView (theSubView: TView): BOOLEAN): TView;
18 1391 0- A    BEGIN
18 1392 --          IF fSubViews <> NIL THEN
18 1393 --              LastSubViewThat := TView(fSubViews.LastThat(TestSubView))
18 1394 --          ELSE
18 1395 --              LastSubViewThat := NIL;
18 1396 -0 A    END {TView.LastSubViewThat};
18 1397 --
18 1398 --      {-----------------------------------+
18 1399 --      |   LocalToSuper                    |
18 1400 --      +-----------------------------------}
18 1401 --      {$S MARes}
18 1402 -- A    PROCEDURE TView.LocalToSuper (VAR thePoint: VPoint);
18 1403 0- A    BEGIN
18 1404 --          AddVPt(fLocation, thePoint);
18 1405 -0 A    END {TView.LocalToSuper};
18 1406 --
18 1407 --      {-----------------------------------+
18 1408 --      |   LocalToWindow                   |
18 1409 --      +-----------------------------------}
18 1410 --      {$S MARes}
18 1411 -- A    PROCEDURE TView.LocalToWindow (VAR thePoint: VPoint);
18 1412 --
18 1413 --      VAR
18 1414 --          aView:      TView;
18 1415 --
18 1416 0- A    BEGIN
18 1417 --          aView := SELF;
18 1418 --          WHILE aView <> NIL DO
18 1419 1-              BEGIN
18 1420 --              AddVPt(aView.fLocation, thePoint);
18 1421 --              aView := aView.fSuperView;
18 1422 -1              END;
18 1423 -0 A    END {TView.LocalToWindow};
18 1424 --
18 1425 --      {-----------------------------------+
18 1426 --      |   Locate                          |
18 1427 --      +-----------------------------------}
18 1428 --      {$S MANonRes}
18 1429 -- A    PROCEDURE TView.Locate (h, v: VCoordinate; invalidate: BOOLEAN);
18 1430 --
18 1431 --          {-----------------------------+
18 1432 --          |   NotifySubView             |
18 1433 --          +-----------------------------}
18 1434 -- B        PROCEDURE NotifySubView (theSubView: TView);
18 1435 0- B        BEGIN
18 1436 --              theSubView.SuperViewMoved(invalidate);
18 1437 -0 B        END;
18 1438 --
18 1439 0- A    BEGIN
18 1440 --          IF (h <> fLocation.h) OR (v <> fLocation.v) THEN
18 1441 1-              BEGIN
18 1442 --              IF invalidate THEN
18 1443 --                  ForceRedraw;
18 1444 --              fLocation.h := h;
18 1445 --              fLocation.v := v;
18 1446 --              gFocusedView := NIL;    {Must re-focus because view moved.}
18 1447 --              IF invalidate THEN
18 1448 --                  ForceRedraw;
18 1449 --
18 1450 --              IF fSuperView <> NIL THEN
18 1451 --                  fSuperView.SubViewMoved(SELF);
18 1452 --              EachSubView(NotifySubView);
18 1453 -1              END;
18 1454 -0 A    END {TView.Locate};
18 1455 --
18 1456 --      {-----------------------------------+
18 1457 --      |   MakeFirstSubView                |
18 1458 --      +-----------------------------------}
18 1459 --      {$S MANonRes}
18 1460 -- A    PROCEDURE TView.MakeFirstSubView (theSubView: TView);
18 1461 0- A    BEGIN
18 1462 --          IF fSubViews <> NIL THEN                {there must be at least one subview}
18 1463 --              IF TView(fSubViews.First) <> theSubView THEN
18 1464 1-                  BEGIN
18 1465 --                  fSubViews.Delete(theSubView);
18 1466 --                  fSubViews.InsertFirst(theSubView);
18 1467 --                  theSubView.ForceRedraw;
18 1468 -1                  END;
18 1469 -0 A    END {TView.MakeFirstSubView};
18 1470 --
18 1471 --      {-----------------------------------+
18 1472 --      |   MakeLastSubView                 |
18 1473 --      +-----------------------------------}
18 1474 --      {$S MANonRes}
18 1475 -- A    PROCEDURE TView.MakeLastSubView (theSubView: TView);
18 1476 0- A    BEGIN
18 1477 --          IF fSubViews <> NIL THEN                {there must be at least one subview}
18 1478 --              IF TView(fSubViews.Last) <> theSubView THEN
18 1479 1-                  BEGIN
18 1480 --                  fSubViews.Delete(theSubView);
18 1481 --                  fSubViews.InsertLast(theSubView);
18 1482 --                  theSubView.ForceRedraw;
18 1483 -1                  END;
18 1484 -0 A    END {TView.MakeLastSubView};
```

```
18 1485 --
18 1486 --      {-----------------------------------+
18 1487 --      |   Open                            |
18 1488 --      +-----------------------------------}
18 1489 --      {$S MAOpen}
18 1490 -- A    PROCEDURE TView.Open;
18 1491 --
18 1492 --          {-------------------------------+
18 1493 --          |   OpenSubView                 |
18 1494 --          +-------------------------------}
18 1495 -- B        PROCEDURE OpenSubView (theSubView: TView);
18 1496 0- B        BEGIN
18 1497 --              theSubView.Open;
18 1498 -0 B        END;
18 1499 --
18 1500 0- A    BEGIN
18 1501 --          EachSubView(OpenSubView);
18 1502 -0 A    END {TView.Open};
18 1503 --
18 1504 --      {-----------------------------------+
18 1505 --      |   PageInteriorChanged             |
18 1506 --      +-----------------------------------}
18 1507 --      {$S MANonRes}
18 1508 -- A    PROCEDURE TView.PageInteriorChanged (newInterior: Rect);
18 1509 0- A    BEGIN
18 1510 -0 A    END {TView.PageInteriorChanged};
18 1511 --
18 1512 --      {-----------------------------------+
18 1513 --      |   QDToViewPt                      |
18 1514 --      +-----------------------------------}
18 1515 --      {$S MARes}
18 1516 -- A    PROCEDURE TView.QDToViewPt (qdPoint: Point; VAR viewPt: VPoint);
18 1517 0- A    BEGIN
18 1518 --          PtToVPt(qdPoint, viewPt);
18 1519 --          AddVPt(gLongOffset, viewPt);
18 1520 -0 A    END {TView.QDToViewPt};
18 1521 --
18 1522 --      {-----------------------------------+
18 1523 --      |   QDToViewRect                    |
18 1524 --      +-----------------------------------}
18 1525 --      {$S MARes}
18 1526 -- A    PROCEDURE TView.QDToViewRect (qdRect: Rect; VAR viewRect: VRect);
18 1527 0- A    BEGIN
18 1528 --          RectToVRect(qdRect, viewRect);
18 1529 --          OffsetVRect(viewRect, gLongOffset.h, gLongOffset.v);
18 1530 -0 A    END {TView.QDToViewRect};
18 1531 --
18 1532 --      {-----------------------------------+
18 1533 --      |   RemoveSubView                   |
18 1534 --      +-----------------------------------}
18 1535 --      {$S MANonRes}
18 1536 -- A    PROCEDURE TView.RemoveSubView (theSubView: TView);
18 1537 0- A    BEGIN
18 1538 --          IF fSubViews <> NIL THEN
18 1539 --              fSubViews.Delete(theSubView);
18 1540 --
18 1541 --          IF theSubView.fNextHandler = SELF THEN      { If the next event handler is the superview    }
18 1542 --              theSubView.fNextHandler := NIL;         { take it out of the event handler chain.       }
18 1543 --
18 1544 --          theSubView.fSuperView := NIL;
18 1545 -0 A    END {TView.RemoveSubView};
18 1546 --
18 1547 --      {-----------------------------------+
18 1548 --      |   Resize                          |
18 1549 --      +-----------------------------------}
18 1550 --      {$S MANonRes}
18 1551 -- A    PROCEDURE TView.Resize (width, height: VCoordinate; invalidate: BOOLEAN);
18 1552 --
18 1553 --      VAR
18 1554 --          delta:          VPoint;
18 1555 --          aRgn:           RgnHandle;
18 1556 --          myFrame:        VRect;
18 1557 --          r:              Rect;
18 1558 --
18 1559 --          {-------------------------------+
18 1560 --          |   NotifySubView               |
18 1561 --          +-------------------------------}
18 1562 -- B        PROCEDURE NotifySubView (theSubView: TView);
18 1563 0- B        BEGIN
18 1564 --              theSubView.SuperViewChangedSize(delta, invalidate);
18 1565 -0 B        END;
18 1566 --
18 1567 0- A    BEGIN
18 1568 --          IF (fSize.h <> width ) OR (fSize.v <> height) THEN
18 1569 1-              BEGIN
18 1570 --              SetVPt(delta, width - fSize.h, height - fSize.v);
18 1571 --              GetFrame(myFrame);                  { Get the old frame, before we change the size }
18 1572 --              fSize.h := width;
18 1573 --              fSize.v := height;
18 1574 --              gFocusedView := NIL;                {Must re-focus because size changed}
18 1575 --
18 1576 --              IF invalidate & FocusOnSuperView THEN
18 1577 2-                  BEGIN
18 1578 --                  {!!! Could be sped up by reusing a region rather than creating one}
18 1579 --                  aRgn := MakeNewRgn;
18 1580 --                  ViewToQDRect(myFrame, r);
18 1581 --                  VisibleRect(r);
18 1582 --                  RectRgn(aRgn, r);
18 1583 --
```

```
18 1584 --                  GetFrame(myFrame);              { Now get the new frame }
18 1585 --                  ViewToQDRect(myFrame, r);
18 1586 --                  VisibleRect(r);
18 1587 --      {$IFC qDebug}
18 1588 --                  UseTempRgn('TView.Resize');
18 1589 --      {$ENDC}
18 1590 --                  RectRgn(gTempRgn, r);
18 1591 --
18 1592 --                  XOrRgn(aRgn, gTempRgn, gTempRgn);
18 1593 --                  InvalRgn(gTempRgn);
18 1594 --      {$IFC qDebug}
18 1595 --                  DoneWithTempRgn;
18 1596 --      {$ENDC}
18 1597 --                  DisposeRgn(aRgn);
18 1598 -2                  END;
18 1599 --
18 1600 --              IF fSuperView <> NIL THEN
18 1601 --                  fSuperView.SubViewChangedSize(SELF, delta);
18 1602 --              EachSubView(NotifySubView);
18 1603 -1              END;
18 1604 -0 A    END {TView.Resize};
18 1605 --
18 1606 --      {-----------------------------------+
18 1607 --      |   RevealBottom                    |
18 1608 --      +-----------------------------------}
18 1609 --      {$S MAScroll}
18 1610 -- A    PROCEDURE TView.RevealBottom (redraw: BOOLEAN);
18 1611 --
18 1612 --      VAR
18 1613 --          rectToReveal:       VRect;
18 1614 --
18 1615 0- A    BEGIN
18 1616 --          rectToReveal.topLeft := fSize;
18 1617 --          rectToReveal.botRight := fSize;
18 1618 --          RevealRect(rectToReveal, Point(0), redraw);
18 1619 -0 A    END {TView.RevealBottom};
18 1620 --
18 1621 --      {-----------------------------------+
18 1622 --      |   RevealRect                      |
18 1623 --      +-----------------------------------}
18 1624 --      {$S MAScroll}
18 1625 -- A    PROCEDURE TView.RevealRect (rectToReveal: VRect; minToSee: Point;
18 1626 --                                  redraw: BOOLEAN);
18 1627 0- A    BEGIN
18 1628 --          IF fSuperView <> NIL THEN
18 1629 1-              BEGIN
18 1630 --              OffsetVRect(rectToReveal, fLocation.h, fLocation.v);
18 1631 --              fSuperView.RevealRect(rectToReveal, minToSee, redraw);
18 1632 -1              END;
18 1633 -0 A    END {TView.RevealRect};
18 1634 --
18 1635 --      {-----------------------------------+
18 1636 --      |   RevealTop                       |
18 1637 --      +-----------------------------------}
18 1638 --      {$S MAScroll}
18 1639 -- A    PROCEDURE TView.RevealTop (redraw: BOOLEAN);
18 1640 0- A    BEGIN
18 1641 --          RevealRect(gZeroVRect, Point(0), redraw);
18 1642 -0 A    END {TView.RevealTop};
18 1643 --
18 1644 --      {-----------------------------------+
18 1645 --      |   Show                            |
18 1646 --      +-----------------------------------}
18 1647 --      {$S MANonRes}
18 1648 -- A    PROCEDURE TView.Show (state, redraw: BOOLEAN);
18 1649 0- A    BEGIN
18 1650 --          redraw := redraw AND (state <> fShown);
18 1651 --          IF redraw THEN
18 1652 1-              BEGIN
18 1653 --              fShown := TRUE;                     { So that ForceRedraw works.    }
18 1654 --              ForceRedraw;
18 1655 -1              END;
18 1656 --          fShown := state;
18 1657 -0 A    END;
18 1658 --
18 1659 --      {-----------------------------------+
18 1660 --      |   ShowReverted                    |
18 1661 --      +-----------------------------------}
18 1662 --      {$S MAReadFile}
18 1663 -- A    PROCEDURE TView.ShowReverted;
18 1664 --
18 1665 --          {------------------------------+
18 1666 --          |   RevertSubView              |
18 1667 --          +------------------------------}
18 1668 -- B        PROCEDURE RevertSubView (theSubView: TView);
18 1669 0- B        BEGIN
18 1670 --              theSubView.ShowReverted;
18 1671 -0 B        END;
18 1672 --
18 1673 0- A    BEGIN
18 1674 --          AdjustSize;
18 1675 --          ForceRedraw;
18 1676 --          EachSubView(RevertSubView);
18 1677 -0 A    END {TView.ShowReverted};
18 1678 --
18 1679 --      {-----------------------------------+
18 1680 --      |   SubViewChangedSize              |
18 1681 --      +-----------------------------------}
18 1682 --      {$S MANonRes}
```

```
18 1683 -- A   PROCEDURE TView.SubViewChangedSize (theSubView: TView; delta: VPoint);
18 1684 0- A   BEGIN
18 1685 -0 A   END {TView.SubViewChangedSize};
18 1686 --
18 1687 --     {-----------------------------------+
18 1688 --     |   SubViewMoved                    |
18 1689 --     +-----------------------------------}
18 1690 --     {$S MANonRes}
18 1691 -- A   PROCEDURE TView.SubViewMoved (theSubView: TView);
18 1692 0- A   BEGIN
18 1693 -0 A   END {TView.SubViewMoved};
18 1694 --
18 1695 --     {-----------------------------------+
18 1696 --     |   SuperToLocal                    |
18 1697 --     +-----------------------------------}
18 1698 --     {$S MARes}
18 1699 -- A   PROCEDURE TView.SuperToLocal (VAR thePoint: VPoint);
18 1700 0- A   BEGIN
18 1701 --         SubVPt(fLocation, thePoint);
18 1702 -0 A   END {TView.SuperToLocal};
18 1703 --
18 1704 --     {-----------------------------------+
18 1705 --     |   SuperViewChangedSize            |
18 1706 --     +-----------------------------------}
18 1707 --     {$S MANonRes}
18 1708 -- A   PROCEDURE TView.SuperViewChangedSize (delta: VPoint; invalidate: BOOLEAN);
18 1709 --     VAR
18 1710 --         newSize:        VPoint;
18 1711 --         needsResizing:  BOOLEAN;
18 1712 --         vhs:            VHSelect;
18 1713 --
18 1714 0- A   BEGIN
18 1715 --         needsResizing := FALSE;
18 1716 --         newSize := fSize;
18 1717 --         FOR vhs := v TO h DO
18 1718 --             IF fSizeDeterminer[vhs] = sizeSuperView THEN
18 1719 --                 needsResizing := TRUE
18 1720 --             ELSE IF fSizeDeterminer[vhs] = sizeRelSuperView THEN
18 1721 1-                 BEGIN
18 1722 --                 newSize.vh[vhs] := newSize.vh[vhs] + delta.vh[vhs];
18 1723 --                 needsResizing := TRUE;
18 1724 -1                 END;
18 1725 --
18 1726 --         IF needsResizing THEN
18 1727 1-             BEGIN
18 1728 --             ComputeSize(newSize);
18 1729 --             Resize(newSize.h, newSize.v, invalidate);
18 1730 --             DoPagination;
18 1731 -1             END;
18 1732 -0 A   END {TView.SuperViewChangedSize};
18 1733 --
18 1734 --     {-----------------------------------+
18 1735 --     |   SuperViewMoved                  |
18 1736 --     +-----------------------------------}
18 1737 --     {$S MANonRes}
18 1738 -- A   PROCEDURE TView.SuperViewMoved (invalidate: BOOLEAN);
18 1739 0- A   BEGIN
18 1740 -0 A   END {TView.SuperViewMoved};
18 1741 --
18 1742 --     {-----------------------------------+
18 1743 --     |   Update                          |
18 1744 --     +-----------------------------------}
18 1745 --     {$S MARes}
18 1746 -- A   PROCEDURE TView.Update;
18 1747 --
18 1748 --     VAR
18 1749 --         itsWindow:      TWindow;
18 1750 --
18 1751 0- A   BEGIN
18 1752 --         itsWindow := GetWindow;
18 1753 --         IF itsWindow <> NIL THEN
18 1754 1-             BEGIN
18 1755 --             BeginUpdate(itsWindow.fWMgrWindow);
18 1756 --             DrawContents;
18 1757 --             EndUpdate(itsWindow.fWMgrWindow);
18 1758 -1             END;
18 1759 -0 A   END {TView.Update};
18 1760 --
18 1761 --     {-----------------------------------+
18 1762 --     |   ValidVRect                      |
18 1763 --     +-----------------------------------}
18 1764 --     {$S MARes}
18 1765 -- A   PROCEDURE TView.ValidVRect (viewRect: VRect);
18 1766 --     VAR
18 1767 --         r:      Rect;
18 1768 --
18 1769 0- A   BEGIN
18 1770 --         IF Focus THEN
18 1771 1-             BEGIN
18 1772 --             ViewToQDRect(viewRect, r);
18 1773 --             VisibleRect(r);
18 1774 --             ValidRect(r);
18 1775 -1             END;
18 1776 -0 A   END {TView.ValidVRect};
18 1777 --
18 1778 --     {-----------------------------------+
18 1779 --     |   ViewEnable                      |
18 1780 --     +-----------------------------------}
18 1781 --     {$S MARes}
```

```
18 1782 -- A   PROCEDURE TView.ViewEnable (state, redraw: BOOLEAN);
18 1783 0- A   BEGIN
18 1784 --         fViewEnabled := state;
18 1785 --         IF redraw THEN
18 1786 --             ForceRedraw;
18 1787 -0 A   END;
18 1788 --
18 1789 --     {-----------------------------------+
18 1790 --     |   ViewToQDPt                      |
18 1791 --     +-----------------------------------}
18 1792 --     {$S MARes}
18 1793 -- A   FUNCTION  TView.ViewToQDPt (viewPt: VPoint): Point;
18 1794 0- A   BEGIN
18 1795 --         SubVPt(gLongOffset, viewPt);
18 1796 --         ViewToQDPt := VPtToPt(viewPt);
18 1797 -0 A   END {TView.ViewToQDPt};
18 1798 --
18 1799 --     {-----------------------------------+
18 1800 --     |   ViewToQDRect                    |
18 1801 --     +-----------------------------------}
18 1802 --     {$S MARes}
18 1803 -- A   PROCEDURE TView.ViewToQDRect (viewRect: VRect; VAR qdRect: Rect);
18 1804 0- A   BEGIN
18 1805 --         OffsetVRect(viewRect, -gLongOffset.h, -gLongOffset.v);
18 1806 --         VRectToRect(viewRect, qdRect);
18 1807 -0 A   END {TView.ViewToQDRect};
18 1808 --
18 1809 --     {-----------------------------------+
18 1810 --     |   WindowToLocal                   |
18 1811 --     +-----------------------------------}
18 1812 --     {$S MARes}
18 1813 -- A   PROCEDURE TView.WindowToLocal (VAR thePoint: VPoint);
18 1814 --
18 1815 --     VAR
18 1816 --         aView:      TView;
18 1817 --
18 1818 0- A   BEGIN
18 1819 --         aView := SELF;
18 1820 --         WHILE aView <> NIL DO
18 1821 1-             BEGIN
18 1822 --             SubVPt(aView.fLocation, thePoint);
18 1823 --             aView := aView.fSuperView;
18 1824 -1             END;
18 1825 -0 A   END {TView.WindowToLocal};
18 1826 --
18 1827 --     {-----------------------------------+
18 1828 --     |   WriteToDeskScrap                |
18 1829 --     +-----------------------------------}
18 1830 --     {$S MANonRes}
18 1831 -- A   PROCEDURE TView.WriteToDeskScrap;
18 1832 --     VAR
18 1833 --         pHndl:      PicHandle;
18 1834 --         qdExtent:   Rect;
18 1835 --         err:        LONGINT;
18 1836 --         tempPort:   GrafPort;
18 1837 --         tempCPort:  CGrafPort;
18 1838 --
18 1839 0- A   BEGIN
18 1840 --         GetQDExtent(qdExtent);
18 1841 --
18 1842 --         IF gConfiguration.hasColorQD THEN
18 1843 --             OpenCPort(@tempCPort)
18 1844 --         ELSE
18 1845 --             OpenPort(@tempPort);
18 1846 --
18 1847 --         {Open color or black & white, depending on the port}
18 1848 --         pHndl := OpenPicture(qdExtent);
18 1849 --
18 1850 --         IF pHndl <> NIL THEN
18 1851 1-             BEGIN
18 1852 --
18 1853 --             gDrawingPictScrap := TRUE;
18 1854 --             ClipRect(qdExtent);
18 1855 --
18 1856 --             Draw(qdExtent);
18 1857 --
18 1858 --             gDrawingPictScrap := FALSE;
18 1859 --             ClosePicture;
18 1860 --
18 1861 --             {On the 128K ROMs the picFrame will be empty if drawing the
18 1862 --                 picture failed.  On the 64K ROM's QuickDraw simply bombs.}
18 1863 --             IF EmptyRect(pHndl^^.picFrame) THEN
18 1864 2-                 BEGIN
18 1865 --     {$IFC qDebug}
18 1866 --                 WRITELN('Picture frame is empty!');
18 1867 --     {$ENDC}
18 1868 --                 KillPicture(pHndl);
18 1869 --                 Failure(memFullErr, 0);
18 1870 -2                 END;
18 1871 --
18 1872 --             err := PutDeskScrapData('PICT', Handle(pHndl));
18 1873 --             KillPicture(pHndl);
18 1874 --
18 1875 --             IF err <> NoErr THEN
18 1876 2-                 BEGIN
18 1877 --     {$IFC qDebug}
18 1878 --                 ProgramBreak('Failed to put PICT-type scrap');
18 1879 --     {$ENDC}
18 1880 --                 Failure(err, 0);
```

```
18 1881 -2                END;
18 1882 -1            END
18 1883 --        ELSE
18 1884 1-                BEGIN
18 1885 --    {$IFC qDebug}
18 1886 --                ProgramBreak('Couldn''t OpenPicture during attempt to write PICT to desk scrap');
18 1887 --    {$ENDC}
18 1888 --                Failure(memFullErr, 0);  {Assume cause of failure was lack of memory}
18 1889 -1                END;
18 1890 --
18 1891 --        IF gConfiguration.hasColorQD THEN
18 1892 --            CloseCPort(@tempCPort)
18 1893 --        ELSE
18 1894 --            ClosePort(@tempPort);
18 1895 -0 A   END {TView.WriteToDeskScrap};
18 1896 --
18 1897 --    {$IFC qDebug}
18 1898 --        {-----------------------------------+
18 1899 --        |   Fields                          |
18 1900 --        +-----------------------------------}
18 1901 --        {$S MAFields}
18 1902 -- A      PROCEDURE TView.Fields (PROCEDURE DoToField (fieldName: Str255;
18 1903 --                                                    fieldAddr: Ptr;
18 1904 --                                                    fieldType: INTEGER)); OVERRIDE;
18 1905 0- A      BEGIN
18 1906 --            DoToField('TView', NIL, bClass);
18 1907 --            DoToField('fSuperView', @fSuperView, bObject);
18 1908 --            DoToField('fSubViews', @fSubViews, bObject);
18 1909 --            DoToField('fDocument', @fDocument, bObject);
18 1910 --            DoToField('fLocation', @fLocation, bVPoint);
18 1911 --            DoToField('fSize', @fSize, bVPoint);
18 1912 --            DoToField('fSizeDeter[h]', @fSizeDeterminer[h], bByte);
18 1913 --            DoToField('fSizeDeter[v]', @fSizeDeterminer[v], bByte);
18 1914 --            DoToField('fHLDesired', @fHLDesired, bByte);
18 1915 --            DoToField('fIdentifier', @fIdentifier, bOSType);
18 1916 --            DoToField('fShown', @fShown, bBoolean);
18 1917 --            DoToField('fViewEnabled', @fViewEnabled, bBoolean);
18 1918 --            DoToField('fPrintHandler', @fPrintHandler, bObject);
18 1919 --            INHERITED Fields(DoToField);
18 1920 -0 A       END {TView.Fields};
18 1921 --    {$ENDC}
13 3006 --    {$I UMacApp.TWindow.p}
```

```
19    1 --      {$P}
19    2 --      { UMacApp.TWindow.p }
19    3 --      { Copyright © 1987, 1988 by Apple Computer Inc.  All rights reserved. }
19    4 --
19    5 --
19    6 --      (*****************************************************************************************)
19    7 --      (*      T W i n d o w                                                                    *)
19    8 --      (*****************************************************************************************)
19    9 --
19   10 --      {$IFC qProceduralViews}
19   11 --      {$S MAOpen}
19   12 --      {-----------------------------------+
19   13 --      |   IWindow                         |
19   14 --      +-----------------------------------}
19   15 -- A    PROCEDURE TWindow.IWindow (itsDocument: TDocument; itsWmgrWindow: WindowPtr;
19   16 --                                 canResize, canClose, disposeOnFree: BOOLEAN);
19   17 --
19   18 --      VAR
19   19 --          preDocname:         INTEGER;
19   20 --          constTitle:         INTEGER;
19   21 --          aString:            Str255;
19   22 --          fi:                 FailInfo;
19   23 --
19   24 --          {-------------------------------+
19   25 --          |   HandleFailure               |
19   26 --          +-------------------------------}
19   27 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
19   28 0- B       BEGIN
19   29 --            Free;
19   30 -0 B       END;
19   31 --
19   32 0- A    BEGIN
19   33 --          { set up fields necessary to free myself }
19   34 --          fWmgrWindow := itsWmgrWindow;
19   35 --          fFreeOnClosing := FALSE;
19   36 --          fDisposeOnFree := disposeOnFree;
19   37 --          fDocument := NIL;
19   38 --
19   39 --          SetPort(itsWMgrWindow);     { ??? Why is this necessary? }
19   40 --          { Start with size of (0, 0).  Will be corrected in TWindow.Open }
19   41 --          IView(NIL, NIL, gZeroVPt, gZeroVPt, SizeVariable, SizeVariable);
19   42 --
19   43 --          fProcID := GetWRefCon(itsWmgrWindow);
19   44 --          SetWRefcon(itsWmgrWindow, LONGINT(SELF));
19   45 --
19   46 --          fAdapted := FALSE;                      { Set by AdaptToScreen }
19   47 --          fHorzCentered := FALSE;                 { Set by Center }
19   48 --          fVertCentered := FALSE;                 { Set by Center }
19   49 --          fStaggered := FALSE;                    { Set by SimpleStagger }
19   50 --          fForcedOnScreen := FALSE;               { Set by ForceOnScreen }
19   51 --          fIsActive := FALSE;
19   52 --          fIsResizable := canResize;
19   53 --          fIsClosable := canClose;
19   54 --          fTarget := SELF;
19   55 --          fTargetID := fIdentifier;
19   56 --          fClosesDocument := TRUE;
19   57 --          fOpenInitially := TRUE;
19   58 --          fIsModal := FALSE;
19   59 --          fDoFirstClick := FALSE;
19   60 --
19   61 --          CatchFailures(fi, HandleFailure);
19   62 --
19   63 --          fMoveBounds := gStdWMoveBounds;
19   64 --          SetResizeLimits(gStdWSizeRect.topLeft, gStdWSizeRect.botRight);
19   65 --          GetWTitle(itsWmgrWindow, aString);
19   66 --          IF ParseTitleTemplate(aString, preDocname, constTitle) THEN
19   67 --              SetWTitle(itsWmgrWindow, aString);
19   68 --          fPreDocname := preDocname;
19   69 --          fConstTitle := constTitle;
19   70 --
19   71 --          InstallDocument(itsDocument);
19   72 --
19   73 --          Success(fi);
19   74 -0 A    END { TWindow.IWindow };
19   75 --      {$ENDC}
19   76 --
19   77 --
19   78 --      {$IFC qTemplateViews}
19   79 --      {$S MAOpen}
19   80 --      {-----------------------------------+
19   81 --      |   IRes                            |
19   82 --      +-----------------------------------}
19   83 -- A    PROCEDURE TWindow.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
19   84 --
19   85 --      VAR
19   86 --          bounds:             Rect;
19   87 --          aWMgrWindow:        WindowPtr;
19   88 --          preDocname:         INTEGER;
19   89 --          constTitle:         INTEGER;
19   90 --          aString:            Str255;
19   91 --          fi:                 FailInfo;
19   92 --
19   93 --          {-------------------------------+
19   94 --          |   HandleFailure               |
19   95 --          +-------------------------------}
19   96 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
19   97 0- B       BEGIN
19   98 --            Free;
```

```
19   99 -0 B       END;
19  100 --
19  101 0- A   BEGIN
19  102 --         { set up fields necessary to free myself }
19  103 --         fDocument := NIL;
19  104 --         fWmgrWindow := NIL;
19  105 --         fFreeOnClosing := FALSE;
19  106 --         fDisposeOnFree := TRUE;
19  107 --
19  108 --         INHERITED IRes(NIL, itsSuperView, itsParams);
19  109 --
19  110 --         SetRect(bounds, fLocation.h, fLocation.v, fLocation.h + fSize.h, fLocation.v + fSize.v);
19  111 --         fLocation := gZeroVPt;
19  112 --
19  113 --         WITH WindowTemplatePtr(itsParams)^.DO
19  114 1-            BEGIN
19  115 --            CatchFailures(fi, HandleFailure);
19  116 --
19  117 --            fProcID := procID;
19  118 --
19  119 --            IF gConfiguration.hasColorQD THEN
19  120 --               aWMgrWindow := WindowPtr(NewCWindow(NIL, bounds, title, FALSE, procID, Pointer(-1),
19  121 --                                 .            hasGoAway, ORD4(SELF)))
19  122 --            ELSE
19  123 --               aWMgrWindow := NewWindow(NIL, bounds, title, FALSE, procID, Pointer(-1),
19  124 --                                          hasGoAway, ORD4(SELF));
19  125 --            fWMgrWindow := aWMgrWindow;
19  126 --
19  127 --            fIsActive := FALSE;
19  128 --            fIsResizable := resizable;
19  129 --            fIsClosable := hasGoAway;
19  130 --            fTarget := SELF;
19  131 --            fTargetID := targetID;
19  132 --            fIsModal := isModal;
19  133 --            fDoFirstClick := doFirstClick;
19  134 --            fFreeOnClosing := freeOnClosing;
19  135 --            fDisposeOnFree := disposeOnFree;
19  136 --            fClosesDocument := closesDocument;
19  137 --            fOpenInitially := openInitially;
19  138 --            fMoveBounds := gStdWMoveBounds;
19  139 --            SetResizeLimits(gStdWSizeRect.topLeft, gStdWSizeRect.botRight);
19  140 --
19  141 --            GetWTitle(aWMgrWindow, aString);
19  142 --            IF ParseTitleTemplate(aString, preDocname, constTitle) THEN
19  143 --               SetWTitle(aWMgrWindow, aString);
19  144 --            fPreDocname := preDocname;
19  145 --            fConstTitle := constTitle;
19  146 --
19  147 --            InstallDocument(itsDocument);
19  148 --
19  149 --            Success(fi);
19  150 --
19  151 --            IF mustAdaptToScreen THEN
19  152 --               AdaptToScreen;
19  153 --            IF horzCenter | vertCenter THEN
19  154 --               Center(horzCenter, vertCenter);
19  155 --            IF stagger THEN
19  156 --               SimpleStagger(kStdStaggerAmount, kStdStaggerAmount, gStdStaggerCount);
19  157 --            IF mustForceOnScreen THEN
19  158 --               ForceOnScreen;
19  159 --
19  160 --            OffsetPtrWStr(itsParams, SIZEOF(WindowTemplate));
19  161 -1            END;
19  162 -0 A   END { TWindow.IRes };
19  163 --     {$ENDC}
19  164 --
19  165 --
19  166 --     {$IFC qWriteTemplates}
19  167 --     {$S MAWriteRes}
19  168 --     {-----------------------------------+
19  169 --     |   WRes                            |
19  170 --     +-----------------------------------}
19  171 -- A   PROCEDURE TWindow.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
19  172 --
19  173 --     VAR
19  174 --         theTitle:   Str255;
19  175 --         wnPtr:      WindowTemplatePtr;
19  176 --
19  177 0- A   BEGIN
19  178 --         INHERITED WRes(theResource, itsParams);
19  179 --
19  180 --         GetWTitle(fWmgrWindow, theTitle);
19  181 --
19  182 --         wnPtr := WindowTemplatePtr(ExpandPtrWStr(theResource, itsParams,
19  183 --           SIZEOF(WindowTemplate), LENGTH(theTitle)));
19  184 --
19  185 --         WITH wnPtr^ DO
19  186 1-            BEGIN
19  187 --            procID := fProcID;
19  188 --            hasGoAway := fIsClosable;
19  189 --            resizable := fIsResizable;
19  190 --            isModal := fIsModal;
19  191 --            doFirstClick := fDoFirstClick;
19  192 --            freeOnClosing := fFreeOnClosing;
19  193 --            disposeOnFree := fDisposeOnFree;
19  194 --            closesDocument := fClosesDocument;
19  195 --            openInitially := fOpenInitially;
19  196 --            mustAdaptToScreen := fAdapted;
19  197 --            stagger := fStaggered;
```

```
19  198 --              mustForceOnScreen := fForcedOnScreen;
19  199 --              vertCenter := fVertCentered;
19  200 --              horzCenter := fHorzCentered;
19  201 --              targetID := fTargetID;
19  202 --              (* title := theTitle; *)
19  203 --              CopyStr255(theTitle, PRStr(title));
19  204 -1             END;
19  205 -0 A    END;
19  206 --      {$ENDC}
19  207 --
19  208 --
19  209 --      {$IFC qWriteTemplates}
19  210 --      {$S MAWriteRes}
19  211 --      {-----------------------------------+
19  212 --      |   WriteRes                        |
19  213 --      +-----------------------------------}
19  214 -- A    PROCEDURE TWindow.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
19  215 0- A    BEGIN
19  216 --          gWResSignature := 'wind';   gWResType := 'TWindow';
19  217 --          WRes(theResource, itsParams);
19  218 -0 A    END;
19  219 --      {$ENDC}
19  220 --
19  221 --
19  222 --      {$S MAClose}
19  223 --      {-----------------------------------+
19  224 --      |   Free                            |
19  225 --      +-----------------------------------}
19  226 -- A    PROCEDURE TWindow.Free; OVERRIDE;
19  227 --
19  228 --      VAR
19  229 --          disposeOnFree:  BOOLEAN;
19  230 --          wmgrWindow:     WindowPtr;
19  231 --
19  232 0- A    BEGIN
19  233 --          disposeOnFree := fDisposeOnFree;
19  234 --          wmgrWindow := fWmgrWindow;
19  235 --
19  236 --          IF fDocument <> NIL THEN
19  237 --              fDocument.DeleteWindow(SELF)
19  238 --          ELSE
19  239 --              gApplication.DeleteFreeWindow(SELF);
19  240 --
19  241 --          INHERITED Free;
19  242 --
19  243 --          { ****** DON'T REFER TO ANY FIELDS OR METHODS OF SELF ****** }
19  244 --
19  245 --          FreeWmgrWindow(wmgrWindow, disposeOnFree);
19  246 -0 A    END { TWindow.Free };
19  247 --
19  248 --
19  249 --      {$S MAActivate}
19  250 --      {-----------------------------------+
19  251 --      |   Activate                        |
19  252 --      +-----------------------------------}
19  253 -- A    PROCEDURE TWindow.Activate (entering: BOOLEAN);
19  254 --
19  255 --      VAR
19  256 --          different:  BOOLEAN;
19  257 --
19  258 0- A    BEGIN
19  259 --          gFocusedView := NIL;                { Must refocus to make sure thePort is set }
19  260 --          different := entering <> fIsActive;
19  261 --
19  262 --          IF different THEN
19  263 1-              BEGIN
19  264 --              Update;
19  265 --              INHERITED Activate(entering);
19  266 --
19  267 --              IF entering THEN
19  268 2-                  BEGIN
19  269 --                  IF gFrontWindow <> NIL THEN     { So lost deactivate doesn't
19  270 --                                                    screw up gTarget }
19  271 --                      gFrontWindow.Activate(FALSE);
19  272 --                  gFrontWindow := SELF;
19  273 --                  gApplication.SetTarget(fTarget);
19  274 --                  gDocument := fDocument;
19  275 -2                  END
19  276 --              ELSE
19  277 2-                  BEGIN
19  278 --                  fTarget := gTarget;
19  279 --                  gApplication.SetTarget(gApplication);
19  280 --                  gFrontWindow := NIL;
19  281 --                  gDocument := NIL;
19  282 --                  SetCursor(arrow);                   { ensure that the cursor is an arrow
19  283 --                                                        when MacApp loses control }
19  284 -2                  END;
19  285 -1              END;
19  286 --
19  287 --          IF fIsResizable THEN
19  288 --              DrawResizeIcon;
19  289 --
19  290 --          fIsActive := entering;
19  291 -0 A    END { TWindow.Activate };
19  292 --
19  293 --
19  294 --      {$S MAOpen}
19  295 --      {-----------------------------------+
19  296 --      |   AdaptToScreen                   |
```

```
19  297 --      +-----------------------------------}
19  298 -- A    PROCEDURE TWindow.AdaptToScreen;
19  299 --
19  300 --      CONST
19  301 --          stdHScreen =    512;
19  302 --          stdVScreen =    342;
19  303 --          stdScreen =     $10000 * stdVScreen + stdHScreen;
19  304 --
19  305 --      VAR
19  306 --          diff:       Point;
19  307 --          windRect:   Rect;
19  308 --          newSize:    Point;
19  309 --
19  310 0- A    BEGIN
19  311 --          fAdapted := TRUE;                       { We adapted to the screen }
19  312 --              { Compute difference between current screen and std Mac screen }
19  313 --          diff := screenBits.bounds.botRight;
19  314 --          SubPt(screenBits.bounds.topLeft, diff);
19  315 --          SubPt(Point(stdScreen), diff);
19  316 --
19  317 --          IF LONGINT(diff) <> 0 THEN { do an adjustment }
19  318 1-              BEGIN
19  319 --              GetGlobalBounds(windRect);
19  320 --              newSize := windRect.botRight;
19  321 --              SubPt(windRect.topLeft, newSize);
19  322 --              AddPt(diff, newSize);
19  323 --              newSize := Point(PinRect(fResizeLimits, newSize));
19  324 --
19  325 --              SizeWindow(fWMgrWindow, newSize.h, newSize.v, FALSE);
19  326 -1              END;
19  327 -0 A    END { TWindow.AdaptToScreen };
19  328 --
19  329 --
19  330 --      {$S MARes}
19  331 --      {-----------------------------------+
19  332 --      |   AllowsMenuAccess                |
19  333 --      +-----------------------------------}
19  334 -- A    FUNCTION  TWindow.AllowsMenuAccess: BOOLEAN;
19  335 --
19  336 0- A    BEGIN
19  337 --          { Default is to always allow menu access, even if modal }
19  338 --          AllowsMenuAccess := TRUE;
19  339 -0 A    END { TWindow.AllowsMenuAccess };
19  340 --
19  341 --
19  342 --      {$S MAOpen}
19  343 --      {-----------------------------------+
19  344 --      |   Center                          |
19  345 --      +-----------------------------------}
19  346 -- A    PROCEDURE TWindow.Center (horizontally, vertically: BOOLEAN);
19  347 --
19  348 --      VAR
19  349 --          screenSize:     Point;
19  350 --          globalbounds:   Rect;
19  351 --
19  352 0- A    BEGIN
19  353 --          fHorzCentered := horizontally;
19  354 --          fVertCentered := vertically;
19  355 --          IF fWMgrWindow <> NIL THEN
19  356 1-              BEGIN
19  357 --              screenSize := screenBits.bounds.botRight;
19  358 --              SubPt(screenBits.bounds.topLeft, screenSize);
19  359 --              GetGlobalBounds(globalbounds);
19  360 --              WITH globalBounds DO
19  361 2-                  BEGIN
19  362 --                  IF horizontally THEN
19  363 --                      left := (screenSize.h - LengthRect(globalBounds, h)) DIV 2;
19  364 --                  IF vertically THEN
19  365 --                      top := (screenSize.v - LengthRect(globalBounds, v)) DIV 2;
19  366 --                  MoveWindow(fWMgrWindow, left, top, FALSE);
19  367 -2                  END;
19  368 -1              END;
19  369 -0 A    END { TWindow.Center };
19  370 --
19  371 --
19  372 --      {$S MAClose}
19  373 --      {-----------------------------------+
19  374 --      |   Close                           |
19  375 --      +-----------------------------------}
19  376 -- A    PROCEDURE TWindow.Close; OVERRIDE;
19  377 --
19  378 0- A    BEGIN
19  379 --          INHERITED Close;
19  380 --
19  381 --          IF gFrontWindow = SELF THEN
19  382 --              gFrontWindow := NIL;
19  383 --
19  384 --          Show(FALSE, kRedraw);
19  385 --          Activate(FALSE);            { ??? Is this necessary? }
19  386 --
19  387 --          IF fFreeOnClosing THEN
19  388 --              Free;
19  389 -0 A    END { TWindow.Close };
19  390 --
19  391 --
19  392 --      {$S MAClose}
19  393 --      {-----------------------------------+
19  394 --      |   CloseByUser                     |
19  395 --      +-----------------------------------}
```

```
19  396 -- A   PROCEDURE TWindow.CloseByUser;
19  397 --
19  398 --      VAR
19  399 --          openDocWindows:     INTEGER;
19  400 --
19  401 --          {--------------------------------+
19  402 --          |   CountOpenWindows             |
19  403 --          +--------------------------------}
19  404 -- B       PROCEDURE CountOpenWindows (aWindow: TWindow);
19  405 --
19  406 0- B       BEGIN
19  407 --              IF IsShown THEN
19  408 --                  openDocWindows := openDocWindows + 1;
19  409 -0 B       END;
19  410 --
19  411 0- A   BEGIN
19  412 --          IF fDocument = NIL THEN { free window }
19  413 --              Close
19  414 --          ELSE
19  415 1-              BEGIN
19  416 --              { See how many open windows this document has }
19  417 --              openDocWindows := 0;
19  418 --              fDocument.ForAllWindowsDo(CountOpenWindows);
19  419 --
19  420 --              IF fClosesDocument OR (openDocWindows = 1) THEN
19  421 --                  fDocument.Close
19  422 --              ELSE
19  423 --                  Close;
19  424 -1              END;
19  425 -0 A   END { TWindow.CloseByUser };
19  426 --
19  427 --
19  428 --      {$S MARes}
19  429 --      {-----------------------------------+
19  430 --      |   DoSetupMenus                    |
19  431 --      +-----------------------------------}
19  432 -- A   PROCEDURE TWindow.DoSetupMenus; OVERRIDE;
19  433 --
19  434 0- A   BEGIN
19  435 --          IF NOT fIsModal THEN            { Don't enable menu/app commands if modal   }
19  436 --              INHERITED DoSetupMenus;
19  437 -0 A   END { TWindow.DoSetupMenus };
19  438 --
19  439 --
19  440 --      {$S MARes}
19  441 --      {-----------------------------------+
19  442 --      |   Draw                            |
19  443 --      +-----------------------------------}
19  444 -- A   PROCEDURE TWindow.Draw (area: Rect); OVERRIDE;
19  445 --
19  446 0- A   BEGIN
19  447 --          EraseRect(fWMgrWindow^.portRect);
19  448 -0 A   END { TWindow.Draw };
19  449 --
19  450 --
19  451 --      {$S MARes}
19  452 --      {-----------------------------------+
19  453 --      |   DrawContents                    |
19  454 --      +-----------------------------------}
19  455 -- A   PROCEDURE TWindow.DrawContents; OVERRIDE;
19  456 --
19  457 0- A   BEGIN
19  458 --          INHERITED DrawContents;
19  459 --          { Ensure that other views don't draw over the grow box by drawing it last }
19  460 --          IF fIsResizable AND Focus THEN
19  461 --              DrawResizeIcon;
19  462 -0 A   END { TWindow.DrawContents };
19  463 --
19  464 --
19  465 --      {$S MARes}
19  466 --      {-----------------------------------+
19  467 --      |   DrawResizeIcon                  |
19  468 --      +-----------------------------------}
19  469 -- A   PROCEDURE TWindow.DrawResizeIcon;
19  470 --
19  471 --      VAR
19  472 --          r:      Rect;
19  473 --
19  474 0- A   BEGIN
19  475 --          SetPort(fWmgrWindow);
19  476 --
19  477 --          { origin doesn't matter since we use the portRect }
19  478 --          r := thePort^.portRect;
19  479 --          r.left := r.right - kSBarSizeMinus1;
19  480 --          r.top := r.bottom - kSBarSizeMinus1;
19  481 --
19  482 --      {$IFC qDebug}
19  483 --          UseTempRgn('TWindow.DrawResizeIcon');
19  484 --      {$ENDC}
19  485 --
19  486 --          GetClip(gTempRgn);
19  487 --          ClipRect(r);
19  488 --          PenNormal;
19  489 --          DrawGrowIcon(fWmgrWindow);
19  490 --          SetClip(gTempRgn);
19  491 --
19  492 --      {$IFC qDebug}
19  493 --          DoneWithTempRgn;
19  494 --      {$ENDC}
```

```
19  495 -0 A   END { TWindow.DrawResizeIcon };
19  496 --
19  497 --
19  498 --     {$S MARes}
19  499 --     {-----------------------------------+
19  500 --     |   Focus                           |
19  501 --     +-----------------------------------}
19  502 -- A   FUNCTION  TWindow.Focus: BOOLEAN; OVERRIDE;
19  503 --
19  504 --     VAR
19  505 --         r:          Rect;
19  506 --         aRgn:       RgnHandle;
19  507 --
19  508 0- A   BEGIN
19  509 --         IF gFocusedView = SELF THEN
19  510 1-             BEGIN
19  511 --     {$IFC qDebug}
19  512 --             IF LONGINT(fWMgrWindow^.portRect.topLeft) <> 0 THEN
19  513 --                 ProgramBreak('TWindow.Focus: Origin is not (0,0)');
19  514 --     {$ENDC}
19  515 --             Focus := IsShown;
19  516 -1             END
19  517 --         ELSE IF fWMgrWindow <> NIL THEN
19  518 1-             BEGIN
19  519 --             SetPort(fWmgrWindow);
19  520 --             SetOrigin(0, 0);
19  521 --             gLongOffset := gZeroVPt;
19  522 --             ClipRect(thePort^.portRect);
19  523 --             gFocusedView := SELF;
19  524 --             Focus := IsShown;
19  525 -1             END
19  526 --         ELSE
19  527 --             Focus := FALSE;
19  528 -0 A   END { TWindow.Focus };
19  529 --
19  530 --
19  531 --     {$S MARes}
19  532 --     {-----------------------------------+
19  533 --     |   FocusOnSuperView                |
19  534 --     +-----------------------------------}
19  535 -- A   FUNCTION  TWindow.FocusOnSuperView: BOOLEAN; OVERRIDE;
19  536 --
19  537 0- A   BEGIN
19  538 --         FocusOnSuperView := Focus;
19  539 -0 A   END { TWindow.FocusOnSuperView };
19  540 --
19  541 --
19  542 --     {$S MAOpen}
19  543 --     {-----------------------------------+
19  544 --     |   ForceOnScreen                   |
19  545 --     +-----------------------------------}
19  546 -- A   PROCEDURE TWindow.ForceOnScreen;
19  547 --
19  548 --     CONST
19  549 --         here =          'ForceOnScreen';
19  550 --         GrayRgn =       $9EE;
19  551 --
19  552 --     VAR
19  553 --         global:         Rect;
19  554 --         tempRect:       Rect;
19  555 --         deltaH:         INTEGER;
19  556 --         deltaV:         INTEGER;
19  557 --         ptrToRgnHandle: ^RgnHandle;
19  558 --
19  559 0- A   BEGIN
19  560 --         fForcedOnScreen := TRUE;
19  561 --
19  562 --         GetGlobalBounds(global);
19  563 --
19  564 --         { Make sure the window is not too big. }
19  565 --         WITH global DO
19  566 1-             BEGIN
19  567 --             IF right - left > fResizeLimits.right THEN
19  568 --                 right := left + fResizeLimits.right - 1;
19  569 --
19  570 --             IF bottom - top > fResizeLimits.bottom THEN
19  571 --                 bottom := top + fResizeLimits.bottom - 1;
19  572 -1             END;
19  573 --
19  574 --         deltaH := 0;
19  575 --         deltaV := 0;
19  576 --
19  577 --         {$IFC qDebug}
19  578 --         UseTempRgn(here);
19  579 --         {$ENDC}
19  580 --
19  581 --         { On some systems (including the Radius FPD), it's possible to have a
19  582 --             non-rectangular desktop.  Try to be nice to people who saved windows
19  583 --             on secondary screens.  GrayRgn is the true indicator of the shape
19  584 --             of the desktop--screenBits.bounds is the size of the screen with the
19  585 --             menu bar on it. }
19  586 --         ptrToRgnHandle := POINTER(GrayRgn);
19  587 --         CopyRgn(ptrToRgnHandle^, gTempRgn);
19  588 --         InsetRgn(gTempRgn, gStdWScreenRect.left - 1,
19  589 --                             gStdWScreenRect.top - gMBarHeight - 1);
19  590 --
19  591 --         { RectInRgn sometimes lies on old ROMs, so if we're running on old ROMs
19  592 --             then always force the window into gStdWScreenBounds. }
19  593 --         IF NOT ((gConfiguration.hasROM128K) AND RectInRgn(global, gTempRgn)) THEN
```

```
19  594 --              WITH global DO
19  595 1-                  BEGIN
19  596 --
19  597 --                  IF bottom < gStdWScreenRect.top THEN
19  598 --                      deltaV := gStdWScreenRect.top - top
19  599 --                  ELSE IF top > gStdWScreenRect.bottom THEN
19  600 --                      deltaV := gStdWScreenRect.bottom - top;
19  601 --
19  602 --                  IF right < gStdWScreenRect.left THEN
19  603 --                      deltaH := gStdWScreenRect.left - right
19  604 --                  ELSE IF left > gStdWScreenRect.right THEN
19  605 --                      deltaH := gStdWScreenRect.right - left;
19  606 --
19  607 -1                  END;
19  608 --
19  609 --          {$IFC qDebug}
19  610 --          DoneWithTempRgn;
19  611 --          {$ENDC}
19  612 --
19  613 --          OffsetRect(global, deltaH, deltaV);
19  614 --          WITH global DO
19  615 1-              BEGIN
19  616 --              MoveWindow(fWMgrWindow, left, top, FALSE);
19  617 --              SizeWindow(fWMgrWindow, right - left, bottom - top, FALSE);
19  618 -1              END;
19  619 -0 A    END { TWindow.ForceOnScreen };
19  620 --
19  621 --
19  622 --      {$S MARes}
19  623 --      {-----------------------------------+
19  624 --      |   GetGlobalBounds                 |
19  625 --      +-----------------------------------}
19  626 -- A    PROCEDURE TWindow.GetGlobalBounds (VAR globalBounds: Rect);
19  627 --
19  628 --      VAR
19  629 --          savedPort:      GrafPtr;
19  630 --
19  631 0- A    BEGIN
19  632 --          IF fWMgrWindow <> NIL THEN
19  633 1-              BEGIN
19  634 --              GetPort(savedPort);
19  635 --              SetPort(fWmgrWindow);
19  636 --              globalBounds := fWmgrWindow^.portRect;
19  637 --              LocalToGlobal(globalBounds.topLeft);
19  638 --              LocalToGlobal(globalBounds.botRight);
19  639 --              SetPort(savedPort);
19  640 -1              END;
19  641 -0 A    END { TWindow.GetGlobalBounds };
19  642 --
19  643 --
19  644 --      {$IFC qInspector}
19  645 --      {$S MAInspector}
19  646 --      {-----------------------------------+
19  647 --      |   GetInspectorName                |
19  648 --      +-----------------------------------}
19  649 -- A    PROCEDURE TWindow.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
19  650 --
19  651 0- A    BEGIN
19  652 --          IF (fWMgrWindow <> NIL) & (NOT ODD(ORD4(fWMgrWindow))) THEN
19  653 --              GetWTitle(fWMgrWindow, inspectorName);
19  654 -0 A    END { TWindow.GetInspectorName };
19  655 --      {$ENDC}
19  656 --
19  657 --
19  658 --      {$S MARes}
19  659 --      {-----------------------------------+
19  660 --      |   GetGrafPort                     |
19  661 --      +-----------------------------------}
19  662 -- A    FUNCTION  TWindow.GetGrafPort: GrafPtr; OVERRIDE;
19  663 --
19  664 0- A    BEGIN
19  665 --          GetGrafPort := GrafPtr(fWMgrWindow);
19  666 -0 A    END { TWindow.GetGrafPort };
19  667 --
19  668 --
19  669 --      {$S MARes}
19  670 --      {-----------------------------------+
19  671 --      |   GetWindow                       |
19  672 --      +-----------------------------------}
19  673 -- A    FUNCTION  TWindow.GetWindow: TWindow; OVERRIDE;
19  674 --
19  675 0- A    BEGIN
19  676 --          GetWindow := SELF;
19  677 -0 A    END { TWindow.GetWindow };
19  678 --
19  679 --
19  680 --      {$S MASelCommand}
19  681 --      {-----------------------------------+
19  682 --      |   HandleMouseDown                 |
19  683 --      +-----------------------------------}
19  684 -- A    FUNCTION  TWindow.HandleMouseDown (theMouse:        VPoint;
19  685 --                                         VAR info:        EventInfo;
19  686 --                                         VAR hysteresis:  Point;
19  687 --                                         VAR theCommand:  TCommand): BOOLEAN; OVERRIDE;
19  688 --
19  689 --      VAR
19  690 --          wantsTheMouse:      BOOLEAN;
19  691 --          anEvent:        EventRecord;
19  692 --
```

```
19  693 0- A   BEGIN
19  694 --         theCommand := gNoChanges;
19  695 --
19  696 --         wantsTheMouse := TRUE;
19  697 --         IF fWMgrWindow <> FrontWindow THEN
19  698 1-             BEGIN
19  699 --             Select;
19  700 --             IF fDoFirstClick THEN
19  701 --                 { Make sure this window is activated and updated }
19  702 --                 WHILE gApplication.GetEvent(activMask, 0, NIL, anEvent) DO
19  703 --                     gApplication.HandleEvent(anEvent)
19  704 --             ELSE
19  705 --                 wantsTheMouse := FALSE;
19  706 -1             END;
19  707 --
19  708 --         IF wantsTheMouse THEN
19  709 --             HandleMouseDown := INHERITED HandleMouseDown(theMouse, info, hysteresis, theCommand)
19  710 --         ELSE
19  711 --             HandleMouseDown := FALSE;
19  712 -0 A   END { TWindow.HandleMouseDown };
19  713 --
19  714 --
19  715 --     {$S MAOpen}
19  716 --     {-----------------------------------+
19  717 --     |   InstallDocument                 |
19  718 --     +-----------------------------------}
19  719 -- A   PROCEDURE TWindow.InstallDocument (itsDocument: TDocument);
19  720 --
19  721 --     VAR
19  722 --         aString:        Str255;
19  723 --
19  724 0- A   BEGIN
19  725 --         fDocument := itsDocument;
19  726 --         IF itsDocument <> NIL THEN
19  727 1-             BEGIN
19  728 --             gApplication.DeleteFreeWindow(SELF);    { Just in case… }
19  729 --             itsDocument.AddWindow(SELF);
19  730 --             aString := itsDocument.fTitle;          { SetTitleForDoc may move memory }
19  731 --             IF aString <> '' THEN                   { not an untitled document }
19  732 --                 SetTitleForDoc(aString);
19  733 --             fNextHandler := itsDocument;
19  734 -1             END
19  735 --         ELSE
19  736 1-             BEGIN
19  737 --             gApplication.AddFreeWindow(SELF);
19  738 --             fNextHandler := gApplication;
19  739 -1             END;
19  740 -0 A   END { TWindow.InstallDocument };
19  741 --
19  742 --
19  743 --     {$S MARes}
19  744 --     {-----------------------------------+
19  745 --     |   IsShown                         |
19  746 --     +-----------------------------------}
19  747 -- A   FUNCTION  TWindow.IsShown: BOOLEAN; OVERRIDE;
19  748 --
19  749 0- A   BEGIN
19  750 --         IF fWMgrWindow <> NIL THEN
19  751 --             IsShown := WindowPeek(fWMgrWindow)^.visible
19  752 --         ELSE
19  753 --             IsShown := FALSE;
19  754 -0 A   END { TWindow.IsShown };
19  755 --
19  756 --
19  757 --     {$S MARes}
19  758 --     {-----------------------------------+
19  759 --     |   Locate                          |
19  760 --     +-----------------------------------}
19  761 -- A   PROCEDURE TWindow.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
19  762 --
19  763 0- A   BEGIN
19  764 --         IF fWMgrWindow <> NIL THEN
19  765 --             MoveWindow(fWMgrWindow, h, v, FALSE);
19  766 -0 A   END { TWindow.Locate };
19  767 --
19  768 --
19  769 --     {$S MARes}
19  770 --     {-----------------------------------+
19  771 --     |   Select                          |
19  772 --     +-----------------------------------}
19  773 -- A   PROCEDURE TWindow.Select;
19  774 --
19  775 0- A   BEGIN
19  776 --         gApplication.SelectWMgrWindow(fWMgrWindow);
19  777 -0 A   END { TWindow.Select };
19  778 --
19  779 --
19  780 --     {$S MARes}
19  781 --     {-----------------------------------+
19  782 --     |   MoveByUser                      |
19  783 --     +-----------------------------------}
19  784 -- A   PROCEDURE TWindow.MoveByUser (globalMouse: Point);
19  785 --
19  786 --     VAR
19  787 --         boundsRect: Rect;
19  788 --
19  789 0- A   BEGIN
19  790 --         boundsRect := fMoveBounds;
19  791 --         DragWindow(fWMgrWindow, globalMouse, boundsRect);
```

```
19  792 --          gLastUpTime := TickCount;   { needed for double clicking the title bar,
19  793 --                                        because DragWindow eats the mouseUpEvent }
19  794 -0 A    END { TWindow.MoveByUser };
19  795 --
19  796 --
19  797 --      {$S MAOpen}
19  798 --      {-----------------------------------+
19  799 --      |   Open                            |
19  800 --      +-----------------------------------}
19  801 -- A    PROCEDURE TWindow.Open; OVERRIDE;
19  802 --
19  803 0- A    BEGIN
19  804 --          WITH fWMgrWindow^.portRect DO
19  805 --              Resize(right - left, bottom - top, FALSE);  { Be sure all views are sized right }
19  806 --          Show(TRUE, kRedraw);        { Make me visible }
19  807 --
19  808 --          INHERITED Open;             { Tell subviews to open in case they care }
19  809 -0 A    END { TWindow.Open };
19  810 --
19  811 --
19  812 --      {$S MANonRes}
19  813 --      {-----------------------------------+
19  814 --      |   Resize                          |
19  815 --      +-----------------------------------}
19  816 -- A    PROCEDURE TWindow.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
19  817 --
19  818 --      VAR
19  819 --          r:      Rect;
19  820 --
19  821 0- A    BEGIN
19  822 --          IF (width <> fSize.h) | (height <> fSize.v) THEN
19  823 1-              BEGIN
19  824 --              SizeWindow(fWMgrWindow, width, height, FALSE);
19  825 --
19  826 --              IF fIsResizable THEN { inval old and new positions of grow box }
19  827 --                  IF Focus THEN
19  828 2-                      BEGIN
19  829 --                      SetRect(r, -kSBarSizeMinus1, -kSBarSizeMinus1, 0, 0);
19  830 --
19  831 --                      OffsetRect(r, fSize.h, fSize.v);
19  832 --                      EraseRect(r);
19  833 --                      InvalidRect(r);
19  834 --
19  835 --                      OffsetRect(r, width - fSize.h, height - fSize.v);
19  836 --                      InvalidRect(r);
19  837 -2                      END;
19  838 --
19  839 --              INHERITED Resize(width, height, invalidate);
19  840 -1              END;
19  841 -0 A    END { TWindow.Resize };
19  842 --
19  843 --
19  844 --      {$S MANonRes}
19  845 --      {-----------------------------------+
19  846 --      |   ResizeByUser                    |
19  847 --      +-----------------------------------}
19  848 -- A    PROCEDURE TWindow.ResizeByUser (globalMouse: Point);
19  849 --
19  850 --      VAR
19  851 --          growResult:     LONGINT;
19  852 --          resizeLimits:   Rect;
19  853 --
19  854 0- A    BEGIN
19  855 --          resizeLimits := fResizeLimits;          { GrowWindow may move SELF }
19  856 --          growResult := GrowWindow(fWMgrWindow, globalMouse, resizeLimits);
19  857 --          IF growResult <> 0 THEN
19  858 --              Resize(LoWrd(growResult), HiWrd(growResult), TRUE);
19  859 -0 A    END { TWindow.ResizeByUser };
19  860 --
19  861 --
19  862 --      {$S MAOpen}
19  863 --      {-----------------------------------+
19  864 --      |   SetResizeLimits                 |
19  865 --      +-----------------------------------}
19  866 -- A    PROCEDURE TWindow.SetResizeLimits (minSize, maxSize: Point);
19  867 --
19  868 --      TYPE
19  869 --          PWStateData =   ^WStateData;
19  870 --          HWStateData =   ^PWStateData;
19  871 --
19  872 0- A    BEGIN
19  873 --          { ??? Do we want to force the window's size to change if it is not
19  874 --              within the minimum and maximum size given above? }
19  875 --
19  876 --          fResizeLimits.topLeft := minSize;
19  877 --          fResizeLimits.botRight := maxSize;
19  878 --
19  879 --          IF BAND(fProcID, 8) <> 0 THEN
19  880 --              WITH HWStateData(WindowPeek(fWMgrWindow)^.DataHandle)^^.stdState DO
19  881 1-                  BEGIN
19  882 --                  right := Min(right, left + maxSize.h);
19  883 --                  bottom := Min(bottom, top + maxSize.v);
19  884 -1                  END;
19  885 -0 A    END { TWindow.SetResizeLimits };
19  886 --
19  887 --
19  888 --      {$S MARes}
19  889 --      {-----------------------------------+
19  890 --      |   SetTarget                       |
```

```
19  891 --      +-----------------------------------}
19  892 -- A    PROCEDURE TWindow.SetTarget (newTarget: TEvtHandler);
19  893 --
19  894 0- A    BEGIN
19  895 --          fTarget := newTarget;
19  896 --          IF gFrontWindow = SELF THEN
19  897 --              gApplication.SetTarget(newTarget);
19  898 -0 A    END { TWindow.SetTarget };
19  899 --
19  900 --
19  901 --      {$S MAFile}
19  902 --      {-----------------------------------+
19  903 --      |   SetTitleForDoc                  |
19  904 --      +-----------------------------------}
19  905 -- A    PROCEDURE TWindow.SetTitleForDoc (newDocTitle: Str255);
19  906 --
19  907 --      VAR
19  908 --          title:          Str255;
19  909 --
19  910 0- A    BEGIN
19  911 --          IF fPreDocname > 0 THEN { optimize for case of nothing to do }
19  912 1-              BEGIN
19  913 --              GetWTitle(fWmgrWindow, title);
19  914 --
19  915 --              IF SubstituteInTitle(title, newDocTitle, fPreDocname, fConstTitle) THEN
19  916 --                  SetWTitle(fWmgrWindow, title);
19  917 -1              END;
19  918 -0 A    END { TWindow.SetTitleForDoc };
19  919 --
19  920 --
19  921 --      {$S MANonRes}
19  922 --      {-----------------------------------+
19  923 --      |   Show                            |
19  924 --      +-----------------------------------}
19  925 -- A    PROCEDURE TWindow.Show (state, redraw: BOOLEAN); OVERRIDE;
19  926 --
19  927 0- A    BEGIN
19  928 --          IF (fWMgrWindow <> NIL) AND redraw THEN
19  929 --              IF state THEN
19  930 --                  ShowWindow(fWMgrWindow)
19  931 --              ELSE
19  932 --                  HideWindow(fWMgrWindow);
19  933 --          fShown := state;
19  934 -0 A    END;
19  935 --
19  936 --
19  937 --      {$S MAOpen}
19  938 --      {-----------------------------------+
19  939 --      |   SimpleStagger                   |
19  940 --      +-----------------------------------}
19  941 -- A    PROCEDURE TWindow.SimpleStagger (dh, dv: INTEGER; VAR counter: INTEGER);
19  942 --
19  943 --      VAR
19  944 --          posRect:        Rect;       { The topLeft of the window must fall in here. }
19  945 --          pt:             Point;
19  946 --          nSlots:         INTEGER;
19  947 --          slot:           INTEGER;
19  948 --          wTopLeft:       Point;
19  949 --
19  950 0- A    BEGIN
19  951 --          fStaggered := TRUE;
19  952 --          { Get window rect, which gives us posRect.topLeft & size of window }
19  953 --          GetGlobalBounds(posRect);
19  954 --          wTopLeft := posRect.topLeft;
19  955 --
19  956 --          { Compute posRect.botRight = wTopLeft + <bounds botRight> - <window botRight> }
19  957 --          pt := wTopLeft;
19  958 --          AddPt(fMoveBounds.botRight, pt);
19  959 --          SubPt(posRect.botRight, pt);
19  960 --          posRect.botRight := pt;
19  961 --
19  962 --          { This covers the case of dh & dv both >= 0; if either is less than
19  963 --              0, the right (bottom) limit is set to the left (top) edge of the
19  964 --              screen -- allowing for some margin.  This makes the right (bottom)
19  965 --              edge less than the left (top) edge, but this is OK since the DIV
19  966 --              statement below will be dividing 2 negative numbers. }
19  967 --          IF dh < 0 THEN
19  968 --              posRect.right := fMoveBounds.left;
19  969 --          IF dv < 0 THEN
19  970 --              posRect.bottom := fMoveBounds.top;
19  971 --
19  972 --          { This code avoids divide by zero problems }
19  973 --          IF (dh = 0) OR (dv = 0) THEN
19  974 --              nSlots := 0
19  975 --          ELSE
19  976 --              nSlots := Min((posRect.right-posRect.left+dh-1) DIV dh,
19  977 --                            (posRect.bottom-posRect.top+dv-1) DIV dv );
19  978 --          IF nSlots = 0 THEN
19  979 --              slot := 0
19  980 --          ELSE
19  981 --              slot := counter MOD nSlots;
19  982 --
19  983 --          IF slot <> 0 THEN { move the window }
19  984 1-              BEGIN
19  985 --              { pt ends up as the place to position the window }
19  986 --              pt := wTopLeft;
19  987 --
19  988 --              pt.h := pt.h + (slot * dh);
19  989 --              pt.v := pt.v + (slot * dv);
```

```
19  990 --
19  991 --              MoveWindow(fWMgrWindow, pt.h, pt.v, FALSE);
19  992 -1              END;
19  993 --
19  994 --          counter := counter + 1;
19  995 -0 A    END { TWindow.SimpleStagger };
19  996 --
19  997 --
19  998 --      {$S MANonRes}
19  999 --      {-----------------------------------+
19 1000 --      |   Zoom                            |
19 1001 --      +-----------------------------------}
19 1002 -- A    PROCEDURE TWindow.Zoom (partCode: INTEGER);
19 1003 --
19 1004 0- A    BEGIN
19 1005 --          { Erasing was suggested by Ernie B.; the Finder does this.
19 1006 --              The ROM has a bug requiring that thePort be the window
19 1007 --              being zoomed. }
19 1008 --          IF Focus THEN       { ??? This may not be necessary }
19 1009 1-              BEGIN
19 1010 --              EraseRect(thePort^.portRect);
19 1011 --              ZoomWindow(fWmgrWindow, partCode, FALSE);
19 1012 --
19 1013 --              WITH fWmgrWindow^.portRect DO
19 1014 --                  Resize(right - left, bottom - top, TRUE);
19 1015 -1              END;
19 1016 -0 A    END { TWindow.Zoom };
19 1017 --
19 1018 --      {$IFC qDebug}
19 1019 --          {$S MAFields}
19 1020 --          {-----------------------------------+
19 1021 --          |   Fields                          |
19 1022 --          +-----------------------------------}
19 1023 -- A        PROCEDURE TWindow.Fields (PROCEDURE DoToField (fieldName: Str255;
19 1024 --                                                        fieldAddr: Ptr;
19 1025 --                                                        fieldType: INTEGER)); OVERRIDE;
19 1026 --          VAR
19 1027 --              theFlag:        BOOLEAN;
19 1028 --
19 1029 0- A        BEGIN
19 1030 --              DoToField('TWindow', NIL, bClass);
19 1031 --              DoToField('fWMgrWindow', @fWMgrWindow, bWindowPtr);
19 1032 --              DoToField('fProcID', @fProcID, bInteger);
19 1033 --              DoToField('fMoveBounds', @fMoveBounds, bRect);
19 1034 --              DoToField('fResizeLimits', @fResizeLimits, bRect);
19 1035 --              DoToField('fTarget', @fTarget, bObject);
19 1036 --              DoToField('fTargetID', @fTargetID, bOSType);
19 1037 --              DoToField('fPreDocName', @fPreDocName, bInteger);
19 1038 --              DoToField('fConstTitle', @fConstTitle, bInteger);
19 1039 --              DoToField('fAdapted', @fAdapted, bBoolean);
19 1040 --              DoToField('fHorzCentered', @fHorzCentered, bBoolean);
19 1041 --              DoToField('fVertCentered', @fVertCentered, bBoolean);
19 1042 --              DoToField('fStaggered', @fStaggered, bBoolean);
19 1043 --              DoToField('fForcedOnScreen', @fForcedOnScreen, bBoolean);
19 1044 --              DoToField('fIsActive', @fIsActive, bBoolean);
19 1045 --              DoToField('fIsResizable', @fIsResizable, bBoolean);
19 1046 --              DoToField('fIsClosable', @fIsClosable, bBoolean);
19 1047 --              DoToField('fFreeOnClosing', @fFreeOnClosing, bBoolean);
19 1048 --              DoToField('fDisposeOnFree', @fDisposeOnFree, bBoolean);
19 1049 --              DoToField('fClosesDocument', @fClosesDocument, bBoolean);
19 1050 --              DoToField('fOpenInitially', @fOpenInitially, bBoolean);
19 1051 --              DoToField('fIsModal', @fIsModal, bBoolean);
19 1052 --              DoToField('fDoFirstClick', @fDoFirstClick, bBoolean);
19 1053 --              INHERITED Fields(DoToField);
19 1054 -0 A        END { TWindow.Fields };
19 1055 --      {$ENDC}
13 3007 --      {$I UMacApp.TScroller.p}
```

```
20    1 --      {$P}
20    2 --      { UMacApp.TScroller.p }
20    3 --      { Copyright © 1987, 1988 by Apple Computer Inc.  All rights reserved. }
20    4 --
20    5 --      (*****************************************************************************************)
20    6 --      (*      T S c r o l l e r                                                                *)
20    7 --      (*****************************************************************************************)
20    8 --      {-----------------------------------+
20    9 --      |   IScroller                       |
20   10 --      +-----------------------------------}
20   11 --      {$IFC qProceduralViews}
20   12 --      {$S MAOpen}
20   13 -- A    PROCEDURE TScroller.IScroller (itsSuperView: TView; itsLocation, itsSize: VPoint;
20   14 --                                     itsHSizeDet, itsVSizeDet: SizeDeterminer;
20   15 --                                     itsHorzMax, itsVertMax: VCoordinate;
20   16 --                                     wantHorzSBar, wantVertSBar: BOOLEAN);
20   17 --      VAR
20   18 --          scrollLimit:    VPoint;
20   19 --
20   20 0- A    BEGIN
20   21 --          fScrollBars[h] := NIL;
20   22 --          fScrollBars[v] := NIL;
20   23 --
20   24 --          IView(NIL, itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
20   25 --
20   26 --          fTranslation := gZeroVPt;
20   27 --          fMaxTranslation := gZeroVPt;
20   28 --          fSBarOffsets := gZeroVRect;
20   29 --
20   30 --          SetScrollParameters(kStdScrollUnit, kStdScrollUnit, FALSE, FALSE);
20   31 --
20   32 --          scrollLimit.h := itsHorzMax;
20   33 --          scrollLimit.v := itsVertMax;
20   34 --          SetScrollLimits(scrollLimit, kDontRedraw);
20   35 --
20   36 --          CreateScrollBars(wantHorzSBar, wantVertSBar);
20   37 --
20   38 -0 A    END {TScroller.IScroller};
20   39 --      {$ENDC}
20   40 --
20   41 --      {-----------------------------------+
20   42 --      |   IRes                            |
20   43 --      +-----------------------------------}
20   44 --      {$IFC qTemplateViews}
20   45 --      {$S MAOpen}
20   46 -- A    PROCEDURE TScroller.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
20   47 --
20   48 --      VAR
20   49 --          scrollLimit:    VPoint;
20   50 --
20   51 0- A    BEGIN
20   52 --          fScrollBars[h] := NIL;
20   53 --          fScrollBars[v] := NIL;
20   54 --
20   55 --          INHERITED IRes(NIL, itsSuperView, itsParams);
20   56 --
20   57 --          WITH ScrollerTemplatePtr(itsParams)^ DO
20   58 1-              BEGIN
20   59 --              fTranslation := gZeroVPt;
20   60 --              scrollLimit.h := horzMax;
20   61 --              scrollLimit.v := vertMax;
20   62 --              SetScrollLimits(scrollLimit, kDontRedraw);
20   63 --
20   64 --              SetScrollParameters(hScrollUnits, vScrollUnits, hConstrain, vConstrain);
20   65 --          {$H-}
20   66 --              RectToVRect(sBarOffsets, fSBarOffsets);
20   67 --          {$H+}
20   68 --              CreateScrollBars(wantHSBar, wantVSBar);
20   69 -1              END;
20   70 --
20   71 --          OffsetPtr(itsParams, SIZEOF(ScrollerTemplate));
20   72 -0 A    END;
20   73 --      {$ENDC}
20   74 --
20   75 --      {$IFC qWriteTemplates}
20   76 --      {$S MAWriteRes}
20   77 --      {-----------------------------------+
20   78 --      |   WRes                            |
20   79 --      +-----------------------------------}
20   80 -- A    PROCEDURE TScroller.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
20   81 --
20   82 --      VAR
20   83 --          scPtr:      ScrollerTemplatePtr;
20   84 --
20   85 0- A    BEGIN
20   86 --          INHERITED WRes(theResource, itsParams);
20   87 --
20   88 --          scPtr := ScrollerTemplatePtr(ExpandPtr(theResource, itsParams,
20   89 --            SIZEOF(ScrollerTemplate)));
20   90 --
20   91 --          WITH scPtr^ DO
20   92 1-              BEGIN
20   93 --              wantVSBar := fScrollBars[h] <> NIL;
20   94 --              wantHSBar := fScrollBars[v] <> NIL;
20   95 --              vertMax := fScrollLimit.v;
20   96 --              horzMax := fScrollLimit.h;
20   97 --              vScrollUnits := fScrollUnit.v;
20   98 --              hScrollUnits := fScrollUnit.h;
```

```
20   99 --              vConstrain := fConstrain[v];
20  100 --              hConstrain := fConstrain[h];
20  101 --              VRectToRect(fSBarOffsets, sBarOffsets);
20  102 -1             END;
20  103 -0 A    END;
20  104 --      {$ENDC}
20  105 --
20  106 --      {$IFC qWriteTemplates}
20  107 --      {$S MAWriteRes}
20  108 --      {-----------------------------------+
20  109 --      |   WriteRes                        |
20  110 --      +-----------------------------------}
20  111 -- A    PROCEDURE TScroller.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
20  112 0- A    BEGIN
20  113 --          gWResSignature := 'scrl';   gWResType := 'TScroller';
20  114 --          WRes(theResource, itsParams);
20  115 -0 A    END;
20  116 --      {$ENDC}
20  117 --
20  118 --      {-----------------------------------+
20  119 --      |   Free                            |
20  120 --      +-----------------------------------}
20  121 --      {$S MAClose}
20  122 -- A    PROCEDURE TScroller.Free; OVERRIDE;
20  123 --
20  124 0- A    BEGIN
20  125 --          IF fScrollBars[h] <> NIL THEN
20  126 --              fScrollBars[h].Free;
20  127 --          IF fScrollBars[v] <> NIL THEN
20  128 --              fScrollBars[v].Free;
20  129 --
20  130 --          INHERITED Free;
20  131 -0 A    END {TScroller.Free};
20  132 --
20  133 --      {-----------------------------------+
20  134 --      |   AddSubView                      |
20  135 --      +-----------------------------------}
20  136 --      {$S MAOpen}
20  137 -- A    PROCEDURE TScroller.AddSubView (theSubView: TView); OVERRIDE;
20  138 --
20  139 0- A    BEGIN
20  140 --          INHERITED AddSubView(theSubView);
20  141 --          theSubView.BeInScroller(SELF);
20  142 -0 A    END {TScroller.AddSubView};
20  143 --
20  144 --      {-----------------------------------+
20  145 --      |   AdjustScrollBars                |
20  146 --      +-----------------------------------}
20  147 --      {$S MANonRes}
20  148 -- A    PROCEDURE TScroller.AdjustScrollBars;
20  149 --
20  150 --      VAR
20  151 --          vhs:            VHSelect;
20  152 --          aScrollBar:     TScrollBar;
20  153 --          ortho:          VHSelect;
20  154 --          loc:            VPoint;
20  155 --          size:           VPoint;
20  156 --
20  157 0- A    BEGIN
20  158 --          FOR vhs := v TO h DO
20  159 1-              BEGIN
20  160 --              aScrollBar := fScrollBars[vhs];
20  161 --              IF aScrollBar <> NIL THEN
20  162 2-                  BEGIN
20  163 --                  ortho := gOrthogonal[vhs];
20  164 --                  loc := fLocation;
20  165 --                  size := fSize;
20  166 --                  loc.vh[vhs] := loc.vh[vhs] + fSBarOffsets.topLeft.vh[vhs] - 1;
20  167 --                  loc.vh[gOrthogonal[vhs]] := loc.vh[ortho] + size.vh[ortho];
20  168 --                  size.vh[vhs] := size.vh[vhs] - fSBarOffsets.topLeft.vh[vhs] +
20  169 --                                                 fSBarOffsets.botRight.vh[vhs] + 2;
20  170 --                  size.vh[ortho] := kSBarSize;
20  171 --
20  172 --                  aScrollBar.Resize(size.h, size.v, TRUE);
20  173 --                  aScrollBar.Locate(loc.h, loc.v, TRUE);
20  174 -2                  END;
20  175 -1              END;
20  176 -0 A    END {TScroller.AdjustScrollBars};
20  177 --
20  178 --      {-----------------------------------+
20  179 --      |   AutoScroll                      |
20  180 --      +-----------------------------------}
20  181 --      {$S MADoCommand}
20  182 -- A    PROCEDURE TScroller.AutoScroll (viewPt: VPoint; VAR delta: VPoint);
20  183 --
20  184 --      VAR
20  185 --          vhs:        VHSelect;
20  186 --          myExtent:   VRect;
20  187 --
20  188 0- A    BEGIN
20  189 --          {??? Do we want to autoscroll if the edges of the frame are covered?}
20  190 --          {??? What about scrolling when the grow icon is in front?}
20  191 --
20  192 --          delta := gZeroVPt;
20  193 --          GetExtent(myExtent);
20  194 --          FOR vhs := v TO h DO
20  195 --              IF viewPt.vh[vhs] < myExtent.topLeft.vh[vhs] THEN
20  196 --                  delta.vh[vhs] := Max(-fScrollUnit.vh[vhs], -fTranslation.vh[vhs])
20  197 --              ELSE IF viewPt.vh[vhs] > myExtent.botRight.vh[vhs] THEN
```

```
20  198 --                  delta.vh[vhs] := Min(fScrollUnit.vh[vhs],
20  199 --                          fMaxTranslation.vh[vhs] - fTranslation.vh[vhs]);
20  200 -0 A    END {TScroller.AutoScroll};
20  201 --
20  202 --      {-----------------------------------+
20  203 --      |   CreateScrollBars                |
20  204 --      +-----------------------------------}
20  205 --      {$S MAOpen}
20  206 -- A    PROCEDURE TScroller.CreateScrollBars (wantHorzSBar, wantVertSBar: BOOLEAN);
20  207 --
20  208 --      VAR
20  209 --          sBarLocation,
20  210 --          sBarSize:       VPoint;
20  211 --          anSScrollBar:   TSScrollBar;
20  212 --          fi:             FailInfo;
20  213 --
20  214 -- B        PROCEDURE HandleFailure (error: OSErr; message: LONGINT);
20  215 0- B       BEGIN
20  216 --              Free;
20  217 -0 B        END;
20  218 --
20  219 0- A    BEGIN
20  220 --          CatchFailures(fi, HandleFailure);
20  221 --
20  222 --          {??? Perhaps use AdjustScrollBars?, or local nested routine to create sbars?}
20  223 --          IF wantHorzSBar THEN
20  224 1-              BEGIN
20  225 --              SetVPt(sBarLocation, fLocation.h + fSBarOffsets.left - 1, fLocation.v + fSize.v);
20  226 --              SetVPt(sBarSize, fSize.h - fSBarOffsets.left + fSBarOffsets.right + 2, kSBarSize);
20  227 --              New(anSScrollBar);
20  228 --              FailNIL(anSScrollBar);
20  229 --              anSScrollBar.ISScrollBar(fSuperView, sBarLocation, sBarSize, SizeVariable,
20  230 --                                       SizeVariable, h, fMaxTranslation.h, SELF);
20  231 -1              END
20  232 --          ELSE
20  233 --              fScrollBars[h] := NIL;
20  234 --
20  235 --          IF wantVertSBar THEN
20  236 1-              BEGIN
20  237 --              SetVPt(sBarLocation, fLocation.h + fSize.h, fLocation.v + fSBarOffsets.top - 1);
20  238 --              SetVPt(sBarSize, kSBarSize, fSize.v - fSBarOffsets.top + fSBarOffsets.bottom + 2);
20  239 --              New(anSScrollBar);
20  240 --              FailNIL(anSScrollBar);
20  241 --              anSScrollBar.ISScrollBar(fSuperView, sBarLocation, sBarSize,  SizeVariable,
20  242 --                                       SizeVariable, v, fMaxTranslation.v, SELF);
20  243 -1              END
20  244 --          ELSE
20  245 --              fScrollBars[v] := NIL;
20  246 --
20  247 --          Success(fi);
20  248 -0 A    END {TScroller.CreateScrollBars};
20  249 --
20  250 --      {-----------------------------------+
20  251 --      |   DoScroll                        |
20  252 --      +-----------------------------------}
20  253 --      {$S MAScroll}
20  254 -- A    PROCEDURE TScroller.DoScroll (delta: VPoint; redraw: BOOLEAN);
20  255 --
20  256 --      VAR
20  257 --          vhs:            VHSelect;
20  258 --          theWindow:      TWindow;
20  259 --
20  260 0- A    BEGIN
20  261 --          FOR vhs := v TO h DO
20  262 --              IF delta.vh[vhs] < 0 THEN
20  263 --                  delta.vh[vhs] := Max(delta.vh[vhs], -fTranslation.vh[vhs])
20  264 --              ELSE IF delta.vh[vhs] > 0 THEN
20  265 --                  delta.vh[vhs] := Min(delta.vh[vhs],
20  266 --                          fMaxTranslation.vh[vhs] - fTranslation.vh[vhs]);
20  267 --
20  268 --          IF (delta.h <> 0) | (delta.v <> 0) THEN
20  269 1-              BEGIN
20  270 --              IF redraw THEN
20  271 2-                  BEGIN
20  272 --                  theWindow := GetWindow;
20  273 --                  IF NOT EmptyRgn(WindowPeek(theWindow.fWMgrWindow)^.updateRgn) THEN
20  274 --                      theWindow.Update;
20  275 -2                  END;
20  276 --
20  277 --      {$H-}
20  278 --              AddVPt(delta, fTranslation);
20  279 --      {$H+}
20  280 --              IF gFocusedView = SELF THEN             { If we're focused              }
20  281 --                  gLongOffset := fTranslation;        { …then keep focus accurate.    }
20  282 --
20  283 --              IF redraw THEN
20  284 --                  ScrollDraw(delta);
20  285 -1              END;
20  286 -0 A    END {TScroller.DoScroll};
20  287 --
20  288 --
20  289 --      {$IFC qDebug}
20  290 --          {-----------------------------------+
20  291 --          |   Fields                          |
20  292 --          +-----------------------------------}
20  293 --          {$S MAFields}
20  294 -- A        PROCEDURE TScroller.Fields (PROCEDURE DoToField (fieldName: Str255;
20  295 --                                                           fieldAddr: Ptr;
20  296 --                                                           fieldType: INTEGER)); OVERRIDE;
```

```
20  297 --
20  298 0- A        BEGIN
20  299 --              DoToField('TScroller', NIL, bClass);
20  300 --              DoToField('fTranslation', @fTranslation, bVPoint);
20  301 --              DoToField('fScrollLimit', @fScrollLimit, bVPoint);
20  302 --              DoToField('fMaxTranslation', @fMaxTranslation, bVPoint);
20  303 --              DoToField('fScrollBars[v]', @fScrollBars[v], bObject);
20  304 --              DoToField('fScrollBars[h]', @fScrollBars[h], bObject);
20  305 --              DoToField('fScrollUnit', @fScrollUnit, bPoint);
20  306 --              DoToField('fSBarOffsets', @fSBarOffsets, bVRect);
20  307 --              DoToField('fConstrain[h]', @fConstrain[h], bBoolean);
20  308 --              DoToField('fConstrain[v]', @fConstrain[v], bBoolean);
20  309 --              INHERITED Fields(DoToField);
20  310 -0 A        END {TScroller.Fields};
20  311 --      {$ENDC}
20  312 --
20  313 --      {-----------------------------------+
20  314 --      |   Focus                           |
20  315 --      +-----------------------------------}
20  316 --      {$S MARes}
20  317 -- A    FUNCTION  TScroller.Focus: BOOLEAN;
20  318 --
20  319 0- A    BEGIN
20  320 --          Focus := INHERITED Focus;
20  321 --          gLongOffset := fTranslation
20  322 -0 A    END {TScroller.Focus};
20  323 --
20  324 --      {-----------------------------------+
20  325 --      |   GetExtent                       |
20  326 --      +-----------------------------------}
20  327 --      {$S MARes}
20  328 -- A    PROCEDURE TScroller.GetExtent (VAR itsExtent: VRect); OVERRIDE;
20  329 --
20  330 0- A    BEGIN
20  331 --          INHERITED GetExtent(itsExtent);
20  332 --          OffsetVRect(itsExtent, fTranslation.h, fTranslation.v);
20  333 -0 A    END {TScroller.GetExtent};
20  334 --
20  335 --      {-----------------------------------+
20  336 --      |   GetScroller                     |
20  337 --      +-----------------------------------}
20  338 --      {$S MARes}
20  339 -- A    FUNCTION TScroller.GetScroller (immediateSuperView: BOOLEAN): TScroller;
20  340 --
20  341 0- A    BEGIN
20  342 --          GetScroller := SELF;
20  343 -0 A    END {TScroller.GetScroller};
20  344 --
20  345 --      {-----------------------------------+
20  346 --      |   HaveScrollBar                   |
20  347 --      +-----------------------------------}
20  348 --      {$S MARes}
20  349 -- A    PROCEDURE TScroller.HaveScrollBar (theScrollBar: TSScrollBar;
20  350 --                                         direction: VHSelect);
20  351 --
20  352 0- A    BEGIN
20  353 --      {$IFC qDebug}
20  354 --          IF (theScrollBar <> NIL) & (theScrollBar.fDirection <> direction) THEN
20  355 --              ProgramBreak('TScroller.HaveScrollBar: Scroll bar is wrong direction.');
20  356 --      {$ENDC}
20  357 --          fScrollBars[direction] := theScrollBar;
20  358 -0 A    END {TScroller.HaveScrollBar};
20  359 --
20  360 --      {-----------------------------------+
20  361 --      |   LocalToSuper                    |
20  362 --      +-----------------------------------}
20  363 --      {$S MARes}
20  364 -- A    PROCEDURE TScroller.LocalToSuper (VAR thePoint: VPoint); OVERRIDE;
20  365 0- A    BEGIN
20  366 --          SubVPt(fTranslation, thePoint);
20  367 --          AddVPt(fLocation, thePoint);
20  368 -0 A    END {TScroller.LocalToSuper};
20  369 --
20  370 --      {-----------------------------------+
20  371 --      |   Locate                          |
20  372 --      +-----------------------------------}
20  373 --      {$S MANonRes}
20  374 -- A    PROCEDURE TScroller.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
20  375 --
20  376 0- A    BEGIN
20  377 --          INHERITED Locate(h, v, invalidate);
20  378 --
20  379 --          AdjustScrollBars;
20  380 -0 A    END {TScroller.Locate};
20  381 --
20  382 --      {-----------------------------------+
20  383 --      |   RemoveSubView                   |
20  384 --      +-----------------------------------}
20  385 --      {$S MANonRes}
20  386 -- A    PROCEDURE TScroller.RemoveSubView (theSubView: TView); OVERRIDE;
20  387 --
20  388 0- A    BEGIN
20  389 --          theSubView.BeInScroller(NIL);
20  390 --          INHERITED RemoveSubView(theSubView);
20  391 -0 A    END {TScroller.RemoveSubView};
20  392 --
20  393 --      {-----------------------------------+
20  394 --      |   Resize                          |
20  395 --      +-----------------------------------}
```

```
20  396 --      {$S MANonRes}
20  397 -- A    PROCEDURE TScroller.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
20  398 --
20  399 --      VAR
20  400 --          vhs:                VHSelect;
20  401 --          sBarWasVisible:     ARRAY [VHSelect] OF BOOLEAN;
20  402 --
20  403 0- A    BEGIN
20  404 --          { If the scroll bars are visible, erase them now so that they aren't temporarily
20  405 --            displayed in the wrong place.}
20  406 --          FOR vhs := v TO h DO
20  407 --              IF (fScrollbars[vhs] <> NIL) & fScrollBars[vhs].Focus & (fScrollBars[vhs].fCmgrControl^^.contrlVis <> 0) THEN
20  408 1-                  BEGIN
20  409 --                  sBarWasVisible[vhs] := TRUE;
20  410 --                  HideControl(fScrollBars[vhs].fCmgrControl);
20  411 -1                  END
20  412 --              ELSE
20  413 --                  sBarWasVisible[vhs] := FALSE;
20  414 --
20  415 --          INHERITED Resize(width, height, invalidate);
20  416 --          AdjustScrollBars;
20  417 --          SetScrollLimits(fScrollLimit, kDontRedraw); {Readjust sbar maximums and fMaxTranslation}
20  418 --
20  419 --          FOR vhs := v TO h DO
20  420 --              IF sBarWasVisible[vhs] THEN
20  421 --                  fScrollBars[vhs].fCMqrControl^^.contrlVis := 1;
20  422 -0 A    END {TScroller.Resize};
20  423 --
20  424 --      {-----------------------------------+
20  425 --      |   RevealRect                      |
20  426 --      +-----------------------------------}
20  427 --      {$S MAScroll}
20  428 -- A    PROCEDURE TScroller.RevealRect (rectToReveal: VRect; minToSee: Point;
20  429 --                                      redraw: BOOLEAN); OVERRIDE;
20  430 --
20  431 --      VAR
20  432 --          myExtent:       VRect;
20  433 --          vhs:            VHSelect;
20  434 --          minAmt:         VCoordinate;
20  435 --          coord:          VCoordinate;
20  436 --          delta:          VPoint;
20  437 --
20  438 0- A    BEGIN
20  439 --          GetExtent(myExtent);
20  440 --
20  441 --          FOR vhs := v TO h DO
20  442 1-              BEGIN
20  443 --              minAmt := MIN(LengthVRect(myExtent, vhs), minToSee.vh[vhs]);
20  444 --              coord := rectToReveal.topLeft.vh[vhs] + minAmt - myExtent.botRight.vh[vhs];
20  445 --              IF coord <= 0 THEN
20  446 --                  coord := Min(0, rectToReveal.botRight.vh[vhs] - minAmt - myExtent.topLeft.vh[vhs]);
20  447 --              delta.vh[vhs] := coord;
20  448 -1              END;
20  449 --
20  450 --          ScrollBy(delta.h, delta.v, redraw);
20  451 -0 A    END {TScroller.RevealRect};
20  452 --
20  453 --      {-----------------------------------+
20  454 --      |   ScrollBy                        |
20  455 --      +-----------------------------------}
20  456 --      {$S MAScroll}
20  457 -- A    PROCEDURE TScroller.ScrollBy (deltaH, deltaV: VCoordinate; redraw: BOOLEAN);
20  458 --
20  459 --      VAR
20  460 --          delta:      VPoint;
20  461 --
20  462 0- A    BEGIN
20  463 --          IF (deltaH <> 0) | (deltaV <> 0) THEN
20  464 1-              BEGIN
20  465 --              IF fScrollBars[v] <> NIL THEN
20  466 --                  fScrollBars[v].DeltaValue(deltaV);
20  467 --              IF fScrollBars[h] <> NIL THEN
20  468 --                  fScrollBars[h].DeltaValue(deltaH);
20  469 --
20  470 --              SetVPt(delta, deltaH, deltaV);
20  471 --              DoScroll(delta, redraw);
20  472 -1              END;
20  473 -0 A    END {TScroller.ScrollBy};
20  474 --
20  475 --      {-----------------------------------+
20  476 --      |   ScrollDraw                      |
20  477 --      +-----------------------------------}
20  478 --      {$S MAScroll}
20  479 -- A    PROCEDURE TScroller.ScrollDraw (delta: VPoint);
20  480 --
20  481 --      VAR
20  482 --          myExtent:       VRect;
20  483 --          r:              Rect;
20  484 --
20  485 0- A    BEGIN
20  486 --          IF Focus THEN
20  487 1-              BEGIN
20  488 --
20  489 --              GetExtent(myExtent);
20  490 --              ViewToQDRect(myExtent, r);  {!!! GetQDExtent doesn't work for scrollers}
20  491 --
20  492 --      {$IFC qDebug}
20  493 --              IF gIntenseDebugging THEN
20  494 2-                  BEGIN
```

```
20  495 --                  WrLblRect('                    r', r);
20  496 --                  WRITELN;
20  497 --                  WrLblVPt('          gLongOffset', gLongOffset);
20  498 --                  WRITELN;
20  499 -2                  END;
20  500 --      {$ENDC}
20  501 --
20  502 --              IF (ABS(delta.h) > kMaxCoord) | (ABS(delta.v) > kMaxCoord) THEN
20  503 --                  InvalidRect(r)
20  504 --              ELSE
20  505 2-                  BEGIN
20  506 --      {$IFC qDebug}
20  507 --                  UseTempRgn('TScroller.ScrollDraw');
20  508 --      {$ENDC}
20  509 --                  ScrollRect(r, -delta.h, -delta.v, gTempRgn);
20  510 --                  InvalRgn(gTempRgn);
20  511 --
20  512 --      {$IFC qDebug}
20  513 --                  DoneWithTempRgn;
20  514 --      {$ENDC}
20  515 -2                  END;
20  516 --
20  517 --              BeginUpdate(thePort);   {Use thePort because we know we're focused}
20  518 --              EraseRect(thePort^.portRect);
20  519 --              DrawContents;
20  520 --              EndUpdate(thePort);
20  521 -1              END;
20  522 -0 A   END {TScroller.ScrollDraw};
20  523 --
20  524 --      {-----------------------------------+
20  525 --      |   ScrollRelative                  |
20  526 --      +-----------------------------------}
20  527 --      {$S MAScroll}
20  528 -- A   FUNCTION  TScroller.ScrollRelative (vhs: VHSelect; sBarValue: VCoordinate): VCoordinate;
20  529 --
20  530 --      VAR
20  531 --          pixelDelta:     VPoint;
20  532 --          newValue:       VCoordinate;
20  533 --
20  534 0- A   BEGIN
20  535 --          IF fConstrain[vhs] & (sBarValue <> fMaxTranslation.vh[vhs]) THEN
20  536 --              newValue := (sBarValue + fScrollUnit.vh[vhs] DIV 2) DIV fScrollUnit.vh[vhs] *
20  537 --                      fScrollUnit.vh[vhs]
20  538 --          ELSE
20  539 --              newValue := sBarValue;
20  540 --
20  541 --          pixelDelta := gZeroVPt;
20  542 --          pixelDelta.vh[vhs] := newValue - fTranslation.vh[vhs];
20  543 --          DoScroll(pixelDelta, kRedraw);
20  544 --          ScrollRelative := newValue - sBarValue;
20  545 -0 A   END {TScroller.ScrollRelative};
20  546 --
20  547 --      {-----------------------------------+
20  548 --      |   ScrollStep                      |
20  549 --      +-----------------------------------}
20  550 --      {$S MAScroll}
20  551 -- A   FUNCTION  TScroller.ScrollStep (vhs: VHSelect; partCode: INTEGER): VCoordinate;
20  552 --
20  553 --      VAR
20  554 --          scrollUnit:     INTEGER;
20  555 --          delta:          VPoint;
20  556 --          frameSize:      VCoordinate;
20  557 --
20  558 0- A   BEGIN
20  559 --          scrollUnit := fScrollUnit.vh[vhs];
20  560 --          delta := gZeroVPt;
20  561 1-          CASE partCode OF
20  562 --              inUpButton,
20  563 --              inDownButton:
20  564 --                  delta.vh[vhs] := scrollUnit;
20  565 --              inPageUp,
20  566 --              inPageDown:
20  567 2-                  BEGIN
20  568 --                  frameSize := fSize.vh[vhs];
20  569 --                  IF fConstrain[vhs] THEN
20  570 --                      frameSize := frameSize - (frameSize MOD scrollUnit);
20  571 --                  delta.vh[vhs] := frameSize;
20  572 -2                  END;
20  573 --      {$IFC qDebug}
20  574 --              OTHERWISE
20  575 2-                  BEGIN
20  576 --                  WRITELN('TScroller.ScrollStep: bad part code=', partCode:0);
20  577 --                  ProgramBreak('');
20  578 -2                  END;
20  579 --      {$ENDC}
20  580 -1              END;
20  581 --
20  582 --          IF (partCode = inUpButton) OR (partCode = inPageUp) THEN
20  583 --              delta.vh[vhs] := -delta.vh[vhs];
20  584 --          DoScroll(delta, kRedraw);
20  585 --          ScrollStep := delta.vh[vhs];
20  586 -0 A   END {TScroller.ScrollStep};
20  587 --
20  588 --      {-----------------------------------+
20  589 --      |   ScrollTo                        |
20  590 --      +-----------------------------------}
20  591 --      {$S MAScroll}
20  592 -- A   PROCEDURE TScroller.ScrollTo (h, v: VCoordinate; redraw: BOOLEAN);
20  593 --
```

```
20  594 0- A   BEGIN
20  595 --         ScrollBy(h - fTranslation.h, v - fTranslation.v, redraw);
20  596 -0 A   END {TScroller.ScrollTo};
20  597 --
20  598 --     {-----------------------------------+
20  599 --     |   SetScrollLimits                 |
20  600 --     +-----------------------------------}
20  601 --     {$S MANonRes}
20  602 -- A   PROCEDURE TScroller.SetScrollLimits (scrollLimit: VPoint; drawScrollBars: BOOLEAN);
20  603 --
20  604 --     VAR
20  605 --         maxCoord:           VCoordinate;
20  606 --         vhs:                VHSelect;
20  607 --         newTranslation:     VPoint;
20  608 --
20  609 0- A   BEGIN
20  610 --         fScrollLimit := scrollLimit;
20  611 --         newTranslation := fTranslation;
20  612 --         FOR vhs := v TO h DO
20  613 1-             BEGIN
20  614 --             maxCoord := Max(0, scrollLimit.vh[vhs] - fSize.vh[vhs]);
20  615 --             IF maxCoord <> fMaxTranslation.vh[vhs] THEN
20  616 2-                 BEGIN
20  617 --                 fMaxTranslation.vh[vhs] := maxCoord;
20  618 --                 IF fScrollBars[vhs] <> NIL THEN
20  619 --                     fScrollBars[vhs].SetLongMax(maxCoord, drawScrollBars);
20  620 --
20  621 --                 IF maxCoord < fTranslation.vh[vhs] THEN
20  622 --                     newTranslation.vh[vhs] := maxCoord;
20  623 -2                 END;
20  624 -1             END;
20  625 --
20  626 --         IF (newTranslation.h <> fTranslation.h) | (newTranslation.v <> fTranslation.v) THEN
20  627 1-             BEGIN
20  628 --             ScrollTo(newTranslation.h, newTranslation.v, kDontRedraw);
20  629 --             ForceRedraw;
20  630 -1             END;
20  631 -0 A   END {TScroller.SetScrollLimits};
20  632 --
20  633 --     {-----------------------------------+
20  634 --     |   SetScrollParameters             |
20  635 --     +-----------------------------------}
20  636 --     {$S MANonRes}
20  637 -- A   PROCEDURE TScroller.SetScrollParameters (horzUnits, vertUnits: VCoordinate;
20  638 --                                               horzConstraint, vertConstraint: BOOLEAN);
20  639 --
20  640 0- A   BEGIN
20  641 --         fScrollUnit.h := horzUnits;
20  642 --         fScrollUnit.v := vertUnits;
20  643 --         fConstrain[h] := horzConstraint;
20  644 --         fConstrain[v] := vertConstraint;
20  645 -0 A   END {TScroller.SetScrollParameters};
20  646 --
20  647 --     {-----------------------------------+
20  648 --     |   SubViewChangedSize              |
20  649 --     +-----------------------------------}
20  650 --     {$S MANonRes}
20  651 -- A   PROCEDURE TScroller.SubViewChangedSize (theSubView: TView; delta: VPoint); OVERRIDE;
20  652 --
20  653 0- A   BEGIN
20  654 --         SetScrollLimits(theSubView.fSize, kRedraw);
20  655 -0 A   END {TScroller.SubViewChangedSize};
20  656 --
20  657 --     {-----------------------------------+
20  658 --     |   SuperToLocal                    |
20  659 --     +-----------------------------------}
20  660 --     {$S MARes}
20  661 -- A   PROCEDURE TScroller.SuperToLocal (VAR thePoint: VPoint); OVERRIDE;
20  662 0- A   BEGIN
20  663 --         SubVPt(fLocation, thePoint);
20  664 --         AddVPt(fTranslation, thePoint);
20  665 -0 A   END {TScroller.SuperToLocal};
13 3008 --     {$I UMacApp.TControls.p}
```

```
21    1 --      {$P}
21    2 --      {UMacApp.TControls.p}
21    3 --      {Copyright © 1987-1988 by Apple Computer Inc.  All rights reserved.}
21    4 --
21    5 --      (*****************************************************************************************)
21    6 --      (*      G L O B A L   R O U T I N E S                                                    *)
21    7 --      (*****************************************************************************************)
21    8 --      {$S Main}
21    9 --      {-----------------------------------+
21   10 --      |   SBarActionProc                  |
21   11 --      +-----------------------------------}
21   12 -- A    PROCEDURE SBarActionProc (aCMgrControl: ControlHandle; partCode: INTEGER);
21   13 --      VAR
21   14 --          aScrollBar:     TScrollBar;
21   15 --          backwards:      BOOLEAN;
21   16 --
21   17 0- A    BEGIN
21   18 --          IF partCode <> 0 THEN
21   19 1-              BEGIN
21   20 --              aScrollBar := TScrollBar(GetCRefCon(aCMgrControl));
21   21 --              backwards := (partCode = inUpButton) | (partCode = inPageUp);
21   22 --              WITH aScrollBar DO
21   23 --                  IF (backwards & (fLongVal > fLongMin)) |
21   24 --                     ((NOT backwards) & (fLongVal < fLongMax)) THEN
21   25 --                      aScrollBar.TrackScrollBar(partCode);
21   26 -1              END;
21   27 -0 A    END ;
21   28 --
21   29 --      (*****************************************************************************************)
21   30 --      (*      T C o n t r o l T r a c k e r                                                    *)
21   31 --      (*****************************************************************************************)
21   32 --      {$S MASelCommand}
21   33 --      {-----------------------------------+
21   34 --      |   IControlTracker                 |
21   35 --      +-----------------------------------}
21   36 -- A    PROCEDURE TControlTracker.IControlTracker (theControl: TControl);
21   37 0- A    BEGIN
21   38 --          ICommand(cTrackingControl, NIL, theControl, NIL);
21   39 --      {$IFC qDebug}
21   40 --          IF theControl = NIL THEN
21   41 --              ProgramBreak('Control passed to IControlTracker can''t be NIL.');
21   42 --      {$ENDC}
21   43 --          fControl := theControl;
21   44 --          fCausesChange := FALSE;
21   45 --          fCanUndo := FALSE;
21   46 --          fTrackNonMovement := TRUE;
21   47 --          fViewConstrain := FALSE;
21   48 -0 A    END;
21   49 --
21   50 --      {$S MASelCommand}
21   51 --      {-----------------------------------+
21   52 --      |   TrackConstrain                  |
21   53 --      +-----------------------------------}
21   54 -- A    PROCEDURE TControlTracker.TrackConstrain (anchorPoint, previousPoint: VPoint;
21   55 --                                      VAR nextPoint: VPoint); OVERRIDE;
21   56 --
21   57 0- A    BEGIN
21   58 --          fControl.TrackConstrain(anchorPoint, previousPoint, nextPoint);
21   59 -0 A    END;
21   60 --
21   61 --      {$S MASelCommand}
21   62 --      {-----------------------------------+
21   63 --      |   TrackFeedback                   |
21   64 --      +-----------------------------------}
21   65 -- A    PROCEDURE TControlTracker.TrackFeedback (anchorPoint, nextPoint: VPoint;
21   66 --                                              turnItOn, mouseDidMove: BOOLEAN); OVERRIDE;
21   67 0- A    BEGIN
21   68 --          fControl.TrackFeedback(anchorPoint, nextPoint, turnItOn, mouseDidMove);
21   69 -0 A    END;
21   70 --
21   71 --      {$S MASelCommand}
21   72 --      {-----------------------------------+
21   73 --      |   TrackMouse                      |
21   74 --      +-----------------------------------}
21   75 -- A    FUNCTION  TControlTracker.TrackMouse (aTrackPhase: TrackPhase;
21   76 --                                      VAR anchorPoint, previousPoint, nextPoint: VPoint;
21   77 --                                      mouseDidMove: BOOLEAN): TCommand; OVERRIDE;
21   78 --
21   79 0- A    BEGIN
21   80 --          fControl.TrackMouse(aTrackPhase, anchorPoint, previousPoint, nextPoint, mouseDidMove);
21   81 --
21   82 --          IF (aTrackPhase = trackRelease) & (NOT fControl.ContainsMouse(nextPoint)) THEN
21   83 --              TrackMouse := gNoChanges
21   84 --          ELSE
21   85 --              TrackMouse := SELF;
21   86 -0 A    END;
21   87 --
21   88 --      (*****************************************************************************************)
21   89 --      (*      T C o n t r o l                                                                  *)
21   90 --      (*****************************************************************************************)
21   91 --
21   92 --      {$IFC qProceduralViews}
21   93 --      {$S MAOpen}
21   94 --      {-----------------------------------+
21   95 --      |   IControl                        |
21   96 --      +-----------------------------------}
21   97 -- A    PROCEDURE TControl.IControl (itsSuperView: TView; itsLocation, itsSize: VPoint;
21   98 --                              itsHSizeDet, itsVSizeDet: SizeDeterminer);
```

```
21   99 0- A   BEGIN
21  100 --     {$IFC qDebug}
21  101 --         IF (itsSize.h > kMaxCoord) | (itsSize.v > kMaxCoord) THEN
21  102 --             ProgramBreak('TControl.IControl:  Size is too large.');
21  103 --     {$ENDC}
21  104 --
21  105 --         fTextStyle := gSystemStyle;
21  106 --         IView(NIL, itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
21  107 --         fDefChoice := mOKHit;
21  108 --         fHilite := FALSE;
21  109 --         fDimmed := FALSE;
21  110 --         fSizeable := TRUE;
21  111 --         fAdornment := [];
21  112 --         fPenSize := Point($00010001);
21  113 --     {$H-}
21  114 --         fInset := gZeroRect;
21  115 --     {$H+}
21  116 --         fTextStyle := gSystemStyle;
21  117 --         fDismissesDialog := FALSE;
21  118 -0 A   END;
21  119 --     {$ENDC}
21  120 --
21  121 --
21  122 --     {$IFC qTemplateViews}
21  123 --     {$S MAOpen}
21  124 --     {-----------------------------------+
21  125 --     |   IRes                            |
21  126 --     +-----------------------------------}
21  127 -- A   PROCEDURE TControl.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
21  128 --
21  129 --     VAR
21  130 --         aTextStyle:     TextStyle;
21  131 --
21  132 0- A   BEGIN
21  133 --         { Don't pass the document to INHERITED IRes.  This avoids putting the control on the
21  134 --             document's view list.}
21  135 --         INHERITED IRes(NIL, itsSuperView, itsParams);
21  136 --         fDocument := itsDocument;
21  137 --
21  138 --         WITH ControlTemplatePtr(itsParams)^ DO
21  139 1-             BEGIN
21  140 --             fDefChoice := mOKHit;       {??? This seems questionable}
21  141 --             fAdornment := CntlAdornment(itsAdornment);
21  142 --             fSizeable := isSizable;
21  143 --             fHilite := isHilited;
21  144 --             fDimmed := isDimmed;
21  145 --             fDismissesDialog := canDismiss;
21  146 --             fInset := itsInset;
21  147 --             fPenSize := itsPenSize;
21  148 --             SetTextStyle(aTextStyle, GetFontNum(itsFontName), itsTextFace, itsTextSize, itsTextColor);
21  149 --             fTextStyle := aTextStyle;
21  150 --
21  151 --             OffsetPtrWStr(itsParams, SIZEOF(ControlTemplate));
21  152 -1             END;
21  153 -0 A   END;
21  154 --     {$ENDC}
21  155 --
21  156 --
21  157 --     {$IFC qWriteTemplates}
21  158 --     {$S MAWriteRes}
21  159 --     {-----------------------------------+
21  160 --     |   WRes                            |
21  161 --     +-----------------------------------}
21  162 -- A   PROCEDURE TControl.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21  163 --     VAR
21  164 --         theSize:    INTEGER;
21  165 --         theFont:    Str255;
21  166 --         cnPtr:      ControlTemplatePtr;
21  167 --
21  168 0- A   BEGIN
21  169 --         INHERITED WRes(theResource, itsParams);
21  170 --
21  171 --         theSize := fTextStyle.tsSize;
21  172 --         GetPortFontInfo(fTextStyle.tsFont, theFont, theSize);
21  173 --
21  174 --         cnPtr := ControlTemplatePtr(ExpandPtrWStr(theResource, itsParams,
21  175 --           SIZEOF(ControlTemplate), LENGTH(theFont)));
21  176 --
21  177 --         WITH cnPtr^ DO
21  178 1-             BEGIN
21  179 --             itsAdornment := SignedByte(fAdornment);
21  180 --             itsPenSize := fPenSize;
21  181 --             isSizable := fSizeable;
21  182 --             isDimmed := fDimmed;
21  183 --             isHilited := fHilite;
21  184 --             canDismiss := fDismissesDialog;
21  185 --             itsInset := fInset;
21  186 --             WITH fTextStyle DO
21  187 2-                 BEGIN
21  188 --                 itsTextFace := tsFace;
21  189 --                 itsTextSize := theSize;
21  190 --                 itsTextColor := tsColor;
21  191 -2                 END;
21  192 --             (* itsFontName := theFont; *)
21  193 --             CopyStr255(theFont, PRStr(itsFontName));
21  194 -1             END;
21  195 -0 A   END;
21  196 --     {$ENDC}
21  197 --
```

```
21  198 --
21  199 --      {$IFC qWriteTemplates}
21  200 --      {$S MAWriteRes}
21  201 --      {-------------------------------------+
21  202 --      |   WriteRes                          |
21  203 --      +-------------------------------------}
21  204 -- A    PROCEDURE TControl.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21  205 0- A    BEGIN
21  206 --          gWResSignature := 'cntl';   gWResType := 'TControl';
21  207 --          WRes(theResource, itsParams);
21  208 -0 A    END;
21  209 --      {$ENDC}
21  210 --
21  211 --
21  212 --      {$S MAActivate}
21  213 --      {-------------------------------------+
21  214 --      |   Activate                          |
21  215 --      +-------------------------------------}
21  216 -- A    PROCEDURE TControl.Activate (entering: BOOLEAN); OVERRIDE;
21  217 --
21  218 0- A    BEGIN
21  219 --          { Don't do anything }
21  220 -0 A    END;
21  221 --
21  222 --      {$S MARes}
21  223 --      {-------------------------------------+
21  224 --      |   ComputeSize                       |
21  225 --      +-------------------------------------}
21  226 -- A    PROCEDURE TControl.ComputeSize (VAR newSize: VPoint); OVERRIDE;
21  227 --      VAR
21  228 --          deltaSz:    Point;
21  229 --
21  230 0- A    BEGIN
21  231 --          INHERITED ComputeSize(newSize);
21  232 --          IF NOT fSizeable THEN
21  233 1-              BEGIN                               { …we need to adjust the bottom/right offset    }
21  234 --              deltaSz.h := newSize.h - fSize.h;   { Controls cannot be bigger than QD space.      }
21  235 --              deltaSz.v := newSize.v - fSize.v;
21  236 --
21  237 --              IF LONGINT(deltaSz) <> 0 THEN       { If view is going to change size, then…        }
21  238 --              {$H-}                                  { …adjust bottom/right offset so it…         }
21  239 --                  AddPt(deltaSz, fInset.botRight);    { …doesn't change control's actual size      }
21  240 --              {$H+}
21  241 -1              END;
21  242 -0 A    END;
21  243 --
21  244 --      {$S MARes}
21  245 --      {-------------------------------------+
21  246 --      |   ContainsMouse                     |
21  247 --      +-------------------------------------}
21  248 -- A    FUNCTION TControl.ContainsMouse (theMouse: VPoint): BOOLEAN; OVERRIDE;
21  249 --      VAR
21  250 --          aRect:      Rect;
21  251 0- A    BEGIN
21  252 --          ControlArea(aRect);
21  253 --          ContainsMouse := PtInRect(VPtToPt(theMouse), aRect);
21  254 -0 A    END;
21  255 --
21  256 --      {$S MARes}
21  257 --      {-------------------------------------+
21  258 --      |   ControlArea                       |
21  259 --      +-------------------------------------}
21  260 -- A    PROCEDURE TControl.ControlArea (VAR theArea: Rect);
21  261 0- A    BEGIN
21  262 --          {$H-}
21  263 --          WITH fInset DO
21  264 --              SetRect(theArea, left, top, fSize.h - right, fSize.v - bottom);
21  265 --          {$H+}
21  266 --          IF adnShadow IN fAdornment THEN
21  267 --              SubPt(fPenSize, theArea.botRight);
21  268 -0 A    END;
21  269 --
21  270 --      {$S MARes}
21  271 --      {-------------------------------------+
21  272 --      |   Dim                               |
21  273 --      +-------------------------------------}
21  274 -- A    PROCEDURE TControl.Dim;
21  275 --      VAR
21  276 --          area:   Rect;
21  277 0- A    BEGIN
21  278 --          ControlArea(area);
21  279 --          PenPat(gray);
21  280 --          PenMode(patBic);
21  281 --          PaintRect(area);
21  282 -0 A    END;
21  283 --
21  284 --      {$S MANonRes}
21  285 --      {-------------------------------------+
21  286 --      |   DimState                          |
21  287 --      +-------------------------------------}
21  288 -- A    PROCEDURE TControl.DimState (state, redraw: BOOLEAN);
21  289 0- A    BEGIN
21  290 --          IF state <> fDimmed THEN
21  291 1-              BEGIN
21  292 --              fDimmed := state;
21  293 --              IF redraw THEN
21  294 --                  DrawContents;                           { Draw change immediately }
21  295 -1              END;
21  296 -0 A    END;
```

```
21  297 --
21  298 --      {$S MASelCommand}
21  299 --      {-----------------------------------+
21  300 --      |   DoMouseCommand                  |
21  301 --      +-----------------------------------}
21  302 -- A    FUNCTION  TControl.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
21  303 --                                         VAR hysteresis: Point): TCommand; OVERRIDE;
21  304 --      VAR
21  305 --          aControlTracker:    TControlTracker;
21  306 0- A    BEGIN
21  307 --          NEW(aControlTracker);
21  308 --          FailNIL(aControlTracker);
21  309 --          aControlTracker.IControlTracker(SELF);
21  310 --          DoMouseCommand := aControlTracker;
21  311 -0 A    END;
21  312 --
21  313 --      {$S MARes}
21  314 --      {-----------------------------------+
21  315 --      |   Draw                            |
21  316 --      +-----------------------------------}
21  317 -- A    PROCEDURE TControl.Draw (area: Rect); OVERRIDE;
21  318 --      VAR
21  319 --          qdExtent:   Rect;
21  320 --
21  321 0- A    BEGIN
21  322 --          IF SignedByte(fAdornment) <> 0 THEN
21  323 1-              BEGIN
21  324 --              GetQDExtent(qdExtent);
21  325 --              Adorn(qdExtent, fPenSize, fAdornment);
21  326 -1              END;
21  327 --          IF fDimmed THEN
21  328 --              Dim;
21  329 --          IF fHilite THEN
21  330 --              Hilite;
21  331 -0 A    END;
21  332 --
21  333 --      {$S MARes}
21  334 --      {-----------------------------------+
21  335 --      |   Focus                           |
21  336 --      +-----------------------------------}
21  337 -- A    FUNCTION  TControl.Focus: BOOLEAN; OVERRIDE;
21  338 0- A    BEGIN
21  339 --          IF INHERITED Focus THEN
21  340 1-              BEGIN
21  341 --              SetPortTextStyle(fTextStyle);
21  342 --              Focus := TRUE;
21  343 -1              END
21  344 --          ELSE
21  345 --              Focus := FALSE;
21  346 -0 A    END;
21  347 --
21  348 --      {$S MARes}
21  349 --      {-----------------------------------+
21  350 --      |   Hilite                          |
21  351 --      +-----------------------------------}
21  352 -- A    PROCEDURE TControl.Hilite;
21  353 --      VAR
21  354 --          aRect:      Rect;
21  355 0- A    BEGIN
21  356 --          ControlArea(aRect);
21  357 --          InvertRect(aRect);
21  358 -0 A    END;
21  359 --
21  360 --      {$S MANonRes}
21  361 --      {-----------------------------------+
21  362 --      |   HiliteState                     |
21  363 --      +-----------------------------------}
21  364 -- A    PROCEDURE TControl.HiliteState (state, redraw: BOOLEAN);
21  365 0- A    BEGIN
21  366 --          IF state <> fHilite THEN
21  367 1-              BEGIN
21  368 --              fHilite := state;
21  369 --              IF redraw & Focus THEN
21  370 --                  Hilite;
21  371 -1              END;
21  372 -0 A    END;
21  373 --
21  374 --      {$S MANonRes}
21  375 --      {-----------------------------------+
21  376 --      |   Inset                           |
21  377 --      +-----------------------------------}
21  378 -- A    PROCEDURE TControl.Inset (dh, dv: INTEGER; redraw: BOOLEAN);
21  379 0- A    BEGIN
21  380 --          IF fSizeable THEN
21  381 1-              BEGIN
21  382 --          {$H-}
21  383 --              OffsetRect(fInset, dh, dv);
21  384 --          {$H+}
21  385 --              Resize(fSize.h, fSize.v, redraw);
21  386 -1              END;
21  387 -0 A    END;
21  388 --
21  389 --      {$S MANonRes}
21  390 --      {-----------------------------------+
21  391 --      |   InstallColor                    |
21  392 --      +-----------------------------------}
21  393 -- A    PROCEDURE TControl.InstallColor(theColor: RGBColor; redraw: BOOLEAN);
21  394 0- A    BEGIN
21  395 --          fTextStyle.tsColor := theColor;
```

```
21  396 --          IF redraw THEN
21  397 --              DrawContents;
21  398 -0 A    END;
21  399 --
21  400 --      {$S MANonRes}
21  401 --      {-----------------------------------+
21  402 --      |   InstallTextStyle                |
21  403 --      +-----------------------------------}
21  404 -- A    PROCEDURE TControl.InstallTextStyle (theTextStyle: TextStyle; redraw: BOOLEAN);
21  405 0- A    BEGIN
21  406 --          fTextStyle := theTextStyle;
21  407 --          IF redraw THEN
21  408 --              DrawContents;
21  409 -0 A    END;
21  410 --
21  411 --      {$S MARes}
21  412 --      {-----------------------------------+
21  413 --      |   IsDimmed                        |
21  414 --      +-----------------------------------}
21  415 -- A    FUNCTION  TControl.IsDimmed: BOOLEAN;
21  416 0- A    BEGIN
21  417 --          IsDimmed := fDimmed;
21  418 -0 A    END;
21  419 --
21  420 --      {$S MANonRes}
21  421 --      {-----------------------------------+
21  422 --      |   Resize                          |
21  423 --      +-----------------------------------}
21  424 -- A    PROCEDURE TControl.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
21  425 --
21  426 0- A    BEGIN
21  427 --          { If we need to invalidate, then invalidate the whole view rather than
21  428 --              the difference between the old and new size. }
21  429 --          INHERITED Resize(width, height, invalidate);
21  430 --          IF invalidate THEN
21  431 --              ForceRedraw;
21  432 -0 A    END;
21  433 --
21  434 --      {$S MANonRes}
21  435 --      {-----------------------------------+
21  436 --      |   SetInset                        |
21  437 --      +-----------------------------------}
21  438 -- A    PROCEDURE TControl.SetInset(newInset: Rect; redraw: BOOLEAN);
21  439 0- A    BEGIN
21  440 --          IF fSizeable THEN
21  441 1-              BEGIN
21  442 --              fInset := newInset;
21  443 --              Resize(fSize.h, fSize.v, redraw);
21  444 -1              END;
21  445 -0 A    END;
21  446 --
21  447 --      {$S MASelCommand}
21  448 --      {-----------------------------------+
21  449 --      |   TrackConstrain                  |
21  450 --      +-----------------------------------}
21  451 -- A    PROCEDURE TControl.TrackConstrain(anchorPoint, previousPoint: VPoint;
21  452 --                                        VAR nextPoint: VPoint);
21  453 0- A    BEGIN
21  454 -0 A    END;
21  455 --
21  456 --      {$S MASelCommand}
21  457 --      {-----------------------------------+
21  458 --      |   TrackFeedback                   |
21  459 --      +-----------------------------------}
21  460 -- A    PROCEDURE TControl.TrackFeedback(anchorPoint, nextPoint: VPoint;
21  461 --                                       turnItOn, mouseDidMove: BOOLEAN);
21  462 0- A    BEGIN
21  463 -0 A    END;
21  464 --
21  465 --      {$S MASelCommand}
21  466 --      {-----------------------------------+
21  467 --      |   TrackMouse                      |
21  468 --      +-----------------------------------}
21  469 -- A    PROCEDURE TControl.TrackMouse(aTrackPhase: TrackPhase;
21  470 --                                    VAR anchorPoint, previousPoint, nextPoint: VPoint;
21  471 --                                    mouseDidMove: BOOLEAN);
21  472 0- A    BEGIN
21  473 1-          CASE aTrackPhase OF
21  474 --              trackPress:
21  475 --                  HiliteState(TRUE, kRedraw);
21  476 --              trackMove:
21  477 --                  HiliteState(ContainsMouse(nextPoint), kRedraw);
21  478 --              trackRelease:
21  479 2-                  BEGIN
21  480 --                  IF fHilite THEN
21  481 --                      HiliteState(FALSE, kRedraw);
21  482 --                  IF ContainsMouse(nextPoint) THEN
21  483 --                      DoChoice(SELF, fDefChoice);
21  484 -2                  END;
21  485 -1              END;
21  486 -0 A    END;
21  487 --
21  488 --      {$S MARes}
21  489 --      {-----------------------------------+
21  490 --      |   Validate                        |
21  491 --      +-----------------------------------}
21  492 -- A    FUNCTION  TControl.Validate: BOOLEAN;
21  493 0- A    BEGIN
21  494 --          Validate := TRUE;
```

```
21  495 -0 A   END;
21  496 --
21  497 --     {$IFC qDebug}
21  498 --         {$S MAFields}
21  499 --         {-----------------------------------+
21  500 --         |   Fields                          |
21  501 --         +-----------------------------------}
21  502 -- A          PROCEDURE TControl.Fields (PROCEDURE DoToField (fieldName: Str255;
21  503 --                                                            fieldAddr: Ptr;
21  504 --                                                            fieldType: INTEGER)); OVERRIDE;
21  505 0- A       BEGIN
21  506 --             DoToField('TControl', NIL, bClass);
21  507 --             DoToField('fDefChoice', @fDefChoice, bInteger);
21  508 --             DoToField('fHilite', @fHilite, bBoolean);
21  509 --             DoToField('fDimmed', @fDimmed, bBoolean);
21  510 --             DoToField('fSizeable', @fSizeable, bBoolean);
21  511 --             DoToField('fDismissesDialog', @fDismissesDialog, bBoolean);
21  512 --             DoToField('fAdornment', @fAdornment, bCntlAdornment);
21  513 --             DoToField('fPenSize', @fPenSize, bPoint);
21  514 --             DoToField('fInset', @fInset, bRect);
21  515 --             {$H-}
21  516 --             TextStyleFields('fTextStyle', fTextStyle, DoToField);
21  517 --             {$H+}
21  518 --             INHERITED Fields(DoToField);
21  519 -0 A       END;
21  520 --     {$ENDC}
21  521 --
21  522 --
21  523 --     (*****************************************************************************************)
21  524 --     (*      T C t l M g r                                                                    *)
21  525 --     (*****************************************************************************************)
21  526 --
21  527 --     {$IFC qProceduralViews}
21  528 --     {$S MAOpen}
21  529 --     {-----------------------------------+
21  530 --     |   ICtlMgr                         |
21  531 --     +-----------------------------------}
21  532 -- A   PROCEDURE TCtlMgr.ICtlMgr (itsSuperView: TView; itsLocation, itsSize: VPoint;
21  533 --                                itsHSizeDet, itsVSizeDet: SizeDeterminer;
21  534 --                                itsTitle: Str255; itsVal, itsMin, itsMax, itsProcID: INTEGER);
21  535 --     VAR
21  536 --         itsBounds:      Rect;
21  537 --
21  538 0- A   BEGIN
21  539 --         fCMgrControl := NIL;
21  540 --         IControl(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet);
21  541 --         SetRect(itsBounds, 0, 0, fSize.h, fSize.v);
21  542 --         CreateCMgrControl(itsBounds, itsTitle, itsVal, itsMin, itsMax, itsProcID);
21  543 -0 A   END;
21  544 --     {$ENDC}
21  545 --
21  546 --
21  547 --     {$IFC qTemplateViews}
21  548 --     {$S MAOpen}
21  549 --     {-----------------------------------+
21  550 --     |   IRes                            |
21  551 --     +-----------------------------------}
21  552 -- A   PROCEDURE TCtlMgr.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr);
21  553 --
21  554 0- A   BEGIN
21  555 --         fCMgrControl := NIL;
21  556 --         INHERITED IRes(itsDocument, itsSuperView, itsParams);
21  557 --         {The subclass must create the Control Manager control}
21  558 -0 A   END {TCtlMgr.IRes};
21  559 --     {$ENDC}
21  560 --
21  561 --
21  562 --     {$IFC qWriteTemplates}
21  563 --     {$S MAWriteRes}
21  564 --     {-----------------------------------+
21  565 --     |   WRes                            |
21  566 --     +-----------------------------------}
21  567 -- A   PROCEDURE TCtlMgr.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21  568 0- A   BEGIN
21  569 --         INHERITED WRes(theResource, itsParams);
21  570 -0 A   END;
21  571 --     {$ENDC}
21  572 --
21  573 --
21  574 --     {$IFC qWriteTemplates}
21  575 --     {$S MAWriteRes}
21  576 --     {-----------------------------------+
21  577 --     |   WriteRes                        |
21  578 --     +-----------------------------------}
21  579 -- A   PROCEDURE TCtlMgr.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21  580 0- A   BEGIN
21  581 --         { We don't write TCtlMgrs_ }
21  582 -0 A   END;
21  583 --     {$ENDC}
21  584 --
21  585 --
21  586 --     {$S MAClose}
21  587 --     {-----------------------------------+
21  588 --     |   Free                            |
21  589 --     +-----------------------------------}
21  590 -- A   PROCEDURE TCtlMgr.Free; OVERRIDE;
21  591 --
21  592 0- A   BEGIN
21  593 --         IF fCMgrControl <> NIL THEN
```

```
21  594 --              DisposeControl(fCMgrControl);
21  595 --          INHERITED Free;
21  596 -0 A    END;
21  597 --
21  598 --      {$S MANonRes}
21  599 --      {-----------------------------------+
21  600 --      |   BeInPort                        |
21  601 --      +-----------------------------------}
21  602 -- A    PROCEDURE TCtlMgr.BeInPort (itsPort: GrafPtr); OVERRIDE;
21  603 --
21  604 0- A    BEGIN
21  605 --          IF fCMgrControl <> NIL THEN
21  606 --              WITH fCMgrControl^^ DO
21  607 --                  IF itsPort = NIL THEN
21  608 1-                      BEGIN
21  609 --                      contrlVis := ORD(FALSE);        { So Ctrl Mgr doesn't try to draw it        }
21  610 --                      contrlOwner := WindowPtr(gWorkPort);
21  611 -1                      END
21  612 --                  ELSE
21  613 1-                      BEGIN
21  614 --                      contrlVis := ORD(TRUE);
21  615 --                      contrlOwner := WindowPtr(itsPort);
21  616 -1                      END;
21  617 -0 A    END;
21  618 --
21  619 --      {$S MAOpen}
21  620 --      {-----------------------------------+
21  621 --      |   CreateCMgrControl               |
21  622 --      +-----------------------------------}
21  623 -- A    PROCEDURE TCtlMgr.CreateCMgrControl (itsBounds: Rect; itsTitle: Str255;
21  624 --                                          itsValue, itsMin, itsMax, itsProcID: INTEGER);
21  625 --
21  626 --      VAR
21  627 --          itsPort:        GrafPtr;
21  628 --          aCMgrControl:   ControlHandle;
21  629 --          fi:             FailInfo;
21  630 --
21  631 -- B        PROCEDURE HandleFailure(error: OSErr; message: LONGINT);
21  632 0- B      BEGIN
21  633 --              Free;
21  634 -0 B      END;
21  635 --
21  636 0- A    BEGIN
21  637 --          itsPort := GetGrafPort;
21  638 --          IF itsPort = NIL THEN
21  639 --              itsPort := gWorkPort;
21  640 --          CatchFailures(fi, HandleFailure);
21  641 --          aCMgrControl := NewControl(itsPort, itsBounds, itsTitle, FALSE,
21  642 --                                     itsValue, itsMin, itsMax, BOR(itsProcID, useWFont),
21  643 --                                     LONGINT(SELF));
21  644 --          FailNIL(aCMgrControl);
21  645 --          Success(fi);
21  646 --          WITH WindowPeek(itsPort)^ DO            { Keep control off Window's control list        }
21  647 --              controlList := controlList^^.nextControl;
21  648 --
21  649 --          aCMgrControl^^.contrlVis := ORD(itsPort <> gWorkPort);
21  650 --          IF fDimmed THEN
21  651 --              aCMgrControl^^.contrlHilite := 255;
21  652 --          fCMgrControl := aCMgrControl;
21  653 -0 A    END {TCtlMgr.CreateCMgrControl};
21  654 --
21  655 --      {$S MANonRes}
21  656 --      {-----------------------------------+
21  657 --      |   DimState                        |
21  658 --      +-----------------------------------}
21  659 -- A    PROCEDURE TCtlMgr.DimState (state, redraw: BOOLEAN); OVERRIDE;
21  660 --
21  661 -- B        PROCEDURE SetHilite;
21  662 0- B      BEGIN
21  663 --              HiliteControl(fCMgrControl, ORD(state) * 255);
21  664 --              fDimmed := state;
21  665 -0 B      END;
21  666 --
21  667 0- A    BEGIN
21  668 --          IF fDimmed <> state THEN
21  669 --              WhileFocused(SetHilite, redraw);
21  670 -0 A    END;
21  671 --
21  672 --      {$S MASelCommand}
21  673 --      {-----------------------------------+
21  674 --      |   DoMouseCommand                  |
21  675 --      +-----------------------------------}
21  676 -- A    FUNCTION TCtlMgr.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
21  677 --                                      VAR hysteresis: Point): TCommand; OVERRIDE;
21  678 --
21  679 0- A    BEGIN
21  680 --          IF TestControl(fCMgrControl, theMouse) <> 0 THEN
21  681 --              IF TrackControl(fCMgrControl, theMouse, NIL) <> 0 THEN
21  682 --                  DoChoice(SELF, fDefChoice);
21  683 --          DoMouseCommand := gNoChanges;
21  684 -0 A    END {TCtlMgr.DoMouseCommand};
21  685 --
21  686 --      {$S MARes}
21  687 --      {-----------------------------------+
21  688 --      |   Draw                            |
21  689 --      +-----------------------------------}
21  690 -- A    PROCEDURE TCtlMgr.Draw (area: Rect); OVERRIDE;
21  691 --
21  692 --      VAR
```

```
21  693 --          savedOwner:     WindowPtr;
21  694 --          qdExtent:       Rect;
21  695 --
21  696 0- A    BEGIN
21  697 --          IF fCMgrControl^^.contrlVis <> 0 THEN
21  698 1-              BEGIN                                   { Set the port in case we're printing       }
21  699 --              savedOwner := fCMgrControl^^.contrlOwner;
21  700 --              fCMgrControl^^.contrlOwner := WindowPtr(thePort);
21  701 --
21  702 --              PenNormal; {NECESSARY?}
21  703 --
21  704 --              fCMgrControl^^.contrlVis := 0;          { Force ShowControl to redraw               }
21  705 --              ShowControl(fCMgrControl);
21  706 --
21  707 --              fCMgrControl^^.contrlOwner := savedOwner;
21  708 -1              END;
21  709 --
21  710 --          IF SignedByte(fAdornment) <> 0 THEN
21  711 1-              BEGIN
21  712 --              GetQDExtent(qdExtent);
21  713 --              Adorn(qdExtent, fPenSize, fAdornment);
21  714 -1              END;
21  715 -0 A    END;
21  716 --
21  717 --      {$S MARes}
21  718 --      {-----------------------------------+
21  719 --      |   GetMax                          |
21  720 --      +-----------------------------------}
21  721 -- A    FUNCTION  TCtlMgr.GetMax: INTEGER;
21  722 --
21  723 0- A    BEGIN
21  724 --          GetMax := GetCtlMax(fCMgrControl)
21  725 -0 A    END;
21  726 --
21  727 --      {$S MARes}
21  728 --      {-----------------------------------+
21  729 --      |   GetMin                          |
21  730 --      +-----------------------------------}
21  731 -- A    FUNCTION  TCtlMgr.GetMin: INTEGER;
21  732 --
21  733 0- A    BEGIN
21  734 --          GetMin := GetCtlMin(fCMgrControl)
21  735 -0 A    END;
21  736 --
21  737 --      {$S MARes}
21  738 --      {-----------------------------------+
21  739 --      |   GetText                         |
21  740 --      +-----------------------------------}
21  741 -- A    PROCEDURE TCtlMgr.GetText (VAR theText: Str255);
21  742 0- A    BEGIN
21  743 --          theText := fCMgrControl^^.contrlTitle;
21  744 -0 A    END;
21  745 --
21  746 --      {$S MARes}
21  747 --      {-----------------------------------+
21  748 --      |   GetVal                          |
21  749 --      +-----------------------------------}
21  750 -- A    FUNCTION  TCtlMgr.GetVal: INTEGER;
21  751 --
21  752 0- A    BEGIN
21  753 --          GetVal := GetCtlValue(fCMgrControl)
21  754 -0 A    END;
21  755 --
21  756 --      {$S MANonRes}
21  757 --      {-----------------------------------+
21  758 --      |   HiliteState                     |
21  759 --      +-----------------------------------}
21  760 -- A    PROCEDURE TCtlMgr.HiliteState (state, redraw: BOOLEAN); OVERRIDE;
21  761 --
21  762 -- B        PROCEDURE SetHilite;
21  763 0- B        BEGIN
21  764 --              HiliteControl(fCMgrControl, ORD(state) * 10);
21  765 -0 B        END;
21  766 --
21  767 0- A    BEGIN
21  768 --          IF fHilite <> state THEN
21  769 --              WhileFocused(SetHilite, redraw);
21  770 --          fHilite := state;
21  771 -0 A    END;
21  772 --
21  773 --      {$S MANonRes}
21  774 --      {-----------------------------------+
21  775 --      |   Resize                          |
21  776 --      +-----------------------------------}
21  777 -- A    PROCEDURE TCtlMgr.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
21  778 --
21  779 -- B        PROCEDURE ResizeCMgrControl;
21  780 --          VAR
21  781 --              aRect:      Rect;
21  782 0- B        BEGIN
21  783 --              SetRect(aRect, fInset.left, fInset.top, width - fInset.right, height - fInset.bottom);
21  784 --              MoveControl(fCMgrControl, aRect.left, aRect.top);
21  785 --              SizeControl(fCMgrControl, ABS(aRect.right-aRect.left), ABS(aRect.bottom-aRect.top));
21  786 -0 B        END;
21  787 --
21  788 0- A    BEGIN
21  789 --          IF fSizeable & (fCMgrControl <> NIL) THEN
21  790 --              WhileFocused(ResizeCMgrControl, kDontRedraw);
21  791 --          INHERITED Resize(width, height, invalidate);
```

```
21  792 -0 A   END;
21  793 --
21  794 --     {$S MARes}
21  795 --     {-----------------------------------+
21  796 --     |   SetMax                          |
21  797 --     +-----------------------------------}
21  798 -- A   PROCEDURE TCtlMgr.SetMax (itsMax: INTEGER; redraw: BOOLEAN);
21  799 --
21  800 --         {--------------------------------+
21  801 --         |   DoSetMax                     |
21  802 --         +--------------------------------}
21  803 -- B       PROCEDURE DoSetMax;
21  804 0- B       BEGIN
21  805 --            SetCtlMax(fCMgrControl, itsMax);
21  806 -0 B       END;
21  807 --
21  808 0- A   BEGIN
21  809 --         WhileFocused(DoSetMax, redraw);
21  810 -0 A   END;
21  811 --
21  812 --     {$S MARes}
21  813 --     {-----------------------------------+
21  814 --     |   SetMin                          |
21  815 --     +-----------------------------------}
21  816 -- A   PROCEDURE TCtlMgr.SetMin (itsMin: INTEGER; redraw: BOOLEAN);
21  817 --
21  818 --         {--------------------------------+
21  819 --         |   DoSetMin                     |
21  820 --         +--------------------------------}
21  821 -- B       PROCEDURE DoSetMin;
21  822 0- B       BEGIN
21  823 --            SetCtlMin(fCMgrControl, itsMin);
21  824 -0 B       END;
21  825 --
21  826 0- A   BEGIN
21  827 --         WhileFocused(DoSetMin, redraw);
21  828 -0 A   END;
21  829 --
21  830 --     {$S MANonRes}
21  831 --     {-----------------------------------+
21  832 --     |   SetText                         |
21  833 --     +-----------------------------------}
21  834 -- A   PROCEDURE TCtlMgr.SetText (itsText: Str255; redraw: BOOLEAN);
21  835 --
21  836 --         {--------------------------------+
21  837 --         |   DoSetText                    |
21  838 --         +--------------------------------}
21  839 -- B       PROCEDURE DoSetText;
21  840 0- B       BEGIN
21  841 --            SetCTitle(fCMgrControl, itsText);
21  842 -0 B       END;
21  843 --
21  844 0- A   BEGIN
21  845 --         WhileFocused(DoSetText, redraw);
21  846 -0 A   END;
21  847 --
21  848 --     {$S MARes}
21  849 --     {-----------------------------------+
21  850 --     |   SetVal                          |
21  851 --     +-----------------------------------}
21  852 -- A   PROCEDURE TCtlMgr.SetVal (newVal: INTEGER; redraw: BOOLEAN);
21  853 --
21  854 --         {--------------------------------+
21  855 --         |   DoSetVal                     |
21  856 --         +--------------------------------}
21  857 -- B       PROCEDURE DoSetVal;
21  858 0- B       BEGIN
21  859 --            SetCtlValue(fCMgrControl, newVal);
21  860 -0 B       END;
21  861 --
21  862 0- A   BEGIN
21  863 --         WhileFocused(DoSetVal, redraw);
21  864 -0 A   END;
21  865 --
21  866 --     {$S MARes}
21  867 --     {-----------------------------------+
21  868 --     |   SetValues                       |
21  869 --     +-----------------------------------}
21  870 -- A   PROCEDURE TCtlMgr.SetValues (itsVal, itsMin, itsMax: INTEGER; redraw: BOOLEAN);
21  871 --
21  872 --         {--------------------------------+
21  873 --         |   DoSetValues                  |
21  874 --         +--------------------------------}
21  875 -- B       PROCEDURE DoSetValues;
21  876 0- B       BEGIN
21  877 --            SetCtlMin(fCMgrControl, itsMin);
21  878 --            SetCtlMax(fCMgrControl, itsMax);
21  879 --            SetCtlValue(fCMgrControl, itsVal);
21  880 -0 B       END;
21  881 --
21  882 0- A   BEGIN
21  883 --         WhileFocused(DoSetValues, redraw);
21  884 -0 A   END;
21  885 --
21  886 --     {$S MARes}
21  887 --     {-----------------------------------+
21  888 --     |   WhileFocused                    |
21  889 --     +-----------------------------------}
21  890 -- A   PROCEDURE TCtlMgr.WhileFocused (PROCEDURE DoToControl; redraw: BOOLEAN);
```

```
21  891 --
21  892 --      VAR
21  893 --          wasVisible:     Byte;
21  894 --
21  895 0- A    BEGIN
21  896 --          IF fCMgrControl <> NIL THEN
21  897 --              IF redraw THEN
21  898 1-                  BEGIN
21  899 --                  IF Focus THEN ;             { Attempt to focus }
21  900 --                  DoToControl;                { Do it even if we couldn't focus }
21  901 -1                  END
21  902 --              ELSE
21  903 1-                  BEGIN
21  904 --                  wasVisible := fCMgrControl^^.contrlVis;
21  905 --                  fCMgrControl^^.contrlVis := 0;
21  906 --                  DoToControl;
21  907 --                  fCMgrControl^^.contrlVis := wasVisible;
21  908 --      {$IFC FALSE}
21  909 --                  GetFocus;
21  910 --                  ClipRect(gZeroRect);
21  911 --                  DoToControl;
21  912 --                  SetFocus;
21  913 --      {$ENDC}
21  914 -1                  END;
21  915 -0 A    END;
21  916 --
21  917 --      {$IFC qDebug}
21  918 --          {$S MAFields}
21  919 --          {-----------------------------------+
21  920 --          |   Fields                          |
21  921 --          +-----------------------------------}
21  922 -- A      PROCEDURE TCtlMgr.Fields (PROCEDURE DoToField (fieldName: Str255;
21  923 --                                                        fieldAddr: Ptr;
21  924 --                                                        fieldType: INTEGER)); OVERRIDE;
21  925 0- A      BEGIN
21  926 --          DoToField('TCtlMgr', NIL, bClass);
21  927 --          DoToField('fCMgrControl', @fCMgrControl, bControlHandle);
21  928 --          INHERITED Fields(DoToField);
21  929 -0 A      END;
21  930 --      {$ENDC}
21  931 --
21  932 --      (*****************************************************************************************)
21  933 --      (*      T S c r o l l B a r                                                              *)
21  934 --      (*****************************************************************************************)
21  935 --
21  936 --      {$IFC qProceduralViews}
21  937 --      {$S MAOpen}
21  938 --      {-----------------------------------+
21  939 --      |   IScrollBar                      |
21  940 --      +-----------------------------------}
21  941 -- A    PROCEDURE TScrollBar.IScrollBar (itsSuperView: TView; itsLocation, itsSize: VPoint;
21  942 --                                       itsHSizeDet, itsVSizeDet: SizeDeterminer;
21  943 --                                       itsDirection: VHSelect; itsVal, itsMin, itsMax: LONGINT);
21  944 0- A    BEGIN
21  945 --          ICtlMgr(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet,
21  946 --            '', itsVal, itsMin, itsMax, scrollBarProc);
21  947 --          fDirection := itsDirection;
21  948 --          SetLongValues(itsVal, itsMin, itsMax, kDontRedraw);
21  949 --          IF itsDirection = h THEN
21  950 --              fDefChoice := mHScrollBarHit
21  951 --          ELSE
21  952 --              fDefChoice := mVScrollBarHit;
21  953 -0 A    END;
21  954 --      {$ENDC}
21  955 --
21  956 --
21  957 --      {$IFC qTemplateViews}
21  958 --      {$S MAOpen}
21  959 --      {-----------------------------------+
21  960 --      |   IRes                            |
21  961 --      +-----------------------------------}
21  962 -- A    PROCEDURE TScrollBar.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
21  963 --
21  964 --      VAR
21  965 --          area:           Rect;
21  966 --
21  967 0- A    BEGIN
21  968 --          INHERITED IRes(itsDocument, itsSuperView, itsParams);
21  969 --
21  970 --          ControlArea(area);
21  971 --          CreateCMgrControl(area, '', 0, 0, 0, scrollBarProc);
21  972 --          WITH ScrollBarTemplatePtr(itsParams)^ DO
21  973 --              SetLongValues(value, minimum, maximum, kDontRedraw);
21  974 --
21  975 --          WITH area DO
21  976 --              IF (bottom - top) >= (right - left) THEN
21  977 --                  fDirection := v
21  978 --              ELSE
21  979 --                  fDirection := h;
21  980 --
21  981 --          OffsetPtr(itsParams, SIZEOF(ScrollBarTemplate));
21  982 -0 A    END;
21  983 --      {$ENDC}
21  984 --
21  985 --
21  986 --      {$IFC qWriteTemplates}
21  987 --      {$S MAWriteRes}
21  988 --      {-----------------------------------+
21  989 --      |   WRes                            |
```

```
21  990 --      +-----------------------------------}
21  991 -- A    PROCEDURE TScrollBar.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21  992 --
21  993 --      VAR
21  994 --          sbPtr:      ScrollBarTemplatePtr;
21  995 --
21  996 0- A    BEGIN
21  997 --          INHERITED WRes(theResource, itsParams);
21  998 --
21  999 --          sbPtr := ScrollBarTemplatePtr(ExpandPtr(theResource, itsParams,
21 1000 --            SIZEOF(ScrollBarTemplate)));
21 1001 --
21 1002 --          WITH sbPtr^ DO
21 1003 1-              BEGIN
21 1004 --              value := fLongVal;
21 1005 --              minimum := fLongMin;
21 1006 --              maximum := fLongMax;
21 1007 -1              END;
21 1008 -0 A    END;
21 1009 --      {$ENDC}
21 1010 --
21 1011 --
21 1012 --      {$IFC qWriteTemplates}
21 1013 --      {$S MAWriteRes}
21 1014 --      {-----------------------------------+
21 1015 --      |   WriteRes                        |
21 1016 --      +-----------------------------------}
21 1017 -- A    PROCEDURE TScrollBar.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21 1018 0- A    BEGIN
21 1019 --          gWResSignature := 'sbar';   gWResType := 'TScrollBar';
21 1020 --          WRes(theResource, itsParams);
21 1021 -0 A    END;
21 1022 --      {$ENDC}
21 1023 --
21 1024 --
21 1025 --      {$S MAScroll}
21 1026 --      {-----------------------------------+
21 1027 --      |   DeltaValue                      |
21 1028 --      +-----------------------------------}
21 1029 -- A    PROCEDURE TScrollBar.DeltaValue (delta: VCoordinate);
21 1030 --
21 1031 0- A    BEGIN
21 1032 --          IF delta <> 0 THEN
21 1033 1-              BEGIN
21 1034 --              {Ensure that delta does not cause an overflow (or underflow)}
21 1035 --              IF delta > 0 THEN
21 1036 --                  delta := Min(delta, fLongMax - fLongVal)
21 1037 --              ELSE
21 1038 --                  delta := Max(delta, fLongMin - fLongVal);
21 1039 --
21 1040 --              SetLongVal(fLongVal + delta, TRUE);
21 1041 -1              END;
21 1042 -0 A    END;
21 1043 --
21 1044 --      {$S MAScroll}
21 1045 --      {-----------------------------------+
21 1046 --      |   DoMouseCommand                  |
21 1047 --      +-----------------------------------}
21 1048 -- A    FUNCTION  TScrollBar.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
21 1049 --                                           VAR hysteresis: Point): TCommand; OVERRIDE;
21 1050 --      VAR
21 1051 --          partCode:       INTEGER;
21 1052 --          oldLongValue:   VCoordinate;
21 1053 --
21 1054 0- A    BEGIN
21 1055 1-          CASE TestControl(fCMgrControl, theMouse) OF
21 1056 --              InUpButton, inDownButton, inPageUp, inPageDown:
21 1057 2-                  BEGIN
21 1058 --                  GetFocus;
21 1059 --                  partCode := TrackControl(fCMgrControl, theMouse, @SBarActionProc);
21 1060 -2                  END;
21 1061 --              inThumb:
21 1062 --                  IF TrackControl(fCMgrControl, theMouse, NIL) = inThumb THEN
21 1063 2-                      BEGIN
21 1064 --                      oldLongValue := fLongVal;
21 1065 --                      {If thumb is dragged to bottom of scroll bar then ensure that
21 1066 --                          the new long value is set to the maximum long value.}
21 1067 --                      IF GetVal = GetMax THEN
21 1068 --                          fLongVal := fLongMax
21 1069 --                      ELSE
21 1070 --                          fLongVal := BSL(GetVal, fBitsToShift);
21 1071 --                      DeltaValue(fLongVal - oldLongValue);
21 1072 -2                      END;
21 1073 -1              END;
21 1074 --          DoMouseCommand := gNoChanges;
21 1075 -0 A    END;
21 1076 --
21 1077 --      {$S MARes}
21 1078 --      {-----------------------------------+
21 1079 --      |   GetLongMax                      |
21 1080 --      +-----------------------------------}
21 1081 -- A    FUNCTION  TScrollBar.GetLongMax: VCoordinate;
21 1082 0- A    BEGIN
21 1083 --          GetLongMax := fLongMax;
21 1084 -0 A    END;
21 1085 --
21 1086 --      {$S MARes}
21 1087 --      {-----------------------------------+
21 1088 --      |   GetLongMin                      |
```

```
21 1089 --      +-----------------------------------}
21 1090 -- A    FUNCTION  TScrollBar.GetLongMin: VCoordinate;
21 1091 0- A    BEGIN
21 1092 --          GetLongMin := fLongMin;
21 1093 -0 A    END;
21 1094 --
21 1095 --      {$S MARes}
21 1096 --      {-----------------------------------+
21 1097 --      |   GetLongVal                      |
21 1098 --      +-----------------------------------}
21 1099 -- A    FUNCTION  TScrollBar.GetLongVal: VCoordinate;
21 1100 0- A    BEGIN
21 1101 --          GetLongVal := fLongVal;
21 1102 -0 A    END;
21 1103 --
21 1104 --      {$S MARes}
21 1105 --      {-----------------------------------+
21 1106 --      |   SetLongMax                      |
21 1107 --      +-----------------------------------}
21 1108 -- A    PROCEDURE TScrollBar.SetLongMax (itsMax: VCoordinate; redraw: BOOLEAN);
21 1109 --
21 1110 0- A    BEGIN
21 1111 --          IF itsMax <> fLongMax THEN
21 1112 1-              BEGIN
21 1113 --              fLongMax := itsMax;
21 1114 --              fBitsToShift := 0;
21 1115 --              WHILE itsMax > MAXINT DO
21 1116 2-                  BEGIN
21 1117 --                  itsMax := BSR(itsMax, 1);
21 1118 --                  fBitsToShift := fBitsToShift + 1;
21 1119 -2                  END;
21 1120 --              SetMax(itsMax, redraw);
21 1121 -1              END;
21 1122 -0 A    END;
21 1123 --
21 1124 --      {$S MARes}
21 1125 --      {-----------------------------------+
21 1126 --      |   SetLongMin                      |
21 1127 --      +-----------------------------------}
21 1128 -- A    PROCEDURE TScrollBar.SetLongMin (itsMin: VCoordinate; redraw: BOOLEAN);
21 1129 --
21 1130 0- A    BEGIN
21 1131 --          IF itsMin <> fLongMin THEN
21 1132 1-              BEGIN
21 1133 --              fLongMin := itsMin;
21 1134 --              SetMin(BSR(itsMin, fBitsToShift), redraw);
21 1135 -1              END;
21 1136 -0 A    END;
21 1137 --
21 1138 --      {$S MARes}
21 1139 --      {-----------------------------------+
21 1140 --      |   SetLongVal                      |
21 1141 --      +-----------------------------------}
21 1142 -- A    PROCEDURE TScrollBar.SetLongVal (itsVal: VCoordinate; redraw: BOOLEAN);
21 1143 --
21 1144 0- A    BEGIN
21 1145 --          itsVal := Max(fLongMin, Min(itsVal, fLongMax));
21 1146 --          IF itsVal <> fLongVal THEN
21 1147 1-              BEGIN
21 1148 --              fLongVal := itsVal;
21 1149 --              SetVal(BSR(itsVal, fBitsToShift), redraw);
21 1150 -1              END;
21 1151 -0 A    END;
21 1152 --
21 1153 --      {$S MARes}
21 1154 --      {-----------------------------------+
21 1155 --      |   SetLongValues                   |
21 1156 --      +-----------------------------------}
21 1157 -- A    PROCEDURE TScrollBar.SetLongValues (itsVal, itsMin, itsMax: VCoordinate; redraw: BOOLEAN);
21 1158 --
21 1159 0- A    BEGIN
21 1160 --          SetLongMax(itsMax, FALSE);
21 1161 --          SetLongMin(itsMin, redraw);
21 1162 --          SetLongVal(itsVal, FALSE);
21 1163 -0 A    END;
21 1164 --
21 1165 --      {$S MARes}
21 1166 --      {-----------------------------------+
21 1167 --      |   TrackScrollBar                  |
21 1168 --      +-----------------------------------}
21 1169 -- A    PROCEDURE TScrollBar.TrackScrollBar (partCode: INTEGER);
21 1170 --
21 1171 0- A    BEGIN
21 1172 --          { Does nothing as a default }
21 1173 -0 A    END;
21 1174 --
21 1175 --      {$IFC qDebug}
21 1176 --          {$S MAFields}
21 1177 --          {-----------------------------------+
21 1178 --          |   Fields                          |
21 1179 --          +-----------------------------------}
21 1180 -- A        PROCEDURE TScrollBar.Fields (PROCEDURE DoToField (fieldName: Str255;
21 1181 --                                                           fieldAddr: Ptr;
21 1182 --                                                           fieldType: INTEGER)); OVERRIDE;
21 1183 --
21 1184 0- A        BEGIN
21 1185 --              DoToField('TScrollBar', NIL, bClass);
21 1186 --              DoToField('fBitsToShift', @fBitsToShift, bInteger);
21 1187 --              DoToField('fDirection', @fDirection, bByte);
```

```
21 1188 --              DoToField('fLongVal', @fLongVal, bLongInt);
21 1189 --              DoToField('fLongMin', @fLongMin, bLongInt);
21 1190 --              DoToField('fLongMax', @fLongMax, bLongInt);
21 1191 --              INHERITED Fields(DoToField);
21 1192 -0 A       END;
21 1193 --      {$ENDC}
21 1194 --
21 1195 --
21 1196 --      (*****************************************************************************************)
21 1197 --      (*      T S S c r o l l B a r                                                            *)
21 1198 --      (*****************************************************************************************)
21 1199 --
21 1200 --      {$IFC qProceduralViews}
21 1201 --      {$S MAOpen}
21 1202 --      {-----------------------------------+
21 1203 --      |   ISScrollBar                     |
21 1204 --      +-----------------------------------}
21 1205 -- A    PROCEDURE TSScrollBar.ISScrollBar (itsSuperView: TView; itsLocation, itsSize: VPoint;
21 1206 --                                         itsHSizeDet, itsVSizeDet: SizeDeterminer;
21 1207 --                                         itsDirection: VHSelect; itsMax: LONGINT;
21 1208 --                                         itsScroller: TScroller);
21 1209 --
21 1210 --      VAR
21 1211 --          fi:         FailInfo;
21 1212 --
21 1213 -- B        PROCEDURE HandleFailure(error: OSErr; message: LONGINT);
21 1214 0- B        BEGIN
21 1215 --              Free;
21 1216 -0 B        END;
21 1217 --
21 1218 0- A    BEGIN
21 1219 --          fScrollers := NIL;
21 1220 --          IScrollBar(itsSuperView, itsLocation, itsSize, itsHSizeDet, itsVSizeDet,
21 1221 --                     itsDirection, 0, 0, itsMax);
21 1222 --          fCMgrControl^^.contrlVis := 0;          { Start out assuming it's inactive. }
21 1223 --
21 1224 --          CatchFailures(fi, HandleFailure);
21 1225 --          fScrollers := NewList;
21 1226 --      {$IFC qDebug}
21 1227 --          fScrollers.SetEltType('TScroller');
21 1228 --      {$ENDC}
21 1229 --          IF itsScroller <> NIL THEN
21 1230 1-              BEGIN
21 1231 --              fScrollers.InsertLast(itsScroller);
21 1232 --              itsScroller.HaveScrollBar(SELF, fDirection);
21 1233 -1              END;
21 1234 --          Success(fi);
21 1235 -0 A    END;
21 1236 --      {$ENDC}
21 1237 --
21 1238 --
21 1239 --      {$IFC qTemplateViews}
21 1240 --      {$S MAOpen}
21 1241 --      {-----------------------------------+
21 1242 --      |   IRes                            |
21 1243 --      +-----------------------------------}
21 1244 -- A    PROCEDURE TSScrollBar.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
21 1245 --
21 1246 --      VAR
21 1247 --          fi:         FailInfo;
21 1248 --
21 1249 -- B        PROCEDURE HandleFailure(error: OSErr; message: LONGINT);
21 1250 0- B        BEGIN
21 1251 --              Free;
21 1252 -0 B        END;
21 1253 --
21 1254 0- A    BEGIN
21 1255 --          INHERITED IRes(itsDocument, itsSuperView, itsParams);
21 1256 --          fCMgrControl^^.contrlVis := 0;          { Start out assuming it's inactive. }
21 1257 --
21 1258 --          CatchFailures(fi, HandleFailure);
21 1259 --          fScrollers := NewList;
21 1260 --      {$IFC qDebug}
21 1261 --          fScrollers.SetEltType('TScroller');
21 1262 --      {$ENDC}
21 1263 --          Success(fi);
21 1264 -0 A    END;
21 1265 --      {$ENDC}
21 1266 --
21 1267 --
21 1268 --      {$IFC qWriteTemplates}
21 1269 --      {$S MAWriteRes}
21 1270 --      {-----------------------------------+
21 1271 --      |   WRes                            |
21 1272 --      +-----------------------------------}
21 1273 -- A    PROCEDURE TSScrollBar.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21 1274 0- A    BEGIN
21 1275 --          INHERITED WRes(theResource, itsParams);
21 1276 -0 A    END;
21 1277 --      {$ENDC}
21 1278 --
21 1279 --
21 1280 --      {$IFC qWriteTemplates}
21 1281 --      {$S MAWriteRes}
21 1282 --      {-----------------------------------+
21 1283 --      |   WriteRes                        |
21 1284 --      +-----------------------------------}
21 1285 -- A    PROCEDURE TSScrollBar.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
21 1286 0- A    BEGIN
```

```
21 1287 --          { We don't write TSScrollBars… }
21 1288 -0 A    END;
21 1289 --      {$ENDC}
21 1290 --
21 1291 --
21 1292 --      {$S MAClose}
21 1293 --      {-----------------------------------+
21 1294 --      |   Free                            |
21 1295 --      +-----------------------------------}
21 1296 -- A    PROCEDURE TSScrollBar.Free; OVERRIDE;
21 1297 --
21 1298 --          {-----------------------------------+
21 1299 --          |   DetachFromScroller              |
21 1300 --          +-----------------------------------}
21 1301 -- B        PROCEDURE DetachFromScroller (theScroller: TScroller);
21 1302 --
21 1303 0- B        BEGIN
21 1304 --              IF theScroller.fScrollBars[fDirection] = SELF THEN
21 1305 --                  theScroller.HaveScrollBar(NIL, fDirection);
21 1306 -0 B        END;
21 1307 --
21 1308 0- A    BEGIN
21 1309 --          IF fScrollers <> NIL THEN
21 1310 --              fScrollers.Each(DetachFromScroller);
21 1311 --          FreeObject(fScrollers);
21 1312 --          INHERITED Free;
21 1313 -0 A    END;
21 1314 --
21 1315 --      {$S MAActivate}
21 1316 --      {-----------------------------------+
21 1317 --      |   Activate                        |
21 1318 --      +-----------------------------------}
21 1319 -- A    PROCEDURE TSScrollBar.Activate (entering: BOOLEAN); OVERRIDE;
21 1320 --
21 1321 -- B        PROCEDURE SetScrollBar;
21 1322 0- B        BEGIN
21 1323 --              IF entering THEN
21 1324 --                  ShowControl(fCMgrControl)
21 1325 --              ELSE
21 1326 1-                  BEGIN
21 1327 --                  HideControl(fCMgrControl);
21 1328 --                  Draw(fCMgrControl^^.contrlRect);    { Get the frame drawn.                      }
21 1329 -1                  END;
21 1330 --              ValidRect(fCMgrControl^^.contrlRect);   { Because Control Manager invalidates it.   }
21 1331 -0 B        END;
21 1332 --
21 1333 0- A    BEGIN
21 1334 --          WhileFocused(SetScrollBar, kRedraw);
21 1335 -0 A    END;
21 1336 --
21 1337 --      {$S MAScroll}
21 1338 --      {-----------------------------------+
21 1339 --      |   DoMouseCommand                  |
21 1340 --      +-----------------------------------}
21 1341 -- A    FUNCTION TSScrollBar.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
21 1342 --                                           VAR hysteresis: Point): TCommand; OVERRIDE;
21 1343 --      VAR
21 1344 --          partCode:       INTEGER;
21 1345 --          sBarDelta:      LONGINT;
21 1346 --
21 1347 --          {-----------------------------------+
21 1348 --          |   ScrollToThumb                   |
21 1349 --          +-----------------------------------}
21 1350 -- B        PROCEDURE ScrollToThumb (theView: TScroller);
21 1351 --
21 1352 0- B        BEGIN
21 1353 --              sBarDelta := sBarDelta + theView.ScrollRelative(fDirection, fLongVal);
21 1354 -0 B        END;
21 1355 --
21 1356 0- A    BEGIN
21 1357 --          IF TestControl(fCMgrControl, theMouse) = inThumb THEN
21 1358 1-              BEGIN
21 1359 --              IF TrackControl(fCMgrControl, theMouse, NIL) = inThumb THEN
21 1360 2-                  BEGIN
21 1361 --                  GetFocus;
21 1362 --                  sBarDelta := 0;
21 1363 --                  {If thumb is dragged to bottom of scroll bar then ensure that
21 1364 --                      the new long value is set to the maximum long value.}
21 1365 --                  IF GetVal = GetMax THEN
21 1366 --                      fLongVal := fLongMax
21 1367 --                  ELSE
21 1368 --                      fLongVal := BSL(GetVal, fBitsToShift);
21 1369 --                  fScrollers.Each(ScrollToThumb);
21 1370 --                  SetFocus;
21 1371 --                  IF sBarDelta <> 0 THEN
21 1372 --                      SetLongVal(fLongVal + sBarDelta, TRUE);
21 1373 -2                  END;
21 1374 --              DoMouseCommand := gNoChanges;
21 1375 -1              END
21 1376 --          ELSE
21 1377 --              DoMouseCommand := INHERITED DoMouseCommand(theMouse, info, hysteresis);
21 1378 -0 A    END;
21 1379 --
21 1380 --      {$S MARes}
21 1381 --      {-----------------------------------+
21 1382 --      |   Draw                            |
21 1383 --      +-----------------------------------}
21 1384 -- A    PROCEDURE TSScrollBar.Draw (area: Rect); OVERRIDE;
21 1385 --
```

```
21 1386 --      VAR
21 1387 --          itsRect:        Rect;
21 1388 --
21 1389 0- A    BEGIN
21 1390 --          IF fCMgrControl^^.contrlVis = 0 THEN
21 1391 1-              BEGIN
21 1392 --              PenNormal;                          { Make sure we draw solid, 1-pixel wide lines   }
21 1393 --              itsRect := fCMgrControl^^.contrlRect;
21 1394 --              FrameRect(itsRect);
21 1395 -1              END;
21 1396 --
21 1397 --          INHERITED Draw(area);
21 1398 -0 A    END;
21 1399 --
21 1400 --      {$S MAScroll}
21 1401 --      {-----------------------------------+
21 1402 --      |   TrackScrollBar                  |
21 1403 --      +-----------------------------------}
21 1404 -- A    PROCEDURE TSScrollBar.TrackScrollBar (partCode: INTEGER);
21 1405 --
21 1406 --      VAR
21 1407 --          sBarDelta:      LONGINT;
21 1408 --
21 1409 --          {-----------------------------------+
21 1410 --          |   ScrollView                      |
21 1411 --          +-----------------------------------}
21 1412 -- B        PROCEDURE ScrollView (theView: TScroller);
21 1413 --
21 1414 0- B        BEGIN
21 1415 --              sBarDelta := sBarDelta + theView.ScrollStep(fDirection, partCode);
21 1416 -0 B        END;
21 1417 --
21 1418 0- A    BEGIN
21 1419 --          sBarDelta := 0;
21 1420 --          fScrollers.Each(ScrollView);
21 1421 --          SetFocus;
21 1422 --          DeltaValue(sBarDelta);
21 1423 -0 A    END;
21 1424 --
21 1425 --      {$IFC qDebug}
21 1426 --          {$S MAFields}
21 1427 --          {-----------------------------------+
21 1428 --          |   Fields                          |
21 1429 --          +-----------------------------------}
21 1430 -- A        PROCEDURE TSScrollBar.Fields (PROCEDURE DoToField (fieldName: Str255;
21 1431 --                                                            fieldAddr: Ptr;
21 1432 --                                                            fieldType: INTEGER)); OVERRIDE;
21 1433 --
21 1434 0- A        BEGIN
21 1435 --              DoToField('TSScrollBar', NIL, bClass);
21 1436 --              DoToField('fScrollers', @fScrollers, bObject);
21 1437 --              INHERITED Fields(DoToField);
21 1438 -0 A        END;
21 1439 --      {$ENDC}
13 3009 --      {$I UMacApp.TCommand.p}
```

```
22    1 --      {$P}
22    2 --      { UMacApp.TCommand.p }
22    3 --      { Copyright © 1985-1988 by Apple Computer, Inc.  All rights reserved. }
22    4 --
22    5 --      (*****************************************************************************************)
22    6 --      (*      T C o m m a n d                                                                  *)
22    7 --      (*****************************************************************************************)
22    8 --      {$S MASelCommand}
22    9 --      {-----------------------------------+
22   10 --      |   ICommand                        |
22   11 --      +-----------------------------------}
22   12 -- A    PROCEDURE TCommand.ICommand(itsCmdNumber: CmdNumber; itsDocument: TDocument;
22   13 --                                  itsView: TView; itsScroller: TScroller);
22   14 0- A    BEGIN
22   15 --          fCmdNumber := itsCmdNumber;
22   16 --          fChangedDocument := itsDocument;
22   17 --          fTarget := NIL;
22   18 --          fCmdDone := FALSE;
22   19 --          fCanUndo := TRUE;
22   20 --          fCausesChange := TRUE;
22   21 --          fChangesClipboard := FALSE;
22   22 --          fConstrainsMouse := FALSE;
22   23 --          fViewConstrain := TRUE;
22   24 --          fTrackNonMovement := FALSE;
22   25 --          fView := itsView;
22   26 --          fScroller := itsScroller;
22   27 -0 A    END;
22   28 --
22   29 --
22   30 --      {$S MADoCommand}
22   31 --      {-----------------------------------+
22   32 --      |   AutoScroll                      |
22   33 --      +-----------------------------------}
22   34 -- A    PROCEDURE TCommand.AutoScroll (deltaH, deltaV: VCoordinate);
22   35 0- A    BEGIN
22   36 --          fScroller.ScrollBy(deltaH, deltaV, kRedraw);
22   37 -0 A    END;
22   38 --
22   39 --
22   40 --      {$S MADoCommand}
22   41 --      {-----------------------------------+
22   42 --      |   Commit                          |
22   43 --      +-----------------------------------}
22   44 -- A    PROCEDURE TCommand.Commit;
22   45 0- A    BEGIN
22   46 -0 A    END;
22   47 --
22   48 --
22   49 --      {$S MADoCommand}
22   50 --      {-----------------------------------+
22   51 --      |   DoIt                            |
22   52 --      +-----------------------------------}
22   53 -- A    PROCEDURE TCommand.DoIt;
22   54 0- A    BEGIN
22   55 -0 A    END;
22   56 --
22   57 --
22   58 --      {$S MADoCommand}
22   59 --      {-----------------------------------+
22   60 --      |   RedoIt                          |
22   61 --      +-----------------------------------}
22   62 -- A    PROCEDURE TCommand.RedoIt;
22   63 0- A    BEGIN
22   64 -0 A    END;
22   65 --
22   66 --
22   67 --      {$S MADoCommand}
22   68 --      {-----------------------------------+
22   69 --      |   TrackConstrain                  |
22   70 --      +-----------------------------------}
22   71 -- A    PROCEDURE TCommand.TrackConstrain(anchorPoint, previousPoint: VPoint;
22   72 --                                        VAR nextPoint: VPoint);
22   73 0- A    BEGIN
22   74 -0 A    END;
22   75 --
22   76 --
22   77 --      {$S MADoCommand}
22   78 --      {-----------------------------------+
22   79 --      |   TrackFeedback                   |
22   80 --      +-----------------------------------}
22   81 -- A    PROCEDURE TCommand.TrackFeedback(anchorPoint, nextPoint: VPoint;
22   82 --                                       turnItOn, mouseDidMove: BOOLEAN);
22   83 --          VAR r:  Rect;
22   84 0- A    BEGIN
22   85 --          IF mouseDidMove THEN
22   86 1-              BEGIN
22   87 --              Pt2Rect(fView.ViewToQDPt(anchorPoint), fView.ViewToQDPt(nextPoint), r);
22   88 --              FrameRect(r);
22   89 -1              END;
22   90 -0 A    END;
22   91 --
22   92 --
22   93 --      {$S MADoCommand}
22   94 --      {-----------------------------------+
22   95 --      |   TrackMouse                      |
22   96 --      +-----------------------------------}
22   97 -- A    FUNCTION  TCommand.TrackMouse(aTrackPhase: TrackPhase;
22   98 --                          VAR anchorPoint, previousPoint, nextPoint: VPoint;
```

```
22   99 --                          mouseDidMove: BOOLEAN): TCommand;
22  100 0- A   BEGIN
22  101 --          TrackMouse := SELF;
22  102 -0 A   END;
22  103 --
22  104 --
22  105 --      {$S MADoCommand}
22  106 --      {-----------------------------------+
22  107 --      |   UndoIt                          |
22  108 --      +-----------------------------------}
22  109 -- A   PROCEDURE TCommand.UndoIt;
22  110 0- A   BEGIN
22  111 -0 A   END;
22  112 --
22  113 --
22  114 --      {$IFC qDebug}
22  115 --          {$S MAFields}
22  116 --          {-----------------------------------+
22  117 --          |   Fields                          |
22  118 --          +-----------------------------------}
22  119 -- A       PROCEDURE TCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
22  120 --                                                         fieldAddr: Ptr;
22  121 --                                                         fieldType: INTEGER)); OVERRIDE;
22  122 --
22  123 0- A       BEGIN
22  124 --              DoToField('TCommand', NIL, bClass);
22  125 --              DoToField('fCmdNumber', @fCmdNumber, bCmdNumber);
22  126 --              DoToField('fChangedDocument', @fChangedDocument, bObject);
22  127 --              DoToField('fTarget', @fTarget, bObject);
22  128 --              DoToField('fCmdDone', @fCmdDone, bBoolean);
22  129 --              DoToField('fCanUndo', @fCanUndo, bBoolean);
22  130 --              DoToField('fCausesChange', @fCausesChange, bBoolean);
22  131 --              DoToField('fChangesClipboard', @fChangesClipboard, bBoolean);
22  132 --              DoToField('fView', @fView, bObject);
22  133 --              DoToField('fScroller', @fScroller, bObject);
22  134 --              DoToField('fConstrainsMouse', @fConstrainsMouse, bBoolean);
22  135 --              DoToField('fViewConstrain', @fViewConstrain, bBoolean);
22  136 --              INHERITED Fields(DoToField);
22  137 -0 A       END ;
22  138 --      {$ENDC}
13 3010 --      {$I UMacApp.TDeskScrapView.p}
```

```
23    1 --      {$P}
23    2 --      { UMacApp.TDeskScrapView.p }
23    3 --      { Copyright © 1985-1988 Apple Computer, Inc.  All rights reserved. }
23    4 --
23    5 --
23    6 --      (*****************************************************************************************)
23    7 --      (*      T D e s k S c r a p V i e w                                                      *)
23    8 --      (*****************************************************************************************)
23    9 --
23   10 --      {$IFC qProceduralViews}
23   11 --      {$S MAInit}
23   12 --      {-----------------------------------+
23   13 --      |   IDeskScrapView                  |
23   14 --      +-----------------------------------}
23   15 -- A    PROCEDURE TDeskScrapView.IDeskScrapView;
23   16 --      VAR
23   17 --          itsLocation:    VPoint;
23   18 --          itsSize:        VPoint;
23   19 --          aPtr:           PScrapStuff;
23   20 0- A    BEGIN
23   21 --          SetVPt(itsLocation, 0, 0);
23   22 --          SetVPt(itsSize, 200, 200);
23   23 --          { Can't know the extent required until we actually read the data in, which we won't bother
23   24 --              to do unless forced to by receiving an Update event }
23   25 --
23   26 --          IView(NIL, NIL, itsLocation, itsSize, sizeVariable, sizeVariable);
23   27 --
23   28 --          aPtr := InfoScrap;
23   29 --          fScrapCount := aPtr^.scrapCount - 1;
23   30 --          fDataHandle := NIL;
23   31 --          fHavePicture := FALSE;
23   32 --          fHaveText := FALSE;
23   33 -0 A    END;
23   34 --      {$ENDC}
23   35 --
23   36 --
23   37 --      {$IFC qTemplateViews}
23   38 --      {$S MAInit}
23   39 --      {-----------------------------------+
23   40 --      |   IRes                            |
23   41 --      +-----------------------------------}
23   42 -- A    PROCEDURE TDeskScrapView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
23   43 --
23   44 --      VAR
23   45 --          aPtr:           PScrapStuff;
23   46 --
23   47 0- A    BEGIN
23   48 --          INHERITED IRes(itsDocument, itsSuperView, itsParams);
23   49 --
23   50 --          aPtr := InfoScrap;
23   51 --          fScrapCount := aPtr^.scrapCount - 1;
23   52 --          fDataHandle := NIL;
23   53 --          fHavePicture := FALSE;
23   54 --          fHaveText := FALSE;
23   55 -0 A    END;
23   56 --      {$ENDC}
23   57 --
23   58 --
23   59 --      {$S MAClipboard}
23   60 --      {-----------------------------------+
23   61 --      |   Free                            |
23   62 --      +-----------------------------------}
23   63 -- A    PROCEDURE TDeskScrapView.Free; OVERRIDE;
23   64 --
23   65 0- A    BEGIN
23   66 --          { Never want to free gClipOrphanage.  What if some creates another view of this type? }
23   67 -0 A    END;
23   68 --
23   69 --
23   70 --      {$S MAClipboard}
23   71 --      {-----------------------------------+
23   72 --      |   CalcMinSize                     |
23   73 --      +-----------------------------------}
23   74 -- A    PROCEDURE TDeskScrapView.CalcMinSize (VAR minSize: VPoint); OVERRIDE;
23   75 --
23   76 --      VAR
23   77 --          vhs:        VHSelect;
23   78 --          aRect:      Rect;
23   79 --          aTEHandle:  TEHandle;
23   80 --          oldHText:   Handle;
23   81 --
23   82 0- A    BEGIN
23   83 --          minSize := fSize;
23   84 --          IF fDataHandle <> NIL THEN
23   85 --              IF fHaveText THEN
23   86 1-                  BEGIN { TEXT }
23   87 --                  IF Focus THEN
23   88 2-                      BEGIN
23   89 --                      SetPortTextStyle(gSystemStyle);
23   90 --                      fSuperView.GetQDExtent(aRect);
23   91 --                      aTEHandle := TENew(aRect, aRect);
23   92 --                      FailNIL(aTEHandle);
23   93 --
23   94 --                      WITH aTEHandle^^ DO
23   95 3-                          BEGIN
23   96 --                          oldHText := hText;  { We'll need this for TEDispose }
23   97 --                          hText := fDataHandle;
23   98 -3                          END;
```

```
23   99 --                    TECalText(aTEHandle);
23  100 --                    WITH aTEHandle^^ DO
23  101 3-                        BEGIN
23  102 --                        minSize.h := aRect.right - aRect.left;
23  103 --                        minSize.v := Min(IntMultiply(nLines, lineHeight), kMaxCoord);
23  104 --                        hText := oldHText;  { So TEDispose doesn't dispose our handle }
23  105 -3                        END;
23  106 --
23  107 --                    TEDispose(aTEHandle);
23  108 -2                    END;
23  109 -1                END
23  110 --
23  111 --            ELSE
23  112 1-                BEGIN
23  113 --        {$IFC qDebug}
23  114 --                IF gIntenseDebugging THEN
23  115 2-                    BEGIN
23  116 --                    Write('Picture Frame:' );
23  117 --                    WriteRect(PicHandle(fDataHandle)^^.picFrame);
23  118 -2                    END;
23  119 --        {$ENDC}
23  120 --
23  121 --                FOR vhs := v TO h DO
23  122 --                    IF NOT gVarClipPicSize THEN
23  123 --                        WITH PicHandle(fDataHandle)^^.picFrame DO
23  124 --                            minSize.vh[vhs] := botRight.vh[vhs] - topLeft.vh[vhs];
23  125 -1                END { PICT };
23  126 -0 A   END;
23  127 --
23  128 --
23  129 --    {$IFC qDebug}
23  130 --    {$S MAFields}
23  131 --    {-----------------------------------+
23  132 --    |   Fields                          |
23  133 --    +-----------------------------------}
23  134 -- A   PROCEDURE TDeskScrapView.Fields (PROCEDURE DoToField (fieldName: Str255;
23  135 --                                                           fieldAddr: Ptr;
23  136 --                                                           fieldType: INTEGER));
23  137 --
23  138 0- A   BEGIN
23  139 --        DoToField('TDeskScrapView', NIL, bClass);
23  140 --        DoToField('fHavePicture', @fHavePicture, bBoolean);
23  141 --        DoToField('fHaveText', @fHaveText, bBoolean);
23  142 --        DoToField('fScrapCount', @fScrapCount, bInteger);
23  143 --        DoToField('fDataHandle', @fDataHandle, bHandle);
23  144 --        INHERITED Fields(DoToField);
23  145 -0 A   END;
23  146 --    {$ENDC}
23  147 --
23  148 --
23  149 --    {$S MAClipboard}
23  150 --    {-----------------------------------+
23  151 --    |   CheckScrapContents              |
23  152 --    +-----------------------------------}
23  153 -- A   PROCEDURE TDeskScrapView.CheckScrapContents;
23  154 --    VAR
23  155 --        length:         LONGINT;
23  156 --        offset:         LONGINT;
23  157 --        aHandle:        Handle;
23  158 --        resTypeFound:   ResType;
23  159 --        savedPerm:      BOOLEAN;
23  160 --        fi:             FailInfo;
23  161 --
23  162 --        {------------------------------+
23  163 --        |   HdlFailure                  |
23  164 --        +------------------------------}
23  165 -- B      PROCEDURE HdlFailure(error: OSErr; message: LONGINT);
23  166 0- B      BEGIN
23  167 --            IF aHandle <> NIL THEN
23  168 --                DisposHandle(aHandle);
23  169 --            fDataHandle := NIL;
23  170 -0 B      END;
23  171 --
23  172 --        {------------------------------+
23  173 --        |   LookForScrapType            |
23  174 --        +------------------------------}
23  175 -- B      FUNCTION  LookForScrapType(theResType: ResType): BOOLEAN;
23  176 --        VAR
23  177 --            err:        LONGINT;
23  178 --            offset:     LongInt;
23  179 0- B      BEGIN
23  180 --            err := GetScrap(NIL, theResType, offset);
23  181 --            IF err > 0 THEN
23  182 --                resTypeFound := theResType;
23  183 --    {$IFC qDebug}
23  184 --            IF (err <> noTypeErr) AND (err < 0) THEN
23  185 --                WriteLn('LookForScrapType: err = ', err:1);
23  186 --    {$ENDC}
23  187 --            LookForScrapType := err > 0;
23  188 -0 B      END;
23  189 --
23  190 0- A   BEGIN
23  191 --        IF fScrapCount <> gNewScrapStuff.scrapCount THEN
23  192 1-            BEGIN
23  193 --
23  194 --            { Make sure we get rid of the old scrap handle if necessary }
23  195 --            IF fDataHandle <> NIL THEN
23  196 2-                BEGIN
23  197 --                DisposHandle(fDataHandle);
```

```
23  198 --                  fDataHandle := NIL;
23  199 -2                 END;
23  200 --
23  201 --              fHaveText := LookForScrapType('TEXT');
23  202 --              fHavePicture := LookForScrapType('PICT');
23  203 --
23  204 --              IF fHavePicture OR fHaveText THEN
23  205 2-                  BEGIN
23  206 --                  aHandle := NewPermHandle(0);
23  207 --                  FailNIL(aHandle);
23  208 --
23  209 --                  CatchFailures(fi, HdlFailure);
23  210 --                  savedPerm := PermAllocation(TRUE);
23  211 --                  length := GetScrap(aHandle, resTypeFound, offset);
23  212 --                  savedPerm := PermAllocation(savedPerm);
23  213 --
23  214 --                  IF length < 0 THEN          { Only results < 0 are really errors }
23  215 --                      FailOSErr(length);
23  216 --                  fDataHandle := aHandle;
23  217 --                  AdjustSize;
23  218 --                  If Focus THEN ;         { Make sure we're still focused }
23  219 --                  Success(fi);
23  220 -2                 END;
23  221 -1              END;
23  222 -0 A    END;
23  223 --
23  224 --
23  225 --      {$S MAClipboard}
23  226 --      {-----------------------------------+
23  227 --      |   Draw                            |
23  228 --      +-----------------------------------}
23  229 -- A    PROCEDURE TDeskScrapView.Draw(area: Rect);
23  230 --
23  231 --      VAR
23  232 --          aRect:      Rect;
23  233 --          vhs:        VHSelect;
23  234 --
23  235 0- A    BEGIN
23  236 --
23  237 --          { Has scrap been updated since we last drew? }
23  238 --          CheckScrapContents;
23  239 --
23  240 --          IF fHaveText THEN
23  241 1-              BEGIN
23  242 --              SetPortTextStyle(gSystemStyle);
23  243 --              GetQDExtent(aRect);
23  244 --              { Does TextEdit handle failure in TextBox??? }
23  245 --              TextBox(fDataHandle^, GetHandleSize(fDataHandle), aRect, teJustLeft);
23  246 -1              END
23  247 --
23  248 --          ELSE
23  249 --
23  250 --          IF fHavePicture THEN
23  251 1-              BEGIN
23  252 --              IF gVarClipPicSize THEN
23  253 --                  GetQDExtent(aRect)
23  254 --              ELSE
23  255 2-                  BEGIN
23  256 --                  aRect := PicHandle(fDataHandle)^^.picFrame;
23  257 --                  FOR vhs := v TO h DO
23  258 3-                      BEGIN
23  259 --                      aRect.botRight.vh[vhs] := aRect.botRight.vh[vhs] - aRect.topLeft.vh[vhs];
23  260 --                      aRect.topLeft.vh[vhs] := 0;
23  261 -3                     END;
23  262 -2                 END;
23  263 --              DrawPicture(PicHandle(fDataHandle), aRect);
23  264 -1              END;
23  265 --
23  266 --          IF fScrapCount <> gNewScrapStuff.scrapCount THEN
23  267 1-              BEGIN
23  268 --              ValidRect(aRect);  { avoid flicker upon clipboard installation -- but could this
23  269 --                                   cause failure to update frame border/appendages sometimes, if that
23  270 --                                   code ever checked for visibility before drawing ??? }
23  271 --              fScrapCount := gNewScrapStuff.scrapCount;
23  272 --              gClipWrittenToDeskScrap := TRUE;    { Don't need to write it back out. }
23  273 -1              END;
23  274 -0 A    END;
23  275 --
23  276 --
23  277 --      {$IFC qInspector}
23  278 --      {$S MAInspector}
23  279 --      {-----------------------------------+
23  280 --      |   GetInspectorName                |
23  281 --      +-----------------------------------}
23  282 -- A    PROCEDURE TDeskScrapView.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
23  283 --
23  284 0- A    BEGIN
23  285 --          IF SELF = gClipView THEN
23  286 --              inspectorName := 'gClipView'
23  287 --          ELSE IF SELF = gClipOrphanage THEN
23  288 --              inspectorName := 'gClipOrphanage';
23  289 -0 A    END;
23  290 --      {$ENDC}
23  291 --
23  292 --
23  293 --      {$S MAClipboard}
23  294 --      {-----------------------------------+
23  295 --      |   SuperViewChangedSize            |
23  296 --      +-----------------------------------}
```

```
23  297 -- A   PROCEDURE TDeskScrapView.SuperViewChangedSize (delta: VPoint; invalidate: BOOLEAN); OVERRIDE;
23  298 --
23  299 0- A   BEGIN
23  300 --          AdjustSize;
23  301 --          ForceRedraw;
23  302 -0 A   END;
23  303 --
23  304 --
23  305 --      {$S MAClipboard}
23  306 --      {-----------------------------------+
23  307 --      |   WriteToDeskScrap                 |
23  308 --      +-----------------------------------}
23  309 -- A   PROCEDURE TDeskScrapView.WriteToDeskScrap; OVERRIDE;
23  310 --
23  311 0- A   BEGIN
23  312 --          { This view represents data that's already written to the desk scrap }
23  313 -0 A   END;
23  314 --
23  315 --
13 3011 --      {$I UMacApp.TPrintHandler.p}
```

```
24    1 --      {$P}
24    2 --      { UMacApp.TPrintHandler.p }
24    3 --      { Copyright © 1985-1988 by Apple Computer, Inc.  All rights reserved. }
24    4 --
24    5 --      (*****************************************************************************************)
24    6 --      (*      T P r i n t H a n d l e r                                                        *)
24    7 --      (*****************************************************************************************)
24    8 --
24    9 --      {$S MAMain}
24   10 --      {-----------------------------------+
24   11 --      |   IPrintHandler                   |
24   12 --      +-----------------------------------}
24   13 -- A    PROCEDURE TPrintHandler.IPrintHandler(itsView: TView);
24   14 0- A    BEGIN
24   15 --          fView := itsView;
24   16 --          IEvtHandler(NIL);
24   17 -0 A    END;
24   18 --
24   19 --
24   20 --      {$S MANever}
24   21 --      {-----------------------------------+
24   22 --      |   BreakFollowing                  |
24   23 --      +-----------------------------------}
24   24 -- A    FUNCTION  TPrintHandler.BreakFollowing(vhs: VHSelect; prevBreak: VCoordinate;
24   25 --                VAR automatic: BOOLEAN): VCoordinate;
24   26 0- A    BEGIN
24   27 --      {$IFC qDebug}
24   28 --          ProgramBreak('BreakFollowing called for non-subclassed Printhandler');
24   29 --      {$ENDC}
24   30 -0 A    END;
24   31 --
24   32 --
24   33 --      {$IFC qDebug}
24   34 --      {$S MAFields}
24   35 --      {-----------------------------------+
24   36 --      |   Fields                          |
24   37 --      +-----------------------------------}
24   38 -- A    PROCEDURE TPrintHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
24   39 --                                                           fieldAddr: Ptr;
24   40 --                                                           fieldType: INTEGER));
24   41 0- A    BEGIN
24   42 --          DoToField('TPrintHandler', NIL, bClass);
24   43 --          DoToField('fView', @fView, bObject);
24   44 --          DoToField('fDeviceRes', @fDeviceRes, bPoint);
24   45 --          DoToField('fEffectiveDeviceRes', @fEffectiveDeviceRes, bPoint);
24   46 --          DoToField('fViewPerPage', @fViewPerPage, bVPoint);
24   47 --          INHERITED Fields(DoToField);
24   48 -0 A    END;
24   49 --      {$ENDC}
24   50 --
24   51 --
24   52 --      {$S MANever}
24   53 --      {-----------------------------------+
24   54 --      |   CalcPageStrips                  |
24   55 --      +-----------------------------------}
24   56 -- A    PROCEDURE TPrintHandler.CalcPageStrips (VAR pageStrips: Point);
24   57 0- A    BEGIN
24   58 --      {$IFC qDebug}
24   59 --          ProgramBreak('CalcPageStrips called for non-subclassed Printhandler');
24   60 --      {$ENDC}
24   61 -0 A    END;
24   62 --
24   63 --
24   64 --      {$S MANever}
24   65 --      {-----------------------------------+
24   66 --      |   CalcViewPerPage                 |
24   67 --      +-----------------------------------}
24   68 -- A    PROCEDURE TPrintHandler.CalcViewPerPage(VAR amtPerPage: VPoint);
24   69 0- A    BEGIN
24   70 --      {$IFC qDebug}
24   71 --          ProgramBreak('CalcViewPerPage called for non-subclassed Printhandler');
24   72 --      {$ENDC}
24   73 -0 A    END;
24   74 --
24   75 --
24   76 --      {$S MANonRes}
24   77 --      {-----------------------------------+
24   78 --      |   CheckPrinter                    |
24   79 --      +-----------------------------------}
24   80 -- A    PROCEDURE TPrintHandler.CheckPrinter; {See if there are changed printer-configuration parameters which
24   81 --                                             need to be absorbed, and if so, absorb them}
24   82 0- A    BEGIN
24   83 -0 A    END;
24   84 --
24   85 --
24   86 --      {$S BBRes}
24   87 --      {-----------------------------------+
24   88 --      |   DrawPrintFeedback               |
24   89 --      +-----------------------------------}
24   90 -- A    PROCEDURE TPrintHandler.DrawPrintFeedback(area: Rect);
24   91 --          {Draw page-breaks, page-numbers, etc.}
24   92 0- A    BEGIN
24   93 -0 A    END;
24   94 --
24   95 --
24   96 --      {$S BBRes}
24   97 --      {-----------------------------------+
24   98 --      |   DrawPageBreak                   |
```

```
24   99 --      +-----------------------------------}
24  100 -- A   PROCEDURE TPrintHandler.DrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
24  101 --                                            loc: VCoordinate; automatic: BOOLEAN);
24  102 0- A   BEGIN
24  103 -0 A   END;
24  104 --
24  105 --
24  106 --      {$S BBRes}
24  107 --      {-----------------------------------+
24  108 --      |   FocusOnInterior                 |
24  109 --      +-----------------------------------}
24  110 -- A   PROCEDURE TPrintHandler.FocusOnInterior;
24  111 0- A   BEGIN
24  112 -0 A   END;
24  113 --
24  114 --
24  115 --      {$IFC qInspector}
24  116 --      {$S MAInspector}
24  117 --      {-----------------------------------+
24  118 --      |   GetInspectorName                |
24  119 --      +-----------------------------------}
24  120 -- A   PROCEDURE TPrintHandler.GetInspectorName (VAR inspectorName: Str255); OVERRIDE;
24  121 --
24  122 --      VAR
24  123 --          hexString:      String8;
24  124 --
24  125 0- A   BEGIN
24  126 --          IF SELF = gNullPrintHandler THEN
24  127 --              inspectorName := 'gNullPrintHandler'
24  128 --          ELSE IF IsObject(fView) THEN
24  129 1-              BEGIN
24  130 --              LIntToHex(ORD4(fView), hexString, 6);
24  131 --              InspectorName := CONCAT('for view ', hexString);
24  132 -1              END;
24  133 -0 A   END;
24  134 --      {$ENDC}
24  135 --
24  136 --
24  137 --      {$S MANever}
24  138 --      {-----------------------------------+
24  139 --      |   LocatePageInterior              |
24  140 --      +-----------------------------------}
24  141 -- A   PROCEDURE TPrintHandler.LocatePageInterior(pageNumber: INTEGER; VAR loc: Point);
24  142 0- A   BEGIN
24  143 -0 A   END;
24  144 --
24  145 --
24  146 --      {$S MANever}
24  147 --      {-----------------------------------+
24  148 --      |   MaxPageNumber                   |
24  149 --      +-----------------------------------}
24  150 -- A   FUNCTION  TPrintHandler.MaxPageNumber: INTEGER;
24  151 0- A   BEGIN
24  152 --          MaxPageNumber := 0;
24  153 -0 A   END;
24  154 --
24  155 --
24  156 --      {$S MAPrint}
24  157 --      {-----------------------------------+
24  158 --      |   Print                           |
24  159 --      +-----------------------------------}
24  160 -- A   FUNCTION  TPrintHandler.Print(itsCmdNumber: CmdNumber; VAR proceed: BOOLEAN): TCommand;
24  161 0- A   BEGIN
24  162 --          proceed := TRUE;
24  163 --          Print := gNoChanges;
24  164 -0 A   END;
24  165 --
24  166 --
24  167 --      {$S MANonRes}
24  168 --      {-----------------------------------+
24  169 --      |   PrinterChanged                  |
24  170 --      +-----------------------------------}
24  171 -- A   PROCEDURE TPrintHandler.PrinterChanged;
24  172 0- A   BEGIN
24  173 -0 A   END;
24  174 --
24  175 --
24  176 --      {$S MAFinder}{??? Signal error ???}
24  177 --      {-----------------------------------+
24  178 --      |   SetupForFinder                  |
24  179 --      +-----------------------------------}
24  180 -- A   FUNCTION  TPrintHandler.SetupForFinder: BOOLEAN;
24  181 0- A   BEGIN
24  182 --      {$IFC qDebug}
24  183 --          WriteLn('SetupForFinder called for a view that can''t');
24  184 --      {$ENDC}
24  185 --          SetupForFinder := FALSE;
24  186 -0 A   END;
24  187 --
24  188 --
24  189 --      {$S MANever}
24  190 --      {-----------------------------------+
24  191 --      |   RedoPageBreaks                  |
24  192 --      +-----------------------------------}
24  193 -- A   PROCEDURE TPrintHandler.RedoPageBreaks;
24  194 0- A   BEGIN
24  195 -0 A   END;
24  196 --
24  197 --
```

```
24  198 --      {$S BBRes}
24  199 --      {-----------------------------------+
24  200 --      |   Reset                           |
24  201 --      +-----------------------------------}
24  202 -- A    PROCEDURE TPrintHandler.Reset;
24  203 0- A    BEGIN
24  204 -0 A    END;
24  205 --
24  206 --
24  207 --      {$S MANever}
24  208 --      {-----------------------------------+
24  209 --      |   SetPageInterior                 |
24  210 --      +-----------------------------------}
24  211 -- A    PROCEDURE TPrintHandler.SetPageInterior(pageNumber: INTEGER);
24  212 0- A    BEGIN
24  213 -0 A    END;
24  214 --
24  215 --
24  216 --      {$S MANever}
24  217 --      {-----------------------------------+
24  218 --      |   SetPageOffset                   |
24  219 --      +-----------------------------------}
24  220 -- A    PROCEDURE TPrintHandler.SetPageOffset(coord: VPoint);
24  221 0- A    BEGIN
24  222 -0 A    END;
13 3012 --
13 3013 --      END.
```

```
25    1 --    {$P}
25    2 --    { UMAUtil.p }
25    3 --    { Copyright © 1984-1988  Apple Computer, Inc.  All rights reserved. }
25    4 --
25    5 --    { Conditional compile flags used by this unit:
25    6 --
25    7 --        qDebug          True includes debugging code, false ignores debugging code.
25    8 --                        True also sets $N+.
25    9 --
25   10 --        qTrace          True sets $D++, causing the compiler to generate routine
25   11 --                        names for the debugger, and to generate calls to %_BP
25   12 --                        and %_EP at the beginning and end of each routine.
25   13 --                        Furthermore, if qTrace it true, some routines are enclosed
25   14 --                        with $D+ and $D++ to prevent generation of %_BP and %_EP.
25   15 --
25   16 --        qNames          Effective only if qTrace is false:  If qNames is true then
25   17 --                        $D+ is set, else $D- is set.
25   18 --
25   19 --        qRangeCheck     True sets $R+, false sets $R-.
25   20 --
25   21 --        qNeedsROM128K   True assumes the code will always be run on 128K ROMs or
25   22 --                        later and ignores code supporting 64K ROMs.  False
25   23 --                        includes code to support 64K ROMs.
25   24 --
25   25 --     }
25   26 --
25   27 --
25   28 --    UNIT UMAUtil;
25   29 --
25   30 --    {
25   31 --        This unit implements a set of simple type declarations and utility
25   32 --        routines.
25   33 --    }
25   34 --
25   35 --    INTERFACE
25   36 --
25   37 --    {$SETC qWWNeeded := qWriteLnWin OR qDebug OR qTrace}   { Need a WriteLnWindow }
25   38 --
25   39 --    {-----------------------------------+
25   40 --    |   Uses                            |
25   41 --    +-----------------------------------}
25   42 --    USES
25   43 --        {$LOAD MacIntf.LOAD}
25   44 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
25   45 --        {$LOAD}
25   46 --            Script
25   47 --    {$IFC qDebug}
25   48 --            , UWritelnWindow        { You don't need this to use UMAUtil }
25   49 --    {$ENDC}
25   50 --            ;
25   51 --
25   52 --    {$IFC qRangeCheck}{$R+}{$ELSEC}{$R-}{$ENDC}
25   53 --
25   54 --    {$IFC qTrace}{$D++}{$ELSEC}{$IFC qNames}{$D+}{$ELSEC}{$D-}{$ENDC}{$ENDC}
25   55 --
25   56 --    {sIFC qDebug}{$N+}{sENDC}
25   57 --
25   58 --    {-----------------------------------+
25   59 --    |   Global constants                |
25   60 --    +-----------------------------------}
25   61 --    CONST
25   62 --        { Character Constants }
25   63 --        chReturn            = CHR(13);          { ASCII code for RETURN character }
25   64 --        chEnter             = CHR(3);           { ASCII code for ENTER character }
25   65 --        chBackspace         = CHR(8);           { ASCII code for BACKSPACE character }
25   66 --        chTab               = CHR(9);           { ASCII code for TAB character }
25   67 --        chSpace             = CHR(32);          { ASCII code for SPACE character }
25   68 --        chClear             = CHR(27);          { ASCII code for CLEAR key (aka ESC) }
25   69 --        chLeft              = CHR(28);          { ASCII code for left arrow }
25   70 --        chRight             = CHR(29);          { ASCII code for right arrow }
25   71 --        chUp                = CHR(30);          { ASCII code for up arrow }
25   72 --        chDown              = CHR(31);          { ASCII code for down arrow }
25   73 --
25   74 --        kSysFontName        = '';               { GetFontNum converts this to zero. }
25   75 --        kApplFontName       = 'A';              { GetFontNum converts this to one. }
25   76 --
25   77 --        kWordAlign          = TRUE;             { Constants for OffsetPtr }
25   78 --        kDontAlign          = FALSE;
25   79 --
25   80 --        kSpace8             = '        ';
25   81 --                               { 12345678 }
25   82 --
25   83 --        kHexDigits          = '0123456789ABCDEF';
25   84 --
25   85 --        kNoFileRefnum       = 0;                { an invalid file refnum; used to
25   86 --                                                    indicate an unopened file }
25   87 --
25   88 --    {$IFC qDebug}
25   89 --        { Field types for the Fields methods.  These are understood by StdFieldToString.}
25   90 --        bInteger            = -1;
25   91 --        bHexInteger         = -2;
25   92 --        bLongInt            = -3;
25   93 --        bHexLongInt         = -4;
25   94 --        bString             = -5;
25   95 --        bBoolean            = -6;
25   96 --        bChar               = -7;
25   97 --        bPointer            = -8;
25   98 --        bHandle             = -9;
```

```
25   99 --          bPoint              = -10;
25  100 --          bRect               = -11;
25  101 --          bObject             = -12;
25  102 --          bByte               = -13;
25  103 --          bCmdNumber          = -14;
25  104 --          bClass              = -15;
25  105 --          bOSType             = -16;
25  106 --          bWindowPtr          = -17;
25  107 --          bControlHandle      = -18;
25  108 --          bTEHandle           = -19;
25  109 --          bLowByte            = -20;
25  110 --          bHighByte           = -21;
25  111 --          bPattern            = -22;
25  112 --          bFixed              = -23;
25  113 --          bRgnHandle          = -24;
25  114 --          bRGBColor           = -25;
25  115 --          bTitle              = -26;
25  116 --          bGrafPtr            = -27;
25  117 --          bStyle              = -28;
25  118 --          bVCoordinate        = -29;
25  119 --          bVPoint             = -30;
25  120 --          bVRect              = -31;
25  121 --          bFontName           = -32;
25  122 --          bStringHandle       = -33;
25  123 --          bCntlAdornment      = -34;
25  124 --          bIDType             = -36;
25  125 --          bResType            = -37;
25  126 --      {$ENDC}
25  127 --
25  128 --      {-----------------------------------+
25  129 --      |   Global types                    |
25  130 --      +-----------------------------------}
25  131 --      TYPE
25  132 --          PInteger            = ^INTEGER;
25  133 --          HInteger            = ^PInteger;
25  134 --          PLongInt            = ^LONGINT;
25  135 --          HLongInt            = ^PLongInt;
25  136 --          PBoolean            = ^BOOLEAN;
25  137 --          PByte               = ^SignedByte;
25  138 --
25  139 --          String8             = STRING[8];
25  140 --          PString8            = ^String8;
25  141 --          HString8            = ^PString8;
25  142 --
25  143 --          IDType              = ResType;
25  144 --
25  145 --          adornPieces         = ( adnLineTop,     { Draw a line at the top of the extent          }
25  146 --                                  adnLineLeft,    { Draw a line on the left side of the extent    }
25  147 --                                  adnLineBottom,  { Draw a line on the bottom side of the extent  }
25  148 --                                  adnLineRight,   { Draw a line on the right side of the extent   }
25  149 --                                  adnDummy,       { Place filler…                                 }
25  150 --                                  adnOval,        { Do a FrameOval using the extent               }
25  151 --                                  adnRRect,       { Do a (16,16) FrameRoundRect using the extent  }
25  152 --                                  adnShadow );    { Draw drop shadows against framed selections   }
25  153 --
25  154 --          CntlAdornment       = SET OF adornPieces;
25  155 --
25  156 --          VCoordinate         = LONGINT;
25  157 --
25  158 --          VPoint              = RECORD
25  159 --                                  CASE INTEGER OF
25  160 --                                      0:  (v, h:      VCoordinate);
25  161 --                                      1:  (vh:        ARRAY [VHSelect] OF VCoordinate);
25  162 --                                  END;
25  163 --
25  164 --          VRect               = RECORD
25  165 --                                  CASE INTEGER OF
25  166 --                                      0: (top, left, bottom, right:   VCoordinate);
25  167 --                                      1: (topLeft, botRight:          VPoint);
25  168 --                                  END;
25  169 --
25  170 --          ConfigRecord        = RECORD
25  171 --                              { Values from SysEnvirons (Version 1) }
25  172 --                                  environsVersion:        INTEGER;
25  173 --                                  machineType:            INTEGER;
25  174 --                                  systemVersion:          INTEGER;
25  175 --                                  processor:              INTEGER;
25  176 --                                  hasFPU:                 BOOLEAN;
25  177 --                                  hasColorQD:             BOOLEAN;
25  178 --                                  keyboardType:           INTEGER;
25  179 --                                  atDrvrVersNum:          INTEGER;
25  180 --                                  sysVRefNum:             INTEGER;
25  181 --                              { Derived values }
25  182 --                                  hasROM128K:             BOOLEAN;
25  183 --                                  hasHFS:                 BOOLEAN;
25  184 --                                  hasHierarchicalMenus:   BOOLEAN;
25  185 --                                  hasScriptManager:       BOOLEAN;
25  186 --                                  hasStyleTextEdit:       BOOLEAN;
25  187 --                                  hasSoundManager:        BOOLEAN;
25  188 --                                  hasWaitNextEvent:       BOOLEAN;
25  189 --                                  hasSCSI:                BOOLEAN;
25  190 --                                  hasDesktopBus:          BOOLEAN;
25  191 --                              END;
25  192 --
25  193 --      {-----------------------------------+
25  194 --      |   Global variables                |
25  195 --      +-----------------------------------}
25  196 --      VAR
25  197 --          gConfiguration:     ConfigRecord; { The host machine's hardware configuration }
```

```
25  198 --      {$IFC qDebug}
25  199 --          gBoolString:        ARRAY [Boolean] OF String[5];
25  200 --                                  { Used to display boolean values. }
25  201 --      {$J+}
25  202 --          GFIELDTOSTRRTN:     ProcPtr;    { Routine that converts fields to strings.  }
25  203 --      {$J-}
25  204 --      {$ENDC}
25  205 --          gRGBBlack,                      { RGB representation for black.             }
25  206 --          gRGBWhite:          RGBColor;   { RGB representation for white.             }
25  207 --
25  208 --
25  209 --      { M I S C E L L A N E O U S   U T I L I T I E S }
25  210 --
25  211 --      {$%+}
25  212 --      FUNCTION  %_GetA5: LONGINT; INLINE $2E8D;       { MOVE.L A5,(A7) }
25  213 --      FUNCTION  %_GetA6: LONGINT; INLINE $2E8E;       { MOVE.L A6,(A7) }
25  214 --      FUNCTION  %_GetA7: LONGINT; INLINE $2E8F;       { MOVE.L A7,(A7) }
25  215 --      {$%-}
25  216 --
25  217 --      {-----------------------------------+
25  218 --      |   Global Procedures                |
25  219 --      +-----------------------------------}
25  220 --      PROCEDURE CopyStr255 (VAR fmStr: Str255; toAddr: UNIV Ptr);
25  221 --          { Copies a string, copying ONLY what's necessary }
25  222 --
25  223 --      FUNCTION  PRStr (astr: Str255): Ptr; INLINE $2E9F;  { Move.L (A7)+,(A7) }
25  224 --          { For brain-damaged compiler, which can't pass the address of a string in a packed record,
25  225 --              even though it has to be byte-aligned by definition.  Maybe it's not word aligned,
25  226 --              but for a BlockMove we don't care.
25  227 --            When calling CopyStr255 where the destination is the address of a Str255 in a packed
25  228 --              record, call it like this:
25  229 --                  CopyStr255(fromStr, PRStr(packedStr)); }
25  230 --
25  231 --      PROCEDURE DefaultSize (VAR theSize: INTEGER);
25  232 --          { Returns 0 if theSize equals the default font size. }
25  233 --
25  234 --      PROCEDURE DefineConfiguration (VAR configuration: ConfigRecord);
25  235 --          { Fills the configuration record for the host machine. }
25  236 --
25  237 --      PROCEDURE DisposIfHandle(aHandle: UNIV Handle);
25  238 --          { Disposes the handle only if it is non-NIL. }
25  239 --
25  240 --      FUNCTION  EqualBlocks(first, second: UNIV Ptr; theSize: INTEGER): BOOLEAN;
25  241 --          { Returns true if the two blocks pointed at by first and second are
25  242 --              equal over theSize bytes.}
25  243 --
25  244 --      {$IFC qDebug}
25  245 --      PROCEDURE FieldToString (theData: Ptr; fieldType: INTEGER; VAR theString: Str255);
25  246 --          { Calls the routine whose address is stored in gFieldToStrRtn.  Its purpose
25  247 --              is to take some data pointed to by theData, and whose type is indicated
25  248 --              by field type, and convert that data into a string representation.  By
25  249 --              default, gFieldToStrRtn points to StdFieldToString. }
25  250 --      {$ENDC}
25  251 --
25  252 --      FUNCTION  GetFontNum (fontName: Str255): INTEGER;
25  253 --          { Returns the font number corresponding to the given name.  If fontName is
25  254 --              equal to kSysFontName then zero is returned.  If fontName is equal to
25  255 --              kApplFontName then 1 is returned.  Otherwise the Toolbox routine GetFNum
25  256 --              is called to get the font number. }
25  257 --
25  258 --      FUNCTION  GetHandleBits(h: Handle): SignedByte;
25  259 --          { Return the flag byte of a handle }
25  260 --
25  261 --      PROCEDURE GetIfColor(VAR aColor: RGBColor);
25  262 --          { Fetches current foreground color.  Works for both old and new QuickDraw. }
25  263 --
25  264 --      PROCEDURE GetPortFontInfo (fontNum: INTEGER; VAR fontName: Str255; VAR fontSize: INTEGER);
25  265 --          { Returns the font name corresponding to a given name, in portable format.
25  266 --              i.e. if fontNum is the system font then kSysFontName is returned; if fontNum is
25  267 --              the application font then kApplFontName is return; otherwise the Toolbox routine
25  268 --              GetFontName is called to get the font name.  Then, if fontNum is the system or
25  269 --              application font, a 0 is returned if the size is the default size. }
25  270 --
25  271 --      FUNCTION  GetResLoad: BOOLEAN; INLINE $1EB8, $0A5E; { MOVE.B        $A5E,(A7) }
25  272 --          { Returns the ResLoad flag from low memory.  True indicates that resources
25  273 --              are loaded on GetResource. }
25  274 --
25  275 --      FUNCTION GetTrapType (theTrap: INTEGER): TrapType;
25  276 --          { Returns the trap's type (OSTrap or ToolTrap). }
25  277 --
25  278 --      FUNCTION IntMultiply (x, y: INTEGER): LONGINT;
25  279 --          INLINE  $301F,      { MOVE.W        (A7)+,D0 }
25  280 --                  $C1DF,      { MULS.W        (A7)+,D0 }
25  281 --                  $2E80;      { MOVE.L        D0,(A0) }
25  282 --          { Multiplies two integers and returns a longint result.  Note that the
25  283 --              Pascal compiler will generate a 16-bit result if you say n := x * y,
25  284 --              truncating the upper 16-bits.  Furthermore, if either x or y is a
25  285 --              longint then the compiler expands both operands to 32-bits and calls
25  286 --              a 32-bit multiply subroutine.  IntMultiply avoids these problems. }
25  287 --
25  288 --      FUNCTION  LengthRect(r: Rect; vhs: VHSelect): INTEGER;
25  289 --          { If vhs is v, returns the length of r; else returns the
25  290 --              width of r. }
25  291 --
25  292 --      PROCEDURE LIntToHex (decNumber: LONGINT; VAR hexNumber: String8;
25  293 --                           noOfDigits: INTEGER);
25  294 --          { LIntToHex converts decNumber to a hexidecimal string of
25  295 --              noOfDigits length. }
25  296 --
```

```
25  297 --      FUNCTION  Max(a, b: LONGINT): LONGINT;
25  298 --          { Returns the maximum of a and b. }
25  299 --
25  300 --      FUNCTION  Min(a, b: LONGINT): LONGINT;
25  301 --          { Returns the minimum of a and b. }
25  302 --
25  303 --      FUNCTION NewSysPtr (noOfBytes: LONGINT): Ptr;
25  304 --          { Creates a new point of noOfBytes length in the system heap.}
25  305 --
25  306 --      PROCEDURE NumberToHex (theNumber: LONGINT; VAR hexString: Str255;
25  307 --                             hexDigits: INTEGER);
25  308 --          { Converts theNumber to a hex string preceeded with '$'.}
25  309 --
25  310 --      FUNCTION  PinOnRect(theRect: Rect; thePt: Point): LONGINT;
25  311 --          { Like PinRect except that if thePt.h (or v) is >= theRect.right (bottom)
25  312 --              it returns theRect.right (bottom).  PinRect returns 1 less than that
25  313 --              except if thePt is exactly on the edge of theRect.}
25  314 --
25  315 --      PROCEDURE PointerToHex (theNumber: LONGINT; VAR hexString: Str255;
25  316 --                              hexDigits: INTEGER);
25  317 --          { If theNumber is zero, then hexString is set to 'Nil'.  Else theNumber
25  318 --              is converted to a hex string preceeded with '$'.}
25  319 --
25  320 --      FUNCTION  RectsNest(outer, inner: Rect): BOOLEAN;
25  321 --          { Determine if inner nests within outer. }
25  322 --
25  323 --      FUNCTION  RoundUp(aNumber: LONGINT; aModulus: INTEGER): LONGINT;
25  324 --          { Rounds aNumber up so that it is evenly divisible by aModulus. }
25  325 --
25  326 --      PROCEDURE SetKeyScript (newKeyScript: INTEGER);
25  327 --          { If newKeyScript is different from the current key script, then the
25  328 --              key script is set to newKeyScript.  (The reason we don't want to
25  329 --              set the key script to the same thing is that the Script Mgr.
25  330 --              does a FlushEvents when it sets the key script.) }
25  331 --
25  332 --      PROCEDURE SetHandleBits(h: Handle; theBits: SignedByte);
25  333 --          { Sets the flag byte of a handle }
25  334 --
25  335 --      PROCEDURE SetIfColor(aColor: RGBColor);
25  336 --          { Sets foreground color.  Works for both old and new QuickDraw. }
25  337 --
25  338 --      PROCEDURE SetPortTextStyle (theTextStyle: TextStyle);
25  339 --          { Sets the font style of the current port to the characteristics of
25  340 --              theTextStyle. }
25  341 --
25  342 --      PROCEDURE SetRGBColor (VAR RGB: RGBColor; red, green, blue: INTEGER);
25  343 --          { Sets RGB to the given colors. }
25  344 --
25  345 --      PROCEDURE SetTextStyle(VAR theTextStyle: TextStyle; theFont: INTEGER;
25  346 --                             theStyle: Style; theSize: INTEGER; theColor: RGBColor);
25  347 --          { Sets theTextStyle to the given characteristics }
25  348 --
25  349 --      {$IFC qDebug}
25  350 --      PROCEDURE StdFieldToString (theData: Ptr; fieldType: INTEGER; VAR theString: Str255);
25  351 --          { This routine converts all field types defined by the constants in this unit. }
25  352 --      {$ENDC}
25  353 --
25  354 --      {$IFC qDebug}
25  355 --      PROCEDURE TextStyleFields (aTitle: Str255; VAR aStyle: TextStyle;
25  356 --                                 PROCEDURE DoToField (fieldName: Str255;
25  357 --                                                      fieldAddr: Ptr;
25  358 --                                                      fieldType: INTEGER));
25  359 --          { Used to inspect the fields of a TextStyle record.  The
25  360 --              record is passed as a var parameter because we must be passed
25  361 --              the address of the original record being referenced.  Passing
25  362 --              by value would simply make a copy of the record, which doesn't
25  363 --              work for the inspector. }
25  364 --      {$ENDC}
25  365 --
25  366 --      FUNCTION UprChar (ch: CHAR): CHAR;
25  367 --          { Returns ch converted to upper case }
25  368 --
25  369 --      PROCEDURE UprString8 (VAR s: String8);
25  370 --          { Converts s to all upper case. }
25  371 --
25  372 --      PROCEDURE UseROMMap(resLoad: BOOLEAN);
25  373 --          { Call this before any Resource Manager call for which you might like to use
25  374 --              ROM resources.  Remember, if you pass FALSE to this then you should
25  375 --              call SetResLoad(TRUE) afterwards. }
25  376 --
25  377 --
25  378 --      { F I L E   U T I L I T I E S }
25  379 --
25  380 --      FUNCTION  CloseFile (dataRefnum, rsrcRefnum: INTEGER): OSErr;
25  381 --          { Closes the data and resource forks of a file.  If dataRefnum =
25  382 --              kNoFileRefnum then the data fork is not closed.  Likewise, if
25  383 --              rsrcRefnum = kNoFileRefnum then the resource fork is not closed.
25  384 --              Returns noErr if successful, else an O.S. error. }
25  385 --
25  386 --      FUNCTION  DeleteFile(namePtr: StringPtr; volRefnum: INTEGER): OSErr;
25  387 --          { Deletes the specified file; uses FillInDirID to bypass the
25  388 --              Poor Man's Search Path. }
25  389 --
25  390 --      FUNCTION  FileModDate(name: Str255; volRefnum: INTEGER): LONGINT;
25  391 --          { Returns file modification date or 0 if an I/O error occurs }
25  392 --
25  393 --      FUNCTION  FillInDirID(pb: HParmBlkPtr): OSErr;
25  394 --          { Based on the ioVRefnum field of pb^, looks up the dirID of the
25  395 --              working directory and fills it into the ioDirID field.  If HFS
```

```
25  396 --              is not installed, it sets the ioDirID to 0.
25  397 --
25  398 --          This is used to inhibit the PMSP.  After setting up your HParamBLockRec,
25  399 --              call this to fill in the dirID and then make the H form of the
25  400 --              call (eg., PBHGetFInfo rather than PBGetFInfo). }
25  401 --
25  402 --      FUNCTION  GetDirID(VAR vRefnum: INTEGER; VAR dirID: LONGINT): OSErr;
25  403 --          { Returns the dirID corresponding to a given wdRefnum; changes vRefnum
25  404 --              to be a real volume refnum. }
25  405 --
25  406 --      FUNCTION GetFileInfo(name: Str255; volRefnum: INTEGER; VAR info: HParamBlockRec): OSErr;
25  407 --          { Makes a PBHGetFInfo call, and returns result in info.  Bypasses PMSP. }
25  408 --
25  409 --      FUNCTION  NumBlocks(numBytes: LONGINT; blkSize: LONGINT): LONGINT;
25  410 --          { Returns the number of blocks required to store numBytes, given that
25  411 --              the block size is blkSize. }
25  412 --
25  413 --      FUNCTION OpenFile(name: Str255; volRefnum: INTEGER;
25  414 --              openData, openRsrc: BOOLEAN; dataPerm, rsrcPerm: INTEGER;
25  415 --              VAR dataRefnum, rsrcRefnum: INTEGER): OSErr;
25  416 --          { Open the specified forks of the file using the specified permissions.
25  417 --              Returns kNoFileRefnum for refnums that are not (or cannot) be opened.
25  418 --              On 64K ROM machines, rsrcPerm is ignored.  Returns an O/S error if
25  419 --              the data fork does not exist; returns a rsrcRefnum of kNoFileRefnum
25  420 --              if the resource fork does not exist. Returns an O/S error for all
25  421 --              other errors.  If no error occurs, returns NoErr. }
25  422 --
25  423 --
25  424 --      { D E B U G G I N G   U T I L I T I E S }
25  425 --
25  426 --      {$IFC qDebug}
25  427 --      FUNCTION  CanReadLn: BOOLEAN;
25  428 --          { returns TRUE if WritelnWindow is active and the display is not suppressed }
25  429 --
25  430 --      PROCEDURE GetMethodName (ppc: LONGINT; VAR s: Str255);
25  431 --          { GetMethodName returns the name of the method (or procedure) in
25  432 --              which ppc points. }
25  433 --
25  434 --      PROCEDURE GetProcName(pPC: LONGINT; VAR className, procName: String8);
25  435 --          { GetProcName returns the name of the procedure or function in
25  436 --              which ppc points.  If it is in a method, then it return's
25  437 --              the name of the method's class in className. }
25  438 --
25  439 --      PROCEDURE MacsBug; INLINE $A9FF;
25  440 --          { Causes a program break to MacsBug. }
25  441 --
25  442 --      PROCEDURE MacsBugStr(s: Str255); INLINE $ABFF;
25  443 --          { Causes a program break to MacsBug--MacsBug displays the message s. }
25  444 --
25  445 --      FUNCTION ReadInteger (prompt: Str255): INTEGER;
25  446 --          { Displays prompt and reads an integer from the debug window. }
25  447 --
25  448 --      FUNCTION ReadYesNo (prompt: Str255): BOOLEAN;
25  449 --          { Displays prompt and accepts a 'y' or 'n' as input from the debug
25  450 --              window. }
25  451 --
25  452 --      PROCEDURE WritePt(pt: Point);
25  453 --          { Writes the point in the debug window. }
25  454 --
25  455 --      PROCEDURE WritePtr(val: UNIV LONGINT);
25  456 --          { Writes the pointer, in hex, in the debug window. }
25  457 --
25  458 --      PROCEDURE WriteRect(r: Rect);
25  459 --          { Writes the rectangle in the debug window. }
25  460 --
25  461 --      PROCEDURE WriteBoolean(b: BOOLEAN);
25  462 --          { Writes the boolean in the debug window. }
25  463 --
25  464 --      PROCEDURE WriteVPt (pt: VPoint);
25  465 --          { Writes the VPoint in the debug window. }
25  466 --
25  467 --      PROCEDURE WriteVRect (r: VRect);
25  468 --          { Writes the VRect in the debug window. }
25  469 --
25  470 --      PROCEDURE WrLblPt(aLabel: Str255; pt: Point);
25  471 --          { Writes the label, ' = ', the point, to the debug window. }
25  472 --
25  473 --      PROCEDURE WrLblPtr(aLabel: Str255; val: UNIV LONGINT);
25  474 --          { Writes the label, ' = ', the pointer in hex, to the debug window. }
25  475 --
25  476 --      PROCEDURE WrLblRect(aLabel: Str255; r: Rect);
25  477 --          { Writes the label, ' = ', the rectangle, to the debug window. }
25  478 --
25  479 --      PROCEDURE WrLblBoolean(aLabel: Str255; b: BOOLEAN);
25  480 --          { Writes the label, ' = ', the boolean, to the debug window. }
25  481 --
25  482 --      PROCEDURE WrLblVPt(aLabel: Str255; pt: VPoint);
25  483 --          { Writes the label, ' = ', the VPoint in the debug window. }
25  484 --
25  485 --      PROCEDURE WrLblVRect (aLabel: Str255; r: VRect);
25  486 --          { Writes the lable, ' = ', the VRect in the debug window. }
25  487 --
25  488 --      PROCEDURE WriteSig(theID: IDType);
25  489 --
25  490 --      PROCEDURE WrLblSig(theLabel: Str255; theID: IDType);
25  491 --
25  492 --      PROCEDURE WriteHexInt(theInt: INTEGER);
25  493 --
25  494 --      PROCEDURE WrLblHexInt(theLabel: Str255; theInt: INTEGER);
```

```
25  495 --
25  496 --      PROCEDURE WriteHexLongint(theLongint: LONGINT);
25  497 --
25  498 --      PROCEDURE WrLblHexLongint(theLabel: Str255; theLongint: LONGINT);
25  499 --      {$ENDC qDebug}
25  500 --
25  501 --
25  502 --      IMPLEMENTATION
25  503 --
25  504 --      {$I UMAUtil.incl.p}
```

```
26    1 --      {$P}
26    2 --      {UMAUtil.incl.p}
26    3 --      {Copyright © 1984-1988  Apple Computer, Inc.  All rights reserved.}
26    4 --
26    5 --      {$IFC qDebug}
26    6 --      {$IFC qTrace}{$D+}{$ENDC}
26    7 --      {$S MADebug}
26    8 --      {-----------------------------------+
26    9 --      |   CanReadLn                       |
26   10 --      +-----------------------------------}
26   11 -- A    FUNCTION  CanReadLn: BOOLEAN;
26   12 --
26   13 0- A    BEGIN
26   14 --          CanReadLn := FALSE;
26   15 --          IF gDebugWindowPtr <> NIL THEN
26   16 --              CanReadLn := gWrToWindow;
26   17 -0 A    END {CanReadLn};
26   18 --      {$IFC qTrace}{$D++}{$ENDC}
26   19 --      {$ENDC qDebug}
26   20 --
26   21 --      {$S MAFile}
26   22 --      {-----------------------------------+
26   23 --      |   CloseFile                       |
26   24 --      +-----------------------------------}
26   25 -- A    FUNCTION  CloseFile (dataRefnum, rsrcRefnum: INTEGER): OSErr;
26   26 --
26   27 --      VAR
26   28 --          err:    OSErr;
26   29 --
26   30 0- A    BEGIN
26   31 --          err := noErr;
26   32 --
26   33 --          IF dataRefnum <> kNoFileRefnum THEN
26   34 --              err := FSClose(dataRefnum);
26   35 --
26   36 --          IF rsrcRefnum <> kNoFileRefnum THEN
26   37 1-              BEGIN
26   38 --              CloseResFile(rsrcRefnum);
26   39 --              IF err = noErr THEN
26   40 --                  err := ResError;
26   41 -1              END;
26   42 --
26   43 --          CloseFile := err;
26   44 -0 A    END {CloseFile};
26   45 --
26   46 --      {$S MAMain}
26   47 --      {-----------------------------------+
26   48 --      |   CopyStr255                      |
26   49 --      +-----------------------------------}
26   50 -- A    PROCEDURE CopyStr255 (VAR fmStr: Str255; toAddr: UNIV Ptr);
26   51 0- A    BEGIN
26   52 --          BlockMove(@fmStr, toAddr, LENGTH(fmStr)+1);
26   53 -0 A    END;
26   54 --
26   55 --      {$S MAMain}
26   56 --      {-----------------------------------+
26   57 --      |   DefaultSize                     |
26   58 --      +-----------------------------------}
26   59 -- A    PROCEDURE DefaultSize (VAR theSize: INTEGER);
26   60 --      CONST
26   61 --          SysFontSize = $BA8;
26   62 0- A    BEGIN
26   63 --          IF gConfiguration.hasROM128K &
26   64 --            (gConfiguration.hasScriptManager & (theSize = GetDefFontSize)) |
26   65 --            (theSize = PInteger(SysFontSize)^) THEN
26   66 --              theSize := 0;
26   67 -0 A    END;
26   68 --
26   69 --      {$IFC qTrace}{$D+}{$ENDC}
26   70 --      {$S MAInit}
26   71 --      {-----------------------------------+
26   72 --      |   DefineConfiguration             |
26   73 --      +-----------------------------------}
26   74 -- A    PROCEDURE DefineConfiguration (VAR configuration: ConfigRecord);
26   75 --
26   76 --      CONST
26   77 --          {Low Memory Globals}
26   78 --          ROM85 =             $28E;
26   79 --          HWCfgFlags =        $B22;
26   80 --          FSFCBLen =          $3F6;
26   81 --
26   82 --          {Masks for the HWCfgFlags}
26   83 --          mSCSIPort =         $8000;
26   84 --          mDesktopBus =       $0400;
26   85 --
26   86 --          {Traps used to determine the presence of features}
26   87 --          UnimplementedTrap = $9F;
26   88 --          tScriptUtil =       $A8B5;          { determines presence of Script Manager }
26   89 --          tPopupMenuSelect =  $A80B;          { determines whether ROM has hier. menu support }
26   90 --          tSndDoCommand =     $A803;          { determines presence of Sound Manager }
26   91 --          tWaitNextEvent =    $A860;          { determines presence of WaitNextEvent }
26   92 --
26   93 --      VAR
26   94 --          kludge:             ^SysEnvRec;
26   95 --          result:             OSErr;
26   96 --
26   97 --          {-------------------------------+
26   98 --          |   TrapExists                  |
```

```
26   99 --          +--------------------------------}
26  100 -- B       FUNCTION  TrapExists (theTrap: INTEGER): BOOLEAN;
26  101 --
26  102 0- B       BEGIN
26  103 --             TrapExists := GetTrapAddress(UnimplementedTrap) <>
26  104 --                           NGetTrapAddress(theTrap, GetTrapType(theTrap));
26  105 -0 B       END {TrapExists};
26  106 --
26  107 0- A   BEGIN
26  108 --         kludge := @configuration;
26  109 --         result := SysEnvirons(1, kludge^);      {Version 1 shouldn't fail}
26  110 --
26  111 --         WITH configuration DO
26  112 1-             BEGIN
26  113 --             hasDesktopBus :=        BAND(pInteger(HwCfgFlags)^, mDesktopBus) > 0;
26  114 --             hasSCSI :=              BAND(pInteger(HwCfgFlags)^, mSCSIPort) > 0;
26  115 --             hasROM128K :=           machineType > envMac;
26  116 --             IF hasROM128K THEN
26  117 --                 hasHFS := TRUE
26  118 --             ELSE
26  119 --                 hasHFS :=               PInteger(FSFCBLen)^ > 0;
26  120 --             hasHierarchicalMenus := hasROM128K AND TrapExists(tPopupMenuSelect);
26  121 --             hasScriptManager :=     hasROM128K AND TrapExists(tScriptUtil);
26  122 --             hasStyleTextEdit :=     systemVersion >= $600;
26  123 --             hasSoundManager :=      hasROM128K AND TrapExists(tSndDoCommand);
26  124 --             hasWaitNextEvent :=     hasROM128K AND TrapExists(tWaitNextEvent);
26  125 -1             END {with};
26  126 -0 A   END {DefineConfiguration};
26  127 --     {$IFC qTrace}{$D++}{$ENDC}
26  128 --
26  129 --     {$S MAFile}
26  130 --     {-----------------------------------+
26  131 --     |   DeleteFile                      |
26  132 --     +-----------------------------------}
26  133 -- A   FUNCTION  DeleteFile(namePtr: StringPtr; volRefnum: INTEGER): OSErr;
26  134 --         VAR hPB:    HParamBlockRec;
26  135 --             err:    OSErr;
26  136 0- A   BEGIN
26  137 --         WITH hPB DO
26  138 1-             BEGIN
26  139 --             ioNamePtr := namePtr;
26  140 --             ioVRefnum := volRefnum;
26  141 --             ioFVersNum := 0;
26  142 -1             END;
26  143 --
26  144 --         err := FillInDirID(@hPB);       {to avoid PMSP}
26  145 --
26  146 --         IF err = noErr THEN
26  147 --             err := PBHDelete(@hPB, FALSE);
26  148 --
26  149 --         DeleteFile := err;
26  150 -0 A   END {DeleteFile};
26  151 --
26  152 --     {$S MAMain}
26  153 --     {-----------------------------------+
26  154 --     |   DisposIfHandle                  |
26  155 --     +-----------------------------------}
26  156 -- A   PROCEDURE DisposIfHandle (aHandle: UNIV Handle);
26  157 --
26  158 0- A   BEGIN
26  159 --         IF aHandle <> NIL THEN
26  160 --             DisposHandle(aHandle);
26  161 -0 A   END {DisposIfHandle};
26  162 --
26  163 --     {-----------------------------------+
26  164 --     |   EqualBlocks                     |
26  165 --     +-----------------------------------}
26  166 -- A   FUNCTION  EqualBlocks(first, second: UNIV Ptr; theSize: INTEGER): BOOLEAN; EXTERNAL;
26  167 --
26  168 --     {$S MAFile}
26  169 --     {-----------------------------------+
26  170 --     |   FileModDate                     |
26  171 --     +-----------------------------------}
26  172 -- A   FUNCTION  FileModDate(name: Str255; volRefnum: INTEGER): LONGINT;
26  173 --         VAR pb: HParamBlockRec;
26  174 --
26  175 0- A   BEGIN
26  176 --         IF GetFileInfo(name, volRefnum, pb) = noErr THEN
26  177 --             FileModDate := pb.ioFlMdDat
26  178 --         ELSE
26  179 --             FileModDate := 0;
26  180 -0 A   END {FileModDate};
26  181 --
26  182 --     {$IFC qDebug}
26  183 --     {-----------------------------------+
26  184 --     |   FieldToString                   |
26  185 --     +-----------------------------------}
26  186 -- A   PROCEDURE FieldToString (theData: Ptr; fieldType: INTEGER; VAR theString: Str255); EXTERNAL;
26  187 --     {$ENDC}
26  188 --
26  189 --
26  190 --     {$S MAFile}
26  191 --     {-----------------------------------+
26  192 --     |   FillInDirID                     |
26  193 --     +-----------------------------------}
26  194 -- A   FUNCTION  FillInDirID(pb: HParmBlkPtr): OSErr;
26  195 --
26  196 0- A   BEGIN
26  197 --         FillInDirID := GetDirID(pb^.ioVRefnum, pb^.ioDirID);
```

```
26  198 -0 A   END {FillInDirID};
26  199 --
26  200 --     {$S MAFile}
26  201 --     {-----------------------------------+
26  202 --     |   GetDirID                        |
26  203 --     +-----------------------------------}
26  204 -- A   FUNCTION  GetDirID(VAR vRefnum: INTEGER; VAR dirID: LONGINT): OSErr;
26  205 --         VAR pb: WDPBRec;
26  206 --
26  207 0- A   BEGIN
26  208 --         IF gConfiguration.hasHFS THEN
26  209 1-             BEGIN
26  210 --             WITH pb DO
26  211 2-                 BEGIN
26  212 --                 ioNamePtr := NIL;
26  213 --                 ioVRefnum := vRefnum;
26  214 --                 ioWDIndex := 0;
26  215 --                 ioWDProcID := 0;
26  216 --                 ioWDVRefnum := vRefnum;
26  217 -2                 END;
26  218 --             GetDirID := PBGetWDInfo(@pb, FALSE);
26  219 --             vRefnum := pb.ioWDVRefnum;
26  220 --             dirID := pb.ioWDDirID;
26  221 -1             END
26  222 --         ELSE
26  223 1-             BEGIN
26  224 --             dirID := 0;
26  225 --             GetDirID := noErr;
26  226 -1             END;
26  227 -0 A   END {GetDirID};
26  228 --
26  229 --     {$S MAFile}
26  230 --     {-----------------------------------+
26  231 --     |   GetFileInfo                     |
26  232 --     +-----------------------------------}
26  233 -- A   FUNCTION  GetFileInfo(name: Str255; volRefnum: INTEGER; VAR info: HParamBlockRec): OSErr;
26  234 --         VAR err:    OSErr;
26  235 0- A   BEGIN
26  236 --         WITH info DO
26  237 1-             BEGIN
26  238 --             ioNamePtr := @name;
26  239 --             ioVRefnum := volRefnum;
26  240 --             ioFVersNum := 0;
26  241 --             ioFDirIndex := 0;
26  242 -1             END;
26  243 --         err := FillInDirId(@info);
26  244 --         IF err = noErr THEN
26  245 --             err := PBHGetFInfo(@info, FALSE);
26  246 --         GetFileInfo := err;
26  247 -0 A   END {GetFileInfo};
26  248 --
26  249 --     {$S MAMain}
26  250 --     {-----------------------------------+
26  251 --     |   GetFontNum                      |
26  252 --     +-----------------------------------}
26  253 -- A   FUNCTION GetFontNum (fontName: Str255): INTEGER;
26  254 --
26  255 --     VAR
26  256 --         fontNum:    INTEGER;
26  257 --
26  258 0- A   BEGIN
26  259 --         UprString(fontName, FALSE);
26  260 --         IF fontName = kSysFontName THEN
26  261 --             fontNum := systemFont
26  262 --         ELSE IF fontName = kApplFontName THEN
26  263 --             fontNum := applFont
26  264 --         ELSE
26  265 --             GetFNum(fontName, fontNum);
26  266 --         GetFontNum := fontNum;
26  267 -0 A   END;
26  268 --
26  269 --     {$S Main} {Must be in Main segment and cannot call to any other segment.}
26  270 --     {$IFC qTrace}{$D+}{$ENDC}
26  271 --     {-----------------------------------+
26  272 --     |   GetHandleBits                   |
26  273 --     +-----------------------------------}
26  274 -- A   FUNCTION  GetHandleBits(h: Handle): SignedByte;
26  275 --
26  276 0- A   BEGIN
26  277 --     {$IFC NOT qNeedsROM128K}
26  278 --         IF NOT gConfiguration.hasROM128K THEN
26  279 --             GetHandleBits := PByte(h)^
26  280 --         ELSE
26  281 --     {$ENDC}
26  282 --             GetHandleBits := HGetState(h);
26  283 -0 A   END;
26  284 --     {$IFC qTrace}{$D++}{$ENDC}
26  285 --
26  286 --     {$S MAMain}
26  287 --     {-----------------------------------+   Get present foreground color
26  288 --     |   GetIfColor                      |       Works with both old and new QuickDraw
26  289 --     +-----------------------------------}
26  290 -- A   PROCEDURE GetIfColor (VAR aColor: RGBColor);
26  291 --     CONST
26  292 --         BlackBit    = 5;
26  293 --         YellowBit   = 6;
26  294 --         MagentaBit  = 7;
26  295 --         CyanBit     = 8;
26  296 --     VAR
```

```
26  297 --          oldColor:   LONGINT;
26  298 0- A    BEGIN
26  299 --          IF gConfiguration.hasColorQD THEN
26  300 --              GetForeColor(aColor)
26  301 --          ELSE
26  302 1-              BEGIN                                   { Map old, dumb CMYB system to RGB color    }
26  303 --              (*                      xxxxxxx C.MY B rgb w b  = RGB
26  304 --              blackColor      =  33 = 0000000 0.00 1 000 0 1  = 000
26  305 --              whiteColor      =  30 = 0000000 0.00 0 111 1 0  = 111
26  306 --              redColor        = 205 = 0000000 0.11 0 011 0 1  = 100
26  307 --              greenColor      = 341 = 0000000 1.01 0 101 0 1  = 010
26  308 --              blueColor       = 409 = 0000000 1.10 0 110 0 1  = 001
26  309 --              cyanColor       = 273 = 0000000 1.00 0 100 0 1  = 011
26  310 --              magentaColor    = 137 = 0000000 0.10 0 010 0 1  = 101
26  311 --              yellowColor     =  69 = 0000000 0.01 0 001 0 1  = 110
26  312 --              *)
26  313 --              oldColor := thePort^.fgColor;           { Fetch old color                           }
26  314 --              aColor := gRGBBlack;                    { Prime returned color to black             }
26  315 --              IF BTST(oldColor, BlackBit) THEN        { If color isn't black, force CMY = 111      }
26  316 --                  oldColor := BOR(oldColor, $1C0);
26  317 --              IF NOT BTST(oldColor, CyanBit) THEN     { Absence of cyan = presence of red         }
26  318 --                  aColor.red := $FFFF;
26  319 --              IF NOT BTST(oldColor, MagentaBit) THEN  { Absence of magenta = presence of green    }
26  320 --                  aColor.green := $FFFF;
26  321 --              IF NOT BTST(oldColor, YellowBit) THEN   { Absence of yellow = presence of blue      }
26  322 --                  aColor.blue := $FFFF;
26  323 -1              END;
26  324 -0 A    END;
26  325 --
26  326 --      {$IFC qDebug}
26  327 --      {$S MADebug}
26  328 --      {-----------------------------------+
26  329 --      |   GetMethodName                   |
26  330 --      +-----------------------------------}
26  331 -- A    PROCEDURE GetMethodName (ppc: LONGINT; VAR s: Str255);
26  332 --
26  333 --      VAR
26  334 --          c:      String8;
26  335 --          m:      String8;
26  336 --          x:      INTEGER;
26  337 --
26  338 0- A    BEGIN
26  339 --          GetProcName(ppc, c, m);
26  340 --
26  341 --          IF c = kSpace8 THEN
26  342 --              s := m
26  343 --          ELSE
26  344 --              s := CONCAT(c, '.', m);
26  345 1-          REPEAT
26  346 --              x := POS(' ', s);
26  347 --              IF x > 0 THEN
26  348 --                  Delete(s, x, 1);
26  349 -1          UNTIL x = 0;
26  350 -0 A    END {GetMethodName};
26  351 --      {$ENDC qDebug}
26  352 --
26  353 --      {$S MAMain}
26  354 --      {-----------------------------------+
26  355 --      |   GetPortFontInfo                 |
26  356 --      +-----------------------------------}
26  357 -- A    PROCEDURE GetPortFontInfo (fontNum: INTEGER; VAR fontName: Str255; VAR fontSize: INTEGER);
26  358 --      CONST
26  359 --          ApFontID =      $984;
26  360 --          SysFontFam =    $BA6;
26  361 --
26  362 0- A    BEGIN
26  363 --          IF (fontNum = systemFont) |
26  364 --            (gConfiguration.hasROM128K &
26  365 --              ((gConfiguration.hasScriptManager & (fontNum = GetSysFont)) |
26  366 --                (fontNum = PInteger(SysFontFam)^))) THEN
26  367 1-              BEGIN
26  368 --              fontName := kSysFontName;
26  369 --              DefaultSize(fontSize);
26  370 -1              END
26  371 --
26  372 --          ELSE IF (fontNum = applFont) |
26  373 --            (gConfiguration.hasScriptManager & (fontNum = GetAppFont)) |
26  374 --            (fontNum = PInteger(ApFontID)^) THEN
26  375 1-              BEGIN
26  376 --              fontName := kApplFontName;
26  377 --              DefaultSize(fontSize);
26  378 -1              END
26  379 --
26  380 --          ELSE
26  381 --              GetFontName(fontNum, fontName);
26  382 -0 A    END;
26  383 --
26  384 --      {$IFC qDebug}
26  385 --      {$MADebug}
26  386 --      {-----------------------------------+
26  387 --      |   GetProcName                     |
26  388 --      +-----------------------------------}
26  389 -- A    PROCEDURE GetProcName(pPC: LONGINT; VAR className, procName: String8);
26  390 --
26  391 --      VAR
26  392 --          pname:              PByte;
26  393 --          pc:                 PInteger;
26  394 --          bothClassAndProc:   BOOLEAN;
26  395 --
```

```
26  396 --          {--------------------------------+
26  397 --          |   CopyName                     |
26  398 --          +--------------------------------}
26  399 -- B       PROCEDURE CopyName(VAR anyName: String8);
26  400 --
26  401 --          VAR
26  402 --              j: INTEGER;
26  403 --
26  404 0- B        BEGIN
26  405 --              anyName := kSpace8;
26  406 --              FOR j := 1 TO 8 DO
26  407 1-                  BEGIN
26  408 --                  anyName[j] := CHR(BAND(pname^, 127));
26  409 --                  pname := PByte(ORD(pname) + 1);
26  410 -1                  END;
26  411 -0 B        END;
26  412 --
26  413 0- A    BEGIN
26  414 --          className := kSpace8;
26  415 --          procName := kSpace8;
26  416 --
26  417 --          pc := HInteger(pPC)^;
26  418 --
26  419 --          IF (ORD(pc) <> 0) AND NOT ODD(ORD(pc)) THEN
26  420 1-              BEGIN
26  421 --              {We add the -1 to the following tests so that the things we are
26  422 --                  searching for don't appear in the body of the procedure.}
26  423 --
26  424 --              {Search for UNLK A6.}
26  425 --              WHILE pc^ - 1 <> $4E5E - 1 DO
26  426 --                  pc := PInteger(ORD(pc) + 2);
26  427 --
26  428 --              {Search for RTS or JMP (A0) or RTD #$00nn}  {Thanks to Mike Bentley}
26  429 --              WHILE (pc^ - 1 <> $4E75 - 1) &
26  430 --                    (pc^ - 1 <> $4ED0 - 1) &
26  431 --                    (pc^ - 1 <> $4E74 - 1) DO
26  432 --                  pc := PInteger(ORD(pc) + 2);
26  433 --
26  434 --              IF pc^ - 1 = $4E74 - 1 THEN                {If it was RTD, skip past displacement}
26  435 --                  pc := PInteger(ORD4(pc) + 2);
26  436 --              pname := PByte(ORD(pc) + 2); {pname read and changed by CopyName}
26  437 --              IF pname^ < 0 THEN
26  438 2-                  BEGIN
26  439 --                  CopyName(procName);
26  440 --                  IF PByte(ORD(pc) + 3)^ < 0 THEN
26  441 --                      CopyName(String8(className))
26  442 --                  ELSE
26  443 --                      className := kSpace8;
26  444 -2                  END;
26  445 --
26  446 -1              END;
26  447 -0 A    END {GetProcName};
26  448 --      {$ENDC qDebug}
26  449 --
26  450 --      {$S Main}
26  451 --      {----------------------------------+
26  452 --      |   GetTrapType                    |
26  453 --      +----------------------------------}
26  454 -- A    FUNCTION  GetTrapType (theTrap: INTEGER): TrapType;
26  455 --
26  456 0- A    BEGIN
26  457 --          { OS traps start with A0, Tool with A8 or AA. }
26  458 --          IF BAND(theTrap, $0F00) = 0 THEN
26  459 --              GetTrapType := OSTrap
26  460 --          ELSE
26  461 --              GetTrapType := ToolTrap;
26  462 -0 A    END {GetTrapType};
26  463 --
26  464 --      {$S MAMain}
26  465 --      {----------------------------------+
26  466 --      |   LengthRect                     |
26  467 --      +----------------------------------}
26  468 -- A    FUNCTION  LengthRect(r: Rect; vhs: VHSelect): INTEGER;
26  469 --
26  470 0- A    BEGIN
26  471 --          LengthRect := r.botRight.vh[vhs] - r.topLeft.vh[vhs];
26  472 -0 A    END {LengthRect};
26  473 --
26  474 --      {$IFC qTrace}{$D+}{$ENDC}
26  475 --      {$S MADebug}
26  476 --      {----------------------------------+
26  477 --      |   LIntToHex                      |
26  478 --      +----------------------------------}
26  479 -- A    PROCEDURE LIntToHex (decNumber: LONGINT; VAR hexNumber: String8;
26  480 --                           noOfDigits: INTEGER);
26  481 --      VAR
26  482 --          i:          INTEGER;
26  483 --
26  484 0- A    BEGIN
26  485 --          noOfDigits := Min(noOfDigits, 8);
26  486 --          hexNumber[0] := CHR(noOfDigits);
26  487 --          FOR i := noOfDigits DOWNTO 1 DO
26  488 1-              BEGIN
26  489 --              hexNumber[i] := kHexDigits[BAND(decNumber,15) + 1];
26  490 --              decNumber := BSR(decNumber,4);
26  491 -1              END;
26  492 -0 A    END {LIntToHex};
26  493 --      {$IFC qTrace}{$D++}{$ENDC}
26  494 --
```

```
26  495 --      {$IFC qTrace}{$D+}{$ENDC}
26  496 --      {$S MAMain}
26  497 --      {-----------------------------------+
26  498 --      |   Max                             |
26  499 --      +-----------------------------------}
26  500 -- A    FUNCTION  Max(a, b: LONGINT): LONGINT;
26  501 0- A    BEGIN
26  502 --          IF a < b THEN
26  503 --              Max := b
26  504 --          ELSE
26  505 --              Max := a;
26  506 -0 A    END {Max};
26  507 --      {$IFC qTrace}{$D++}{$ENDC}
26  508 --
26  509 --      {$IFC qTrace}{$D+}{$ENDC}
26  510 --      {$S MAMain}
26  511 --      {-----------------------------------+
26  512 --      |   Min                             |
26  513 --      +-----------------------------------}
26  514 -- A    FUNCTION  Min(a, b: LONGINT): LONGINT;
26  515 0- A    BEGIN
26  516 --          IF a < b THEN
26  517 --              Min := a
26  518 --          ELSE
26  519 --              Min := b;
26  520 -0 A    END {Min};
26  521 --      {$IFC qTrace}{$D++}{$ENDC}
26  522 --
26  523 --      {-----------------------------------+
26  524 --      |   NumberToHex                     |
26  525 --      +-----------------------------------}
26  526 -- A    FUNCTION NewSysPtr (noOfBytes: LONGINT): Ptr; EXTERNAL;
26  527 --
26  528 --      {$IFC qTrace}{$D+}{$ENDC}
26  529 --      {$S MADebug}
26  530 --      {-----------------------------------+
26  531 --      |   NumberToHex                     |
26  532 --      +-----------------------------------}
26  533 -- A    PROCEDURE NumberToHex (theNumber: LONGINT; VAR hexString: Str255; hexDigits: INTEGER);
26  534 --
26  535 --      VAR
26  536 --          tempString:     String8;
26  537 --
26  538 0- A    BEGIN
26  539 --          LIntToHex(theNumber, tempString, hexDigits);
26  540 --          hexString := CONCAT('$', tempString);
26  541 -0 A    END {NumberToHex};
26  542 --      {$IFC qTrace}{$D++}{$ENDC}
26  543 --
26  544 --      {$IFC qTrace}{$D+}{$ENDC}
26  545 --      {$S MADebug}
26  546 --      {-----------------------------------+
26  547 --      |   PointerToHex                    |
26  548 --      +-----------------------------------}
26  549 -- A    PROCEDURE PointerToHex (theNumber: LONGINT; VAR hexString: Str255; hexDigits: INTEGER);
26  550 --
26  551 --      VAR
26  552 --          tempString:     String8;
26  553 --
26  554 0- A    BEGIN
26  555 --          IF theNumber = 0 THEN
26  556 --              hexString := 'Nil'
26  557 --          ELSE
26  558 1-              BEGIN
26  559 --              LIntToHex(theNumber, tempString, hexDigits);
26  560 --              hexString := CONCAT('$', tempString);
26  561 -1              END;
26  562 -0 A    END {PointerToHex};
26  563 --      {$IFC qTrace}{$D++}{$ENDC}
26  564 --
26  565 --      {$S MAFile}
26  566 --      {-----------------------------------+
26  567 --      |   NumBlocks                       |
26  568 --      +-----------------------------------}
26  569 -- A    FUNCTION  NumBlocks (numBytes: LONGINT; blkSize: LONGINT): LONGINT;
26  570 --
26  571 0- A    BEGIN
26  572 --          NumBlocks := (numBytes + blkSize - 1) DIV blkSize;
26  573 -0 A    END {NumBlocks};
26  574 --
26  575 --      {$S MAFile}
26  576 --      {-----------------------------------+
26  577 --      |   OpenFile                        |
26  578 --      +-----------------------------------}
26  579 -- A    FUNCTION  OpenFile(name: Str255; volRefnum: INTEGER; openData, openRsrc: BOOLEAN;
26  580 --                                          dataPerm, rsrcPerm: INTEGER;
26  581 --                                          VAR dataRefnum, rsrcRefnum: INTEGER): OSErr;
26  582 --      VAR
26  583 --          pb:         HParamBlockRec;
26  584 --          oldVRefnum: INTEGER;
26  585 --          result:     OSErr;
26  586 --
26  587 --          {-------------------------------+
26  588 --          |   TestForError                |
26  589 --          +-------------------------------}
26  590 -- B        PROCEDURE TestForError (err: OSErr);
26  591 --
26  592 0- B        BEGIN
26  593 --              IF err <> noErr THEN
```

```
26  594 1-                  BEGIN
26  595 --                  OpenFile := err;
26  596 --                  EXIT(OpenFile);
26  597 -1                  END;
26  598 -0 B        END {TestForError};
26  599 --
26  600 0- A    BEGIN
26  601 --              {always open data fork, to establish that the file does exist}
26  602 --          WITH pb DO
26  603 1-              BEGIN
26  604 --              ioNamePtr := @name;
26  605 --              ioVRefNum := volRefnum;
26  606 --              ioVersNum := 0;
26  607 --              ioPermssn := dataPerm;
26  608 --              ioMisc := NIL;
26  609 -1              END;
26  610 --          TestForError(FillInDirID(@pb));
26  611 --
26  612 --          IF gConfiguration.hasHFS THEN
26  613 --              result := PBHOpenDeny(@pb, FALSE)           { Try the shared volume open.           }
26  614 --          ELSE
26  615 --              result := paramErr;
26  616 --          IF result = paramErr THEN                       { Not on a shared volume, try HFS open. }
26  617 1-              BEGIN
26  618 --              pb.ioPermssn := BAND(dataPerm ,3);
26  619 --              result := PBHOpen(@pb, FALSE);
26  620 -1              END;
26  621 --          TestForError(result);
26  622 --
26  623 --          IF openData THEN
26  624 --              dataRefnum := pb.ioRefnum
26  625 --          ELSE
26  626 1-              BEGIN
26  627 --                  { we did not want the data fork open, so close it now }
26  628 --              TestForError(FSClose(pb.ioRefnum));
26  629 --              dataRefnum := kNoFileRefnum;
26  630 -1              END;
26  631 --
26  632 --          IF openRsrc THEN
26  633 1-              BEGIN
26  634 --      {$IFC NOT qNeedsROM128K}
26  635 --              IF NOT gConfiguration.hasROM128K THEN
26  636 2-                  BEGIN
26  637 --                  TestForError(GetVol(NIL, oldVRefnum));
26  638 --                  TestForError(SetVol(NIL, volRefnum));
26  639 --
26  640 --                  rsrcRefnum := OpenResFile(name);
26  641 --
26  642 --                  TestForError(SetVol(NIL, oldVRefnum));
26  643 -2                  END
26  644 --              ELSE
26  645 --      {$ENDC}
26  646 2-                  BEGIN
26  647 --                  IF gConfiguration.hasHFS THEN
26  648 --                      result := PBHOpenRFDeny(@pb, FALSE)
26  649 --                  ELSE
26  650 --                      result := paramErr;
26  651 --                  IF result = paramErr THEN
26  652 3-                      BEGIN
26  653 --                      rsrcRefnum := OpenRFPerm(name, volRefnum, BAND(rsrcPerm, 7));
26  654 --                      result := ResError;
26  655 -3                      END
26  656 --                  ELSE
26  657 --                      rsrcRefNum := pb.ioRefNum;
26  658 -2                  END;
26  659 --
26  660 --              IF result <> noErr THEN
26  661 --                  rsrcRefnum := kNoFileRefnum;
26  662 -1              END
26  663 --          ELSE
26  664 --              rsrcRefnum := kNoFileRefnum;
26  665 --
26  666 --          OpenFile := noErr;
26  667 --
26  668 -0 A    END {OpenFile};
26  669 --
26  670 --      {$S MARes}
26  671 --      {-----------------------------------+
26  672 --      |   PinOnRect                       |
26  673 --      +-----------------------------------}
26  674 -- A    FUNCTION  PinOnRect(theRect: Rect; thePt: Point): LONGINT;
26  675 --
26  676 0- A    BEGIN
26  677 --          IF thePt.h < theRect.left   THEN    thePt.h := theRect.left;
26  678 --          IF thePt.h > theRect.right  THEN    thePt.h := theRect.right;
26  679 --          IF thePt.v < theRect.top    THEN    thePt.v := theRect.top;
26  680 --          IF thePt.v > theRect.bottom THEN    thePt.v := theRect.bottom;
26  681 --
26  682 --          PinOnRect := LONGINT(thePt);
26  683 -0 A    END {PinOnRect};
26  684 --
26  685 --      {$IFC qDebug}
26  686 --      {$S MADebug}
26  687 --      {-----------------------------------+
26  688 --      |   ReadInteger                     |
26  689 --      +-----------------------------------}
26  690 -- A    FUNCTION  ReadInteger (prompt: Str255): INTEGER;
26  691 --
26  692 --      VAR
```

```
26  693 --          i:          INTEGER;
26  694 --
26  695 0- A    BEGIN
26  696 --          WWForceOutput(forceOn, forceUnchanged);
26  697 --          Write(prompt);
26  698 --          Readln(i);
26  699 --          WWEndForce;
26  700 --          ReadInteger := i;
26  701 -0 A    END {ReadInteger};
26  702 --      {$ENDC qDebug}
26  703 --
26  704 --      {$IFC qDebug}
26  705 --      {$S MADebug}
26  706 --      {-----------------------------------+
26  707 --      |   ReadYesNo                       |
26  708 --      +-----------------------------------}
26  709 -- A    FUNCTION  ReadYesNo (prompt: Str255): BOOLEAN;
26  710 --
26  711 --      VAR
26  712 --          s:          Str255;
26  713 --
26  714 0- A    BEGIN
26  715 --          WWForceOutput(forceOn, forceUnchanged);
26  716 --          Write(prompt);
26  717 --          Readln(s);
26  718 --          WWEndForce;
26  719 --          ReadYesNo :=  (s <> '') & (s[1] IN ['y', 'Y']);
26  720 -0 A    END {ReadYesNo};
26  721 --      {$ENDC qDebug}
26  722 --
26  723 --      {$S MARes}
26  724 --      {-----------------------------------+
26  725 --      |   RectsNest                       |
26  726 --      +-----------------------------------}
26  727 -- A    FUNCTION  RectsNest(outer, inner: Rect): BOOLEAN;
26  728 0- A    BEGIN
26  729 --          WITH inner DO
26  730 --              RectsNest := (left >= outer.left) AND (right <= outer.right)
26  731 --                           AND (top >= outer.top) AND (bottom <= outer.bottom);
26  732 -0 A    END;
26  733 --
26  734 --      {$S MARes}
26  735 --      {-----------------------------------+
26  736 --      |   RoundUp                         |
26  737 --      +-----------------------------------}
26  738 -- A    FUNCTION  RoundUp(aNumber: LONGINT; aModulus: INTEGER): LONGINT;
26  739 --
26  740 0- A    BEGIN
26  741 --          RoundUp := ((aNumber + aModulus - 1) DIV aModulus) * aModulus;
26  742 -0 A    END {RoundUp};
26  743 --
26  744 --      {$S MAMain}
26  745 --      {-----------------------------------+
26  746 --      |   SetKeyScript                    |
26  747 --      +-----------------------------------}
26  748 -- A    PROCEDURE SetKeyScript (newKeyScript: INTEGER);
26  749 --
26  750 --      VAR
26  751 --          currentKeyScript:   INTEGER;
26  752 --
26  753 0- A    BEGIN
26  754 --          currentKeyScript := GetEnvirons(smKeyScript);
26  755 --          IF currentKeyScript <> newKeyScript THEN
26  756 --              KeyScript(newKeyScript);
26  757 -0 A    END {SetKeyScript};
26  758 --
26  759 --      {$S Main} {Must be in Main segment and cannot call to any other segment.}
26  760 --      {$IFC qTrace}{$D+}{$ENDC}
26  761 --      {-----------------------------------+
26  762 --      |   SetHandleBits                   |
26  763 --      +-----------------------------------}
26  764 -- A    PROCEDURE SetHandleBits(h: Handle; theBits: SignedByte);
26  765 --
26  766 0- A    BEGIN
26  767 --      {$IFC NOT qNeedsROM128K}
26  768 --          IF NOT gConfiguration.hasROM128K THEN
26  769 --              PByte(h)^ := theBits
26  770 --          ELSE
26  771 --      {$ENDC}
26  772 --              HSetState(h, theBits);
26  773 -0 A    END;
26  774 --      {$IFC qTrace}{$D++}{$ENDC}
26  775 --
26  776 --      {$S MAMain}
26  777 --      {-----------------------------------+   Set present foreground color
26  778 --      |   SetIfColor                      |       Works with both old and new QuickDraw
26  779 --      +-----------------------------------}
26  780 -- A    PROCEDURE SetIfColor (aColor: RGBColor);
26  781 --      CONST
26  782 --          SignBit     = 15;
26  783 --      VAR
26  784 --          index:      INTEGER;
26  785 --          oldColor:   LONGINT;
26  786 0- A    BEGIN
26  787 --          IF gConfiguration.hasColorQD THEN
26  788 --              RGBForeColor(aColor)
26  789 --          ELSE
26  790 1-              BEGIN
26  791 --              index := 0;                         { Prime index                       }
```

```
26  792 --              IF BTST(aColor.red, SignBit) THEN       { Set bit if red >= $8000                }
26  793 --                  index := 4;
26  794 --              IF BTST(aColor.green, SignBit) THEN     { Set bit if green >= $8000              }
26  795 --                  index := index + 2;
26  796 --              IF BTST(aColor.blue, SignBit) THEN      { Set bit if blue >= $8000               }
26  797 --                  index := index + 1;
26  798 2-              CASE index OF
26  799 --                  0:  oldColor := blackColor;
26  800 --                  1:  oldColor := blueColor;
26  801 --                  2:  oldColor := greenColor;
26  802 --                  3:  oldColor := cyanColor;
26  803 --                  4:  oldColor := redColor;
26  804 --                  5:  oldColor := magentaColor;
26  805 --                  6:  oldColor := yellowColor;
26  806 --                  7:  oldColor := whiteColor;
26  807 -2                 END;
26  808 --              ForeColor(oldColor);
26  809 -1              END;
26  810 -0 A    END;
26  811 --
26  812 --      {$S MAMain}
26  813 --      {-----------------------------------+
26  814 --      |   SetPortTextStyle                |
26  815 --      +-----------------------------------}
26  816 -- A    PROCEDURE SetPortTextStyle(theTextStyle: TextStyle);
26  817 --
26  818 0- A    BEGIN
26  819 --          TextFont(theTextStyle.tsFont);
26  820 --          TextFace(theTextStyle.tsFace);
26  821 --          TextSize(theTextStyle.tsSize);
26  822 --          SetIfColor(theTextStyle.tsColor);
26  823 -0 A    END {SetPortTextStyle};
26  824 --
26  825 --      {$S MAMain}
26  826 --      {-----------------------------------+
26  827 --      |   SetRGBColor                     |
26  828 --      +-----------------------------------}
26  829 -- A    PROCEDURE SetRGBColor (VAR RGB: RGBColor; red, green, blue: INTEGER);
26  830 --
26  831 0- A    BEGIN
26  832 --          RGB.red := red;
26  833 --          RGB.green := green;
26  834 --          RGB.blue := blue;
26  835 -0 A    END;
26  836 --
26  837 --      {$S MAMain}
26  838 --      {-----------------------------------+
26  839 --      |   SetTextStyle                    |   Convenience routine ala SetRect
26  840 --      +-----------------------------------}
26  841 -- A    PROCEDURE SetTextStyle(VAR theTextStyle: TextStyle; theFont: INTEGER;
26  842 --                             theStyle: Style; theSize: INTEGER; theColor: RGBColor);
26  843 0- A    BEGIN
26  844 --          WITH theTextStyle DO
26  845 1-              BEGIN
26  846 --              tsFont := theFont;
26  847 --              tsFace := theStyle;
26  848 --              tsSize := theSize;
26  849 --              tsColor := theColor;
26  850 -1              END;
26  851 -0 A    END;
26  852 --
26  853 --      {$IFC qDebug}
26  854 --      {$IFC qTrace}{$D+}{$ENDC}
26  855 --      {$S MADebug}
26  856 --      {-----------------------------------+
26  857 --      |   StdFieldToString                |
26  858 --      +-----------------------------------}
26  859 -- A    PROCEDURE StdFieldToString (theData: Ptr; fieldType: INTEGER; VAR theString: Str255);
26  860 --
26  861 --      CONST
26  862 --          adnFrame =      [adnLineTop, adnLineLeft, adnLineBottom, adnLineRight];
26  863 --
26  864 --      TYPE
26  865 --          TAlias =    RECORD
26  866 --                          CASE INTEGER OF
26  867 --                              bBoolean:       (asBoolean:     BOOLEAN);
26  868 --                              bFontName,
26  869 --                              bCmdNumber,
26  870 --                              bHighByte,
26  871 --                              bLowByte,
26  872 --                              bHexInteger,
26  873 --                              bInteger:       (asInteger:     INTEGER);
26  874 --                              bFixed,
26  875 --                              bHexLongInt,
26  876 --                              bLongInt:       (asLongInt:     LONGINT);
26  877 --                              bString:        (asString:      Str255);
26  878 --                              bChar:          (asChar:        CHAR);
26  879 --                              bGrafPtr,
26  880 --                              bWindowPtr,
26  881 --                              bPointer:       (asPointer:     Ptr);
26  882 --                              bRgnHandle,
26  883 --                              bControlHandle,
26  884 --                              bTEHandle,
26  885 --                              bHandle:        (asHandle:      Handle);
26  886 --                              bPoint:         (asPoint:       Point);
26  887 --                              bRect:          (asRect:        Rect);
26  888 --                              bObject:        (asObject:      Handle);
26  889 --                              bByte:          (asByte:        SignedByte);
26  890 --                              bOSType:        (asOSType:      OSType);
```

```
26  891 --                              bPattern:       (asPattern:     Pattern);
26  892 --                              bRGBColor:      (asRGBColor:    RGBColor);
26  893 --                              bStyle:         (asStyle:       Style);
26  894 --                              bVCoordinate:   (asVCoordinate: VCoordinate);
26  895 --                              bVPoint:        (asVPoint:      VPoint);
26  896 --                              bVRect:         (asVRect:       VRect);
26  897 --                              bStringHandle:  (asStrHandle:   StringHandle);
26  898 --                              bCntlAdornment: (asCntlAdornment: CntlAdornment);
26  899 --                      END;
26  900 --
26  901 --      VAR
26  902 --          alias:      ^TAlias;
26  903 --          aString:    Str255;
26  904 --          hexString:  String8;
26  905 --          i:          INTEGER;
26  906 --
26  907 --          {-------------------------------+
26  908 --          |   CheckStyleItem              |
26  909 --          +-------------------------------}
26  910 -- B       PROCEDURE CheckStyleItem (s: StyleItem; name: Str255);
26  911 --
26  912 0- B       BEGIN
26  913 --              IF s IN alias^.asStyle THEN
26  914 --                  IF theString = '[' THEN
26  915 --                      theString := CONCAT(theString, name)
26  916 --                  ELSE
26  917 --                      theString := CONCAT(theString, ',', name);
26  918 -0 B       END;
26  919 --
26  920 --          {-------------------------------+
26  921 --          |   CheckAdornment              |
26  922 --          +-------------------------------}
26  923 -- B       PROCEDURE CheckAdornment (p: CntlAdornment; name: Str255);
26  924 --
26  925 0- B       BEGIN
26  926 --              { "set1 <= set2" means set1 is wholly contained in set2 }
26  927 --              IF p <= alias^.asCntlAdornment THEN
26  928 --                  IF theString = '[' THEN
26  929 --                      theString := CONCAT(theString, name)
26  930 --                  ELSE
26  931 --                      theString := CONCAT(theString, ',', name);
26  932 -0 B       END;
26  933 --
26  934 0- A   BEGIN
26  935 --          alias := Pointer(theData);
26  936 --          theString := '';
26  937 --          WITH alias^ DO
26  938 1-              CASE fieldType OF
26  939 --                  bBoolean:
26  940 --                      theString := gBoolString[ORD(asBoolean) <> 0];
26  941 --                  bFontName:
26  942 --                      GetFontName(asInteger, theString);
26  943 --                  bInteger:
26  944 --                      NumToString(asInteger, theString);
26  945 --                  bLongInt:
26  946 --                      NumToString(asLongInt, theString);
26  947 --                  bHexInteger:
26  948 --                      NumberToHex(asInteger, theString, 4);
26  949 --                  bHexLongInt:
26  950 --                      NumberToHex(asLongInt, theString, 8);
26  951 --                  bHighByte:
26  952 --                      NumberToHex(BAND(asInteger, $FF00), theString, 2);
26  953 --                  bLowByte:
26  954 --                      NumberToHex(BAND(asInteger, $00FF), theString, 2);
26  955 --                  bFixed:
26  956 2-                      BEGIN
26  957 --                      NumToString(HiWrd(asLongInt), aString);
26  958 --                      NumToString(LoWrd(asLongInt), theString);
26  959 --                      theString := CONCAT(aString, ':', theString);
26  960 -2                      END;
26  961 --                  bString:
26  962 --                      theString := asString;
26  963 --                  bChar:
26  964 2-                      BEGIN
26  965 --                      theString := ' ';
26  966 --                      theString[1] := asChar;
26  967 -2                      END;
26  968 --                  bGrafPtr,
26  969 --                  bWindowPtr,
26  970 --                  bPointer:
26  971 --                      PointerToHex(ORD4(asPointer), theString, 6);
26  972 --                  bRgnHandle,
26  973 --                  bControlHandle,
26  974 --                  bTEHandle,
26  975 --                  bHandle:
26  976 --                      IF asHandle = NIL THEN
26  977 --                          theString := 'Nil'
26  978 --                      ELSE
26  979 --                          NumberToHex(ORD4(asHandle), theString, 6);
26  980 --                  bPoint:
26  981 2-                      BEGIN
26  982 --                      NumToString(asPoint.h, aString);
26  983 --                      NumToString(asPoint.v, theString);
26  984 --                      theString := CONCAT('(', aString, ', ', theString, ')');
26  985 -2                      END;
26  986 --                  bRect:
26  987 2-                      BEGIN
26  988 --                      NumToString(asRect.left, aString);
26  989 --                      NumToString(asRect.top, theString);
```

```
26  990 --                      theString := CONCAT('(', aString, ', ', theString, ')/(');
26  991 --                      NumToString(asRect.right, aString);
26  992 --                      theString := CONCAT(theString, aString, ', ');
26  993 --                      NumToString(asRect.bottom, aString);
26  994 --                      theString := CONCAT(theString, aString, ')');
26  995 -2                    END;
26  996 --                  bObject:
26  997 --                      PointerToHex(ORD4(asObject), theString, 6);
26  998 --                  bByte:
26  999 --                      NumToString(asByte, theString);
26 1000 --                  bCmdNumber:
26 1001 --                      NumToString(asInteger, theString);
26 1002 --                  bOSType:
26 1003 2-                      BEGIN
26 1004 --                      theString := '''    ''';
26 1005 --                      FOR i := 1 TO 4 DO
26 1006 --                          theString[i + 1] := asOSType[i];
26 1007 -2                      END;
26 1008 --                  bPattern:
26 1009 2-                      BEGIN
26 1010 --                      theString := '$';
26 1011 --                      FOR i := 0 TO 7 DO
26 1012 3-                          BEGIN
26 1013 --                          LIntToHex(asPattern[i], hexString, 2);
26 1014 --                          theString := CONCAT(theString, hexString);
26 1015 -3                          END;
26 1016 -2                      END;
26 1017 --                  bRGBColor:
26 1018 --                      WITH asRGBColor DO
26 1019 --                          IF (red = 0) AND (green = 0) AND (blue = 0) THEN
26 1020 --                              theString := 'Black'
26 1021 --                          ELSE IF (red = $FFFF) AND (green = $FFFF) AND (blue = $FFFF) THEN
26 1022 --                              theString := 'White'
26 1023 --                          ELSE
26 1024 2-                              BEGIN
26 1025 --                              NumberToHex(asRGBColor.red, theString, 4);
26 1026 --                              NumberToHex(asRGBColor.green, aString, 4);
26 1027 --                              theString := CONCAT(theString, '/', aString);
26 1028 --                              NumberToHex(asRGBColor.blue, aString, 4);
26 1029 --                              theString := CONCAT(theString, '/', aString);
26 1030 -2                              END;
26 1031 --                  bStyle:
26 1032 2-                      BEGIN
26 1033 --                      theString := '[';
26 1034 --                      CheckStyleItem(bold, 'bold');
26 1035 --                      CheckStyleItem(italic, 'italic');
26 1036 --                      CheckStyleItem(underline, 'underline');
26 1037 --                      CheckStyleItem(outline, 'outline');
26 1038 --                      CheckStyleItem(shadow, 'shadow');
26 1039 --                      CheckStyleItem(condense, 'condense');
26 1040 --                      CheckStyleItem(extend, 'extend');
26 1041 --                      theString := CONCAT(theString, ']');
26 1042 -2                      END;
26 1043 --                  bVCoordinate:
26 1044 --                      NumToString(asVCoordinate, theString);
26 1045 --                  bVPoint:
26 1046 2-                      BEGIN
26 1047 --                      NumToString(asVPoint.h, aString);
26 1048 --                      NumToString(asVPoint.v, theString);
26 1049 --                      theString := CONCAT('(', aString, ', ', theString, ')');
26 1050 -2                      END;
26 1051 --                  bVRect:
26 1052 2-                      BEGIN
26 1053 --                      NumToString(asVRect.left, aString);
26 1054 --                      NumToString(asVRect.top, theString);
26 1055 --                      theString := CONCAT('(', aString, ', ', theString, ')/(');
26 1056 --                      NumToString(asVRect.right, aString);
26 1057 --                      theString := CONCAT(theString, aString, ', ');
26 1058 --                      NumToString(asVRect.bottom, aString);
26 1059 --                      theString := CONCAT(theString, aString, ')');
26 1060 -2                      END;
26 1061 --                  bStringHandle:
26 1062 --                      IF asStrHandle = NIL THEN
26 1063 --                          theString := 'Nil'
26 1064 --                      ELSE
26 1065 --                          theString := asStrHandle^^;
26 1066 --                  bCntlAdornment:
26 1067 2-                      BEGIN
26 1068 --                      theString := '[';
26 1069 --                      IF adnFrame <= alias^.asCntlAdornment THEN
26 1070 --                          CheckAdornment(adnFrame, 'frame')
26 1071 --                      ELSE
26 1072 3-                          BEGIN
26 1073 --                          CheckAdornment([adnLineTop], 'top');
26 1074 --                          CheckAdornment([adnLineLeft], 'left');
26 1075 --                          CheckAdornment([adnLineBottom], 'bottom');
26 1076 --                          CheckAdornment([adnLineRight], 'right');
26 1077 -3                          END;
26 1078 --                      (* CheckAdornment(adnPatFill, 'fill'); *)
26 1079 --                      CheckAdornment([adnOval], 'oval');
26 1080 --                      CheckAdornment([adnRRect], 'rrect');
26 1081 --                      CheckAdornment([adnShadow], 'shadow');
26 1082 --                      theString := CONCAT(theString, ']');
26 1083 -2                      END;
26 1084 -1                  END;
26 1085 -0 A   END {StdFieldToString};
26 1086 --     {$IFC qTrace}{$D++}{$ENDC}
26 1087 --     {$ENDC qDebug}
26 1088 --
```

```
26 1089 --      {$IFC qDebug}
26 1090 --      {$S MAFields}       { Must be in MAFields as we use this routine to make MAFields resident }
26 1091 --      {-----------------------------------+
26 1092 --      |   TextStyleFields                 |
26 1093 --      +-----------------------------------}
26 1094 -- A    PROCEDURE TextStyleFields (aTitle: Str255; VAR aStyle: TextStyle;
26 1095 --                                 PROCEDURE DoToField (fieldName: Str255;
26 1096 --                                                      fieldAddr: Ptr;
26 1097 --                                                      fieldType: INTEGER));
26 1098 0- A    BEGIN
26 1099 --          DoToField(aTitle, NIL, bTitle);
26 1100 --          DoToField('  Font', @aStyle.tsFont, bFontName);
26 1101 --          DoToField('  Face', @aStyle.tsFace, bStyle);
26 1102 --          DoToField('  Size', @aStyle.tsSize, bInteger);
26 1103 --          DoToField('  Color', @aStyle.tsColor, bRGBColor);
26 1104 -0 A    END;
26 1105 --      {$ENDC}
26 1106 --
26 1107 --      {$IFC qTrace}{$D+}{$ENDC}
26 1108 --      {$S MAMain}
26 1109 --      {-----------------------------------+
26 1110 --      |   UprChar                         |
26 1111 --      +-----------------------------------}
26 1112 -- A    FUNCTION  UprChar (ch: CHAR): CHAR;
26 1113 --
26 1114 0- A    BEGIN
26 1115 --          IF (ch >= 'a') AND (ch <= 'z') THEN
26 1116 --              UprChar := CHR(ORD(ch) - 32)
26 1117 --          ELSE
26 1118 --              UprChar := ch;
26 1119 -0 A    END {UprChar};
26 1120 --      {$IFC qTrace}{$D++}{$ENDC}
26 1121 --
26 1122 --      {$IFC qTrace}{$D+}{$ENDC}
26 1123 --      {$S MAMain}
26 1124 --      {-----------------------------------+
26 1125 --      |   UprString8                      |
26 1126 --      +-----------------------------------}
26 1127 -- A    PROCEDURE UprString8 (VAR s: String8);
26 1128 --
26 1129 --      VAR
26 1130 --          i:      INTEGER;
26 1131 --
26 1132 0- A    BEGIN
26 1133 --          FOR i := 1 TO LENGTH(s) DO
26 1134 --              IF (s[i] IN ['a'..'z']) THEN
26 1135 --                  s[i] := CHR(ORD(s[i]) - 32)
26 1136 -0 A    END {UprString8};
26 1137 --      {$IFC qTrace}{$D++}{$ENDC}
26 1138 --
26 1139 --      {$S MAMain}
26 1140 --      {$IFC qTrace}{$D+}{$ENDC}
26 1141 --      {-----------------------------------+
26 1142 --      |   UseROMMap                       |
26 1143 --      +-----------------------------------}
26 1144 -- A    PROCEDURE UseROMMap(resLoad: BOOLEAN);
26 1145 --          CONST   mapTrue =       $FFFF;
26 1146 --                  mapFalse =      $FF00;
26 1147 --                  ROMMapInsert =  $0B9E;
26 1148 --
26 1149 --          VAR p:  PInteger;
26 1150 0- A    BEGIN
26 1151 --      {$IFC NOT qNeedsROM128K}
26 1152 --          IF NOT gConfiguration.hasROM128K THEN
26 1153 --              SetResLoad(resLoad)
26 1154 --          ELSE
26 1155 --      {$ENDC}
26 1156 1-              BEGIN
26 1157 --              p := PInteger(ROMMapInsert);
26 1158 --              IF resLoad THEN
26 1159 --                  p^ := mapTrue
26 1160 --              ELSE
26 1161 --                  p^ := mapFalse;
26 1162 -1              END;
26 1163 -0 A    END {UseROMMap};
26 1164 --      {$IFC qTrace}{$D++}{$ENDC}
26 1165 --
26 1166 --      {$IFC qDebug}
26 1167 --      {$IFC qTrace}{$D+}{$ENDC}
26 1168 --      {$S MADebug}
26 1169 --      {-----------------------------------+
26 1170 --      |   WritePt                         |
26 1171 --      +-----------------------------------}
26 1172 -- A    PROCEDURE WritePt (pt: Point);
26 1173 --
26 1174 0- A    BEGIN
26 1175 --          Write('(', pt.h:1, ', ', pt.v:1, ')');
26 1176 -0 A    END {WritePt};
26 1177 --
26 1178 --      {$S MADebug}
26 1179 --      {-----------------------------------+
26 1180 --      |   WrLblPt                         |
26 1181 --      +-----------------------------------}
26 1182 -- A    PROCEDURE WrLblPt(aLabel: Str255; pt: Point);
26 1183 --
26 1184 0- A    BEGIN
26 1185 --          Write(aLabel, ' = '); WritePt(pt);
26 1186 -0 A    END {WrLblPt};
26 1187 --      {$IFC qTrace}{$D++}{$ENDC}
```

```
26 1188 --
26 1189 --      {$IFC qTrace}{$D+}{$ENDC}
26 1190 --      {$S MADebug}
26 1191 --      {-----------------------------------+
26 1192 --      |   WritePtr                        |
26 1193 --      +-----------------------------------}
26 1194 -- A    PROCEDURE WritePtr (val: UNIV LONGINT);
26 1195 --
26 1196 --      VAR
26 1197 --          hex:    String8;
26 1198 --
26 1199 0- A    BEGIN
26 1200 --          LintToHex(val, hex, 6);
26 1201 --          Write('$', hex);
26 1202 -0 A    END {WritePtr};
26 1203 --
26 1204 --      {$S MADebug}
26 1205 --      {-----------------------------------+
26 1206 --      |   WrLblPtr                        |
26 1207 --      +-----------------------------------}
26 1208 -- A    PROCEDURE WrLblPtr (aLabel: Str255; val: UNIV LONGINT);
26 1209 --
26 1210 0- A    BEGIN
26 1211 --          Write(aLabel, ' = '); WritePtr(val);
26 1212 -0 A    END {WrLblPtr};
26 1213 --      {$IFC qTrace}{$D++}{$ENDC}
26 1214 --
26 1215 --      {$IFC qTrace}{$D+}{$ENDC}
26 1216 --      {$S MADebug}
26 1217 --      {-----------------------------------+
26 1218 --      |   WriteRect                       |
26 1219 --      +-----------------------------------}
26 1220 -- A    PROCEDURE WriteRect (r: Rect);
26 1221 --
26 1222 0- A    BEGIN
26 1223 --          WritePt(r.topLeft);
26 1224 --          Write('/');
26 1225 --          WritePt(r.botRight);
26 1226 -0 A    END {WriteRect};
26 1227 --
26 1228 --      {$S MADebug}
26 1229 --      {-----------------------------------+
26 1230 --      |   WrLblRect                       |
26 1231 --      +-----------------------------------}
26 1232 -- A    PROCEDURE WrLblRect (aLabel: Str255; r: Rect);
26 1233 --
26 1234 0- A    BEGIN
26 1235 --          Write(aLabel, ' = '); WriteRect(r);
26 1236 -0 A    END {WrLblRect};
26 1237 --      {$IFC qTrace}{$D++}{$ENDC}
26 1238 --
26 1239 --      {$IFC qTrace}{$D+}{$ENDC}
26 1240 --      {$S MADebug}
26 1241 --      {-----------------------------------+
26 1242 --      |   WriteBoolean                    |
26 1243 --      +-----------------------------------}
26 1244 -- A    PROCEDURE WriteBoolean (b: BOOLEAN);
26 1245 --
26 1246 0- A    BEGIN
26 1247 --          IF b THEN
26 1248 --              Write('TRUE')
26 1249 --          ELSE
26 1250 --              Write('FALSE');
26 1251 -0 A    END {WriteBoolean};
26 1252 --
26 1253 --      {$S MADebug}
26 1254 --      {-----------------------------------+
26 1255 --      |   WrLblBoolean                    |
26 1256 --      +-----------------------------------}
26 1257 -- A    PROCEDURE WrLblBoolean (aLabel: Str255; b: BOOLEAN);
26 1258 --
26 1259 0- A    BEGIN
26 1260 --          Write(aLabel);
26 1261 --          WriteBoolean(b);
26 1262 -0 A    END {WrLblBoolean};
26 1263 --      {$IFC qTrace}{$D++}{$ENDC}
26 1264 --
26 1265 --      {$IFC qTrace}{$D+}{$ENDC}
26 1266 --      {$S MADebug}
26 1267 --      {-----------------------------------+
26 1268 --      |   WriteVPt                        |
26 1269 --      +-----------------------------------}
26 1270 -- A    PROCEDURE WriteVPt (pt: VPoint);
26 1271 --
26 1272 0- A    BEGIN
26 1273 --          Write('(', pt.h:1, ', ', pt.v:1, ')');
26 1274 -0 A    END {WriteVPt};
26 1275 --
26 1276 --      {$S MADebug}
26 1277 --      {-----------------------------------+
26 1278 --      |   WrLblVPt                        |
26 1279 --      +-----------------------------------}
26 1280 -- A    PROCEDURE WrLblVPt(aLabel: Str255; pt: VPoint);
26 1281 --
26 1282 0- A    BEGIN
26 1283 --          Write(aLabel, ' = '); WriteVPt(pt);
26 1284 -0 A    END {WrLblVPt};
26 1285 --      {$IFC qTrace}{$D++}{$ENDC}
26 1286 --
```

```
26 1287 --      {$IFC qTrace}{$D+}{$ENDC}
26 1288 --      {$S MADebug}
26 1289 --      {-----------------------------------+
26 1290 --      |   WriteVRect                      |
26 1291 --      +-----------------------------------}
26 1292 -- A    PROCEDURE WriteVRect (r: VRect);
26 1293 --
26 1294 0- A    BEGIN
26 1295 --          WriteVPt(r.topLeft);
26 1296 --          Write('/');
26 1297 --          WriteVPt(r.botRight);
26 1298 -0 A    END {WriteVRect};
26 1299 --
26 1300 --      {$S MADebug}
26 1301 --      {-----------------------------------+
26 1302 --      |   WrLblVRect                      |
26 1303 --      +-----------------------------------}
26 1304 -- A    PROCEDURE WrLblVRect (aLabel: Str255; r: VRect);
26 1305 --
26 1306 0- A    BEGIN
26 1307 --          Write(aLabel, ' = '); WriteVRect(r);
26 1308 -0 A    END {WrLblVRect};
26 1309 --      {$IFC qTrace}{$D++}{$ENDC}
26 1310 --
26 1311 --      {$IFC qTrace}{$D+}{$ENDC}
26 1312 --      {$S MADebug}
26 1313 --      {-----------------------------------+
26 1314 --      |   WriteSig                        |
26 1315 --      +-----------------------------------}
26 1316 -- A    PROCEDURE WriteSig(theID: IDType);
26 1317 --      VAR
26 1318 --          theSig: STRING[4];
26 1319 0- A    BEGIN
26 1320 --          theSig := '....';
26 1321 --          IF LONGINT(theID) <> 0 THEN
26 1322 --              BlockMove(@theID, Ptr(ORD4(@theSig)+1), SIZEOF(IDType));
26 1323 --          Write('''', theSig, '''');
26 1324 -0 A    END;
26 1325 --
26 1326 --      {$S MADebug}
26 1327 --      {-----------------------------------+
26 1328 --      |   WrLblSig                        |
26 1329 --      +-----------------------------------}
26 1330 -- A    PROCEDURE WrLblSig(theLabel: Str255; theID: IDType);
26 1331 0- A    BEGIN
26 1332 --          Write(theLabel, ' = '); WriteSig(theID);
26 1333 -0 A    END;
26 1334 --      {$IFC qTrace}{$D++}{$ENDC}
26 1335 --
26 1336 --      {$IFC qTrace}{$D+}{$ENDC}
26 1337 --      {$S MADebug}
26 1338 --      {-----------------------------------+
26 1339 --      |   WriteHexInt                     |
26 1340 --      +-----------------------------------}
26 1341 -- A    PROCEDURE WriteHexInt(theInt: INTEGER);
26 1342 --      VAR
26 1343 --          tempString:     String8;
26 1344 0- A    BEGIN
26 1345 --          LIntToHex(theInt, tempString, 4);
26 1346 --          Write('$', tempString);
26 1347 -0 A    END {WriteHexInt};
26 1348 --
26 1349 --      {$S MADebug}
26 1350 --      {-----------------------------------+
26 1351 --      |   WrLblHexInt                     |
26 1352 --      +-----------------------------------}
26 1353 -- A    PROCEDURE WrLblHexInt(theLabel: Str255; theInt: INTEGER);
26 1354 0- A    BEGIN
26 1355 --          Write(theLabel, ' = '); WriteHexInt(theInt);
26 1356 -0 A    END;
26 1357 --      {$IFC qTrace}{$D++}{$ENDC}
26 1358 --
26 1359 --      {$IFC qTrace}{$D+}{$ENDC}
26 1360 --      {$S MADebug}
26 1361 --      {-----------------------------------+
26 1362 --      |   WriteHexLongint                 |
26 1363 --      +-----------------------------------}
26 1364 -- A    PROCEDURE WriteHexLongint(theLongint: LONGINT);
26 1365 --      VAR
26 1366 --          tempString:     String8;
26 1367 0- A    BEGIN
26 1368 --          LIntToHex(theLongint, tempString, 8);
26 1369 --          Write('$', tempString);
26 1370 -0 A    END;
26 1371 --
26 1372 --      {$S MADebug}
26 1373 --      {-----------------------------------+
26 1374 --      |   WrLblHexLongint                 |
26 1375 --      +-----------------------------------}
26 1376 -- A    PROCEDURE WrLblHexLongint(theLabel: Str255; theLongint: LONGINT);
26 1377 0- A    BEGIN
26 1378 --          Write(theLabel, ' = '); WriteHexLongint(theLongint);
26 1379 -0 A    END;
26 1380 --      {$IFC qTrace}{$D++}{$ENDC}
26 1381 --      {$ENDC qDebug}
25  505 --
25  506 --      END { UMAUtil }.
```

```
27    1 --    {$P}
27    2 --    { UMemory.p }
27    3 --    { Copyright 1985-1988 by Apple Computer, Inc.  All rights reserved. }
27    4 --
27    5 --    { Conditional compilation flags used by this unit:
27    6 --
27    7 --        qNeedsROM128K   True skips code for Mac XL and 64K ROMs,
27    8 --                        false leaves it in.
27    9 --
27   10 --        qDebug          True includes debugging code, false ignores debugging code.
27   11 --                        True also sets $N+.
27   12 --
27   13 --        qTrace          True surrounds trap patches with $D+ and $D++ (because
27   14 --                        the default is $D++), false skips these directives.
27   15 --
27   16 --        The settings of $D, $R and $N are imported from UMAUtil.
27   17 --     }
27   18 --
27   19 --    UNIT UMemory;
27   20 --
27   21 --    {
27   22 --        This unit implements MacApp's memory management and segment management
27   23 --        schemes.
27   24 --
27   25 --        The memory management scheme works by distinguishing between "permanent"
27   26 --        and "temporary" heap allocation requests.  Permanent requests are
27   27 --        typically used for data your application allocates:  objects and
27   28 --        handles.  Temporary requests are used for code segments and Toolbox
27   29 --        resources and data, such as WDEF's and CDEF's.
27   30 --
27   31 --        Permanent memory objects are considered permanent because they
27   32 --        are not purged from memory until you explicitly dispose of or free them.
27   33 --        (Of course, the Macintosh Memory Manager will purge them if they're
27   34 --        marked purgable.)  Permanent objects are allocated with NewPermHandle.
27   35 --        This prevents the MacApp GrowZone from purging temporary objects to
27   36 --        accomodate a permanent one.  In MacApp, all TObjects and its descendants
27   37 --        are considered permanent, and are allocated with NewPermHandle.
27   38 --
27   39 --        Temporary memory objects are considered temporary because MacApp's
27   40 --        GrowZone procedure may purge them from memory to satisfy a memory
27   41 --        allocation request.  This is true regardless of whether the object is
27   42 --        marked non-purgable, although the GrowZone procedure will not purge
27   43 --        locked objects (such as code segments in use).  Typically, temporary
27   44 --        objects are marked non-purgable so that MacApp's GrowZone can control
27   45 --        when they are purged.
27   46 --
27   47 --        MacApp reserves a specific amount of heap space for temporary objects,
27   48 --        the idea being that the space reserved is large enough to handle the
27   49 --        largest number of temporary objects (e.g. code and system resources)
27   50 --        needed at any given time.  If this amount is sufficiently large, your
27   51 --        program will never fail loading a segment or system resource.  This
27   52 --        amount is defined by the internal variable gSzCodeReserve.  You can
27   53 --        retrieve its value by calling GetReserveSize.  It is initially set by
27   54 --        the 'seg!' and 'mem!' resources, and can be changed at run-time by
27   55 --        calling SetReserveSize.  The procedure BuildCodeReserve reserves the
27   56 --        space by allocating the handle gCodeReserve and setting its size to
27   57 --        gSzCodeReserve - (the total size of all temporary objects in memory).
27   58 --
27   59 --        When a temporary object is loaded into memory, the size of gCodeReserve
27   60 --        is adjusted accordingly. Permanent objects can be loaded into memory
27   61 --        only so long as there will still be at least gSzCodeReserve bytes
27   62 --        available for temporary objects.
27   63 --
27   64 --        To identify which objects in the heap are temporary, MacApp maintains
27   65 --        four lists of handles.  The objects identified by these handles are
27   66 --        considered temporary and fall under the control of MacApp's GrowZone
27   67 --        procedure.  The lists are:
27   68 --
27   69 --            gCodeSegs       A list of handles to all CODE resources in the appli-
27   70 --                            cation and system resource forks.
27   71 --
27   72 --            gSysMemList     A list handles to RAM-based system resources.  By
27   73 --                            default this lists includes all PACK, LDEF, MDEF,
27   74 --                            CDEF and WDEF resources in the system and appli-
27   75 --                            cation resource forks.  You can add other resources,
27   76 --                            such as fonts, by calling AddHandle.
27   77 --
27   78 --            gAppl1MemList,  Lists of handles to application data or resources.
27   79 --            gApp2MemList    MacApp initializes both lists to NIL.  The difference
27   80 --                            between the two lists is that handles in gAppl1MemList
27   81 --                            are purged before those in gApp2MemList.  You may
27   82 --                            add your own handles to these lists by allocating
27   83 --                            them with NewHandle and calling AddHandle for each
27   84 --                            handle to be added to the list.
27   85 --
27   86 --        The Macintosh Memory Manager calls MacApp's GrowZone procedure only when
27   87 --        all purgable objects have been purged from the heap and there is still
27   88 --        insufficient space to satisfy a memory request.  The GrowZone will
27   89 --        look through the lists of temporary objects in memory, making one
27   90 --        purgable so long as there is still at least gSzCodeReserve bytes
27   91 --        allocated to temporary memory (the total size of the temporary objects
27   92 --        in memory and the gCodeReserve handle).  The GrowZone procedure attempts
27   93 --        to never allow the size of the temporary objects and reserve to fall
27   94 --        below gSzCodeReserve, thereby guarenteeing that space is always avail-
27   95 --        able for code segments and system resources (provided gSzCodeReserve is
27   96 --        large enough to handle your application at its most memory intensive
27   97 --        state).
27   98 --
```

```
27   99 --          The value of gSzCodeReserve is determined at startup-time by adding up
27  100 --          the size of all the segments named in the 'seg!' resources, and adding
27  101 --          the first value of each of the 'mem!' resources.  You can derive these
27  102 --          resources by observing your program and using the MacApp debugger to
27  103 --          help you determine when your application uses the largest amount of
27  104 --          code and system resources.  Typically, we've found that this happens
27  105 --          while printing on the LaserWriter, or during initialization or term-
27  106 --          ination.
27  107 --
27  108 --          MacApp maintains another reserve, known as the low memory reserve.  This
27  109 --          is a kind of emergency reserve--when all else fails we release the
27  110 --          gMemReserve handle.  Its size is initially determined by the second
27  111 --          number of each 'mem!' resource, and can be changed by call
27  112 --          SetReserveSize.  You can retrieve its size by calling GetReserveSize.
27  113 --          Internally, the low-memory reserve is allocated with the gMemReserve
27  114 --          handle, and its size is stored in gSzMemReserve.
27  115 --
27  116 --          The function InitUMemory is responsible for initially setting up the
27  117 --          temporary and low-memory reserves, setting up the GrowZone procedure,
27  118 --          initializing handle lists, and patching LoadSeg.  It returns true if
27  119 --          the temporary reserve could be allocated.
27  120 --       }
27  121 --
27  122 --
27  123 --     INTERFACE
27  124 --
27  125 --     {-----------------------------------+
27  126 --     |   Uses                            |
27  127 --     +-----------------------------------}
27  128 --     USES
27  129 --         {$LOAD MacIntf.LOAD}
27  130 --             MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
27  131 --         {$LOAD}
27  132 --             UMAUtil,
27  133 --             UFailure,
27  134 --             UPatch
27  135 --     {$IFC qDebug}
27  136 --             , UTrace
27  137 --     {$ENDC}
27  138 --             ;
27  139 --
27  140 --     {-----------------------------------+
27  141 --     |   Global constants                |
27  142 --     +-----------------------------------}
27  143 --     {$IFC NOT qDebug}
27  144 --     CONST
27  145 --         gUnloadAllSegs      =       TRUE;   { Always TRUE when not debugging }
27  146 --     {$ENDC}
27  147 --
27  148 --     {-----------------------------------+
27  149 --     |   Global types                    |
27  150 --     +-----------------------------------}
27  151 --     TYPE
27  152 --         HHandleList = ^PHandleList;
27  153 --         PHandleList = ^AHandleList;
27  154 --         AHandleList = ARRAY[1..5000] OF Handle;     { A list of handles }
27  155 --
27  156 --     {-----------------------------------+
27  157 --     |   Global variables                |
27  158 --     +-----------------------------------}
27  159 --     VAR
27  160 --         gMaxLockedRsrc:     LONGINT;
27  161 --     {$IFC qDebug}
27  162 --         gMemMgtBreak:       BOOLEAN;        { if TRUE, break into debugger rather than
27  163 --                                                 just report memory mgt. information }
27  164 --         gRsrcReport:        BOOLEAN;        { report resource maximums }
27  165 --         gSegReport:         BOOLEAN;        { report segment loadings }
27  166 --     {$ENDC}
27  167 --         gSysMemList:        HHandleList;    { List of system handles used to
27  168 --                                               compute current allocated temporary
27  169 --                                               memory (all PACK, LDEF, MDEF, CDEF,
27  170 --                                               WDEF resources) }
27  171 --         gApp1MemList:       HHandleList;
27  172 --         gApp2MemList:       HHandleList;
27  173 --             { These start out NIL.  You can create lists of handles and place
27  174 --               then in either of these variables.  (One list might be permanent
27  175 --               handles in your application, and the other based on the current
27  176 --               situation.)  The values stored here should be a handle to a
27  177 --               list of other handles.
27  178 --
27  179 --               If you modify either of these lists, call CheckReserve.  It
27  180 --               calls BuildAllReserves to recompute the code and low space
27  181 --               reserve.  CheckReserve's result indicates whether the full code
27  182 --               reserve is present.  If not, then your program may crash because
27  183 --               a segment (or defproc) can't be loaded.
27  184 --
27  185 --               The handles in the lists should normally be resources that your
27  186 --               application might require at any given time.  All these handles
27  187 --               will normally be marked non-purgeable.  The GrowZone proc,
27  188 --               however, can purge any of these handles that are not locked.}
27  189 --
27  190 --         gCodeSegs:          HHandleList;    { List of all code segments }
27  191 --         gNonResidents:      HHandleList;    { List of non resident segments; a copy
27  192 --                                               of gCodeSegs with resident segments
27  193 --                                               replaced by NIL }
27  194 --     {$IFC qDebug}
27  195 --         gUnloadAllSegs:     BOOLEAN;        { UnloadAllSegments doesn't do anything
27  196 --                                               if this is false. }
27  197 --     {$ENDC}
```

```
27  198 --
27  199 --      {-----------------------------------+
27  200 --      |   Global routines                 |
27  201 --      +-----------------------------------}
27  202 --          { I N I T I A L I Z A T I O N }
27  203 --
27  204 --      FUNCTION InitUMemory: BOOLEAN;
27  205 --          { Initializes this unit.  Must be called before using this unit.  The
27  206 --              called must be in the program's main segment.  Sets up the
27  207 --              gCodeSegs and gSysMemList handle lists, sets the grow zone,
27  208 --              calls MaxApplZone, sets gApplMemList and gApp2MemList to NIL,
27  209 --              patches LoadSeg and allocates the temporary and low-memory
27  210 --              reserves.
27  211 --
27  212 --              Returns false if unable to allocate the temporary reserve. }
27  213 --
27  214 --
27  215 --          { S E G M E N T   L O A D I N G }
27  216 --
27  217 --          { DEBUGGING NOTE: You cannot set a MacApp breakpoint at any of these
27  218 --              routines, because they must not call anything (eg. %_BP) that may
27  219 --              require a segment load. }
27  220 --
27  221 --      FUNCTION  GetSegNumber(aProc: ProcPtr): INTEGER;
27  222 --          { GetSegNumber returns the number of the segment in which aProc resides }
27  223 --
27  224 --      FUNCTION  GetSegSize(segnum: INTEGER): Size;
27  225 --          { GetSegSize returns the size, in bytes, of the segment whose number
27  226 --              is segnum. }
27  227 --
27  228 --      FUNCTION  LoadMacAppSegment(segNum: INTEGER): LONGINT;
27  229 --          { This function patches LoadSeg.  It is called from the assembly-language
27  230 --              routine AMacAppLoadSeg, which is the actual patch setup by PatchTrap.
27  231 --              AMacAppLoadSeg saves D0-D1-A0/A1 and sets up the stack by
27  232 --              1) allocating space for LoadMacAppSegment's result, and
27  233 --              2) pushing a copy of LoadSeg's parameter onto the stack for use
27  234 --              by LoadMacAppSegment. }
27  235 --
27  236 --      PROCEDURE LoadResidentSegments;
27  237 --          { Makes resident the segments whose names are included in any 'res!'
27  238 --              resources. }
27  239 --
27  240 --      FUNCTION  PreloadSegment(segNum: INTEGER): BOOLEAN;
27  241 --          { PreloadSegment is available to programmers who want to lock a
27  242 --              segment at the top of the heap without having to call a dummy
27  243 --              procedure in that segment.  It is also useful for determining
27  244 --              whether a segment can in fact be loaded before you try to
27  245 --              execute code in it.  PreloadSegment returns TRUE if the segment
27  246 --              could be loaded, false otherwise. }
27  247 --
27  248 --      PROCEDURE SetResidentSegment(segNum: INTEGER; makeResident: BOOLEAN);
27  249 --          { SetResidentSegment can be used to make a segment resident (or no
27  250 --              longer resident); resident segments will not be unloaded by
27  251 --              UnloadAllSegments; if a segment is made resident, it is also
27  252 --              preloaded.  MacApp automatically marks its resident segment as
27  253 --              resident (the one containing the procedure CmdFromMenuItem); you
27  254 --              probably should do UnloadAllSegments before making a segment
27  255 --              resident, to ensure that it is locked at the top of the heap. }
27  256 --
27  257 --      PROCEDURE UnloadAllSegments;
27  258 --          { UnloadAllSegments unloads all segments except the blank segment
27  259 --              or the ones marked resident. }
27  260 --
27  261 --
27  262 --          { H A N D L E   L I S T   M A N A G E M E N T }
27  263 --
27  264 --      PROCEDURE AddHandle(h: Handle; toList: HHandleList);
27  265 --          { Adds a handle to the list of handles; does not check if the handle
27  266 --              already exists in the list.  Simply calls Munger to add to the
27  267 --              front of the list. }
27  268 --
27  269 --      PROCEDURE AddAllRsrc(rType: ResType; toList: HHandleList);
27  270 --          { Adds all the resources of type rType to the list.  Filters out all
27  271 --              ROM resources.  Calls AddHandle. }
27  272 --
27  273 --      PROCEDURE RemHandle(h: Handle; toList: HHandleList);
27  274 --          { Removes the handle from list by calling Munger. }
27  275 --
27  276 --      PROCEDURE ScanHandles(PROCEDURE DoToHandle(h: Handle));
27  277 --          { Calls DoToHandle for each handle in the lists gCodeSegs, gSysMemList,
27  278 --              gApplMemList and gApp2MemList.  This procedure assumes that DoToHandle
27  279 --              does not compact memory. }
27  280 --
27  281 --
27  282 --          { M E M O R Y   M A N A G E M E N T }
27  283 --
27  284 --      PROCEDURE BuildAllReserves;
27  285 --          { BuildAllReserves creates the code (temporary memory) and low space
27  286 --              reserves. }
27  287 --
27  288 --      FUNCTION  CheckReserve: BOOLEAN;
27  289 --          { Checks to see if the code reserve is OK.  Calls BuildAllReserves and
27  290 --              returns true if the full code reserve is present.  If this returns
27  291 --              false your application may bomb because a segment or system
27  292 --              resource can't be loaded. }
27  293 --
27  294 --      {$IFC qDebug}
27  295 --      PROCEDURE CheckRsrcUsage;
27  296 --          { Checks to see if the total size of the currently loaded resources
```

```
27  297 --              exceeds the maximum.  If so, the new maximum is set.  If gRsrcReport
27  298 --              is true, then the new maximum is reported in the debugger window.
27  299 --              If gMemMgtBreak then program execution is stopped. }
27  300 --      {$ENDC qDebug}
27  301 --
27  302 --      {$IFC qDebug}
27  303 --      PROCEDURE DoChangeReserve(alter: BOOLEAN;
27  304 --              VAR codeReserve, codeShort, lowSpaceReserve: LONGINT;
27  305 --              VAR gotCode, gotLowSpace: BOOLEAN);
27  306 --          { Called by the MacApp debugger.  Not normally called by an
27  307 --              application. }
27  308 --      {$ENDC qDebug}
27  309 --
27  310 --      PROCEDURE FailNoReserve;
27  311 --          { IF NOT CheckReserve THEN Failure(memFullErr, 0). }
27  312 --
27  313 --      PROCEDURE FailSpaceIsLow;
27  314 --          { IF MemSpaceIsLow THEN Failure(memFullErr, 0). }
27  315 --
27  316 --      PROCEDURE GetReserveSize (VAR szCodeReserve, szMemReserve: SIZE);
27  317 --          { Returns the amount of memory that is to be reserved for the code and
27  318 --              memory reserve.  This is not a true indication of whether this
27  319 --              amount of memory has in fact been reserved.  Call CheckReserve to
27  320 --              find out if the code reserve could be allocated, and call
27  321 --              MemSpaceIsLow to find out if the low-memory reserve could be
27  322 --              allocated. }
27  323 --
27  324 --      FUNCTION  MemSpaceIsLow: BOOLEAN;
27  325 --          { Returns TRUE if the low space reserve is missing.}
27  326 --
27  327 --      FUNCTION  NewPermHandle(logicalSize: Size): Handle;
27  328 --          { Allocates a permanent handle; you should call this instead of
27  329 --              NewHandle if allocating some permanent memory. }
27  330 --
27  331 --      FUNCTION  PermAllocation(permanent: BOOLEAN): BOOLEAN;
27  332 --          { PermAllocation controls whether subsequent memory allocations
27  333 --              are considered permanent or temporary.  Pass TRUE to setup
27  334 --              things for a permanent allocation.  Returns the previous
27  335 --              state of the permanent flag. }
27  336 --
27  337 --      PROCEDURE SetReserveSize(forCode, forOther: Size);
27  338 --          { Call this to set the size of the memory reserved for code (temp-
27  339 --              orary) and permanent (low memory) requests. }
27  340 --
27  341 --      FUNCTION  TotalTempSize(justLocked: BOOLEAN; VAR canPurge: Handle): Size;
27  342 --          { TotalTempSize returns the total number of bytes of the handles
27  343 --              in the
27  344 --          {Totals up the "appropriate" handles currently in RAM
27  345 --              (or only locked/in use handles if justLocked is TRUE).
27  346 --              canPurge is set to an unlocked handle.  Uses ScanHandles. }
27  347 --
27  348 --      {$IFC qDebug}
27  349 --      PROCEDURE WriteReserves;
27  350 --          { WRITELN's the temporary reserve and low-memory reserves in the
27  351 --              debug window. }
27  352 --      {$ENDC}
27  353 --
27  354 --
27  355 --          { U T I L I T I E S }
27  356 --
27  357 --      FUNCTION  AddSegSizes(segRsrc: Handle): LONGINT;
27  358 --          { Returns the total size of the code segments whose names are in the
27  359 --              string list segRrsc. }
27  360 --
27  361 --      PROCEDURE SetStackSpace(numBytes: LONGINT);
27  362 --          { Set the stack space to at least numBytes }
27  363 --
27  364 --
27  365 --      IMPLEMENTATION
27  366 --
27  367 --      {$I UMemory.incl.p}
```

```
28    1 --      {$P}
28    2 --      { UMemory.incl.p }
28    3 --      { Copyright 1985-1988 by Apple Computer, Inc.  All rights reserved. }
28    4 --
28    5 --      {-----------------------------------+
28    6 --      |   Local constants                 |
28    7 --      +-----------------------------------}
28    8 --      CONST
28    9 --          kGZMaxAlloc     =   $7FFFFFFF;
28   10 --
28   11 --      {-----------------------------------+
28   12 --      |   Local types                     |
28   13 --      +-----------------------------------}
28   14 --      TYPE
28   15 --          HBoolArray =        ^PBoolArray;
28   16 --          PBoolArray =        ^BoolArray;
28   17 --          BoolArray =         ARRAY [1..1000] OF BOOLEAN; { Intended to be dynamic }
28   18 --
28   19 --      {-----------------------------------+
28   20 --      |   Local variables                 |
28   21 --      +-----------------------------------}
28   22 --      VAR
28   23 --          gSegNeedsUnloading: HBoolArray; { Maintains a flag for each segment,
28   24 --                                            indicating whether the segment needs to
28   25 --                                            be unloaded, thereby optimizing
28   26 --                                            UnloadAllSegs. }
28   27 --
28   28 --          gCodeReserve:       Handle;     { Allocates temporary (code) reserve. }
28   29 --          gGZPurgeNotify:     ProcPtr;    { If non-NIL, then this will be called
28   30 --                                            before the Grow Zone proc purges a
28   31 --                                            handle.  The proc will be passed the
28   32 --                                            actual handle. }
28   33 --          gMemReserve:        Handle;     { Allocates low memory reserve. }
28   34 --          gNoOfSegments:      INTEGER;    { number of code segments }
28   35 --          gOKCodeReserve:     BOOLEAN;    { if TRUE then we have an adequate code
28   36 --                                            reserve; if FALSE then the application
28   37 --                                            could crash if memory is tight }
28   38 --          gPermAllocation:    BOOLEAN;
28   39 --          gReserveExists:     BOOLEAN;    { TRUE if the code reserve is known to
28   40 --                                            be fully allocated }
28   41 --      {$IFC qDebug}
28   42 --          gReserveShortfall:  LONGINT;    { amt. that we are lacking in the code reserve }
28   43 --      {$ENDC}
28   44 --          gSzCodeReserve:     Size;       { Attempt to reserve this much memory for
28   45 --                                            the temporary (code) reserve. }
28   46 --          gSzMemReserve:      Size;       { Attempt to reserve this much memory for
28   47 --                                            the low-memory reserve. }
28   48 --
28   49 --          gSegLoadPatch:      TrapPatch;  { patch for LoadSeg }
28   50 --
28   51 --      {-----------------------------------+
28   52 --      |   Forward declarations            |
28   53 --      +-----------------------------------}
28   54 -- A    FUNCTION  GrowZoneProc(needed: Size): LONGINT; FORWARD;
28   55 -- A    PROCEDURE BuildCodeReserve(allocLim: LONGINT; fromGZ: BOOLEAN); FORWARD;
28   56 -- A    FUNCTION  HandleIsEligible(h: Handle): BOOLEAN; FORWARD;
28   57 --
28   58 --      {-----------------------------------+
28   59 --      |   External references             |
28   60 --      +-----------------------------------}
28   61 -- A    PROCEDURE ALoadMacAppSeg; EXTERNAL;
28   62 --          { LoadSeg is TailPatched to call ALoadMacAppSeg, which in turn calls
28   63 --              LoadMacAppSegment.  ALoadMacAppSeg can only be referenced as a
28   64 --              procedure pointer, because no args are declared }
28   65 --
28   66 --
28   67 --      (*****************************************************************************************)
28   68 --      (*      G l o b a l   R o u t i n e s                                                    *)
28   69 --      (*****************************************************************************************)
28   70 --      {-----------------------------------+
28   71 --      |   CallNotify                      |
28   72 --      +-----------------------------------}
28   73 -- A    PROCEDURE CallNotify(h: Handle; routine: ProcPtr); INLINE $205F, $4E90;
28   74 --          {
28   75 --              MOVE.L      (A7)+,A0
28   76 --              JSR         (A0)
28   77 --           }
28   78 --
28   79 --
28   80 --      {-----------------------------------+
28   81 --      |   GZMoveHand                      |
28   82 --      +-----------------------------------}
28   83 -- A    FUNCTION  GZMoveHand: Handle; INLINE $2EB8, $0330;
28   84 --          {
28   85 --              MOVE.L      $330,(A7)
28   86 --           }
28   87 --
28   88 --
28   89 --      {-----------------------------------+
28   90 --      |   GZRootHand                      |
28   91 --      +-----------------------------------}
28   92 -- A    FUNCTION  GZRootHand: Handle; INLINE $2EB8, $0328;
28   93 --          {
28   94 --              MOVE.L      $328,(A7)
28   95 --           }
28   96 --
28   97 --      {$S MAInit}
28   98 --      {-----------------------------------+
```

```
28   99 --      |   AddAllRsrc                      |
28  100 --      +-----------------------------------}
28  101 -- A    PROCEDURE AddAllRsrc(rType: ResType; toList: HHandleList);
28  102 --      VAR
28  103 --          oldResLoad: BOOLEAN;
28  104 --          i:          INTEGER;
28  105 --          h:          Handle;
28  106 --          theID:      INTEGER;
28  107 --          theType:    ResType;
28  108 --          theName:    Str255;
28  109 0- A    BEGIN
28  110 --          oldResLoad := GetResLoad;
28  111 --          SetResLoad(FALSE);
28  112 --
28  113 --          FOR i := 1 TO CountResources(rType) DO
28  114 1-              BEGIN
28  115 --              h := GetIndResource(rType, i);
28  116 --              GetResInfo(h, theID, theType, theName);
28  117 --
28  118 --              { If there is a ROM resource for this type and ID, don't put it
28  119 --                  on the list. }
28  120 --              UseROMMap(FALSE);
28  121 --              h := GetResource(rType, theID);
28  122 --              UseROMMap(FALSE);
28  123 --              IF HomeResFile(h) <> 1 THEN
28  124 --                  AddHandle(h, toList);
28  125 --
28  126 -1              END;
28  127 --
28  128 --          SetResLoad(oldResLoad);
28  129 -0 A    END { AddAllRsrc };
28  130 --
28  131 --
28  132 --      {$S MAInit}
28  133 --      {-----------------------------------+
28  134 --      |   AddHandle                       |
28  135 --      +-----------------------------------}
28  136 -- A    PROCEDURE AddHandle(h: Handle; toList: HHandleList);
28  137 --      VAR
28  138 --          offset:     LONGINT;
28  139 0- A    BEGIN
28  140 --          offset := Munger(Handle(toList), 0, NIL, 0, @h, 4);
28  141 -0 A    END { AddHandle };
28  142 --
28  143 --
28  144 --      {$S MAInit}
28  145 --      {-----------------------------------+
28  146 --      |   AddSegSizes                     |
28  147 --      +-----------------------------------}
28  148 -- A    FUNCTION  AddSegSizes(segRsrc: Handle): LONGINT;
28  149 --
28  150 --      VAR
28  151 --          p:          PByte;
28  152 --          oldResLoad: BOOLEAN;
28  153 --          total:      LONGINT;
28  154 --          seg:        Handle;
28  155 --          i:          INTEGER;
28  156 --          s:          Str255;
28  157 0- A    BEGIN
28  158 --          MoveHHi(segRsrc);
28  159 --          HLock(segRsrc);
28  160 --
28  161 --          oldResLoad := GetResLoad;
28  162 --          SetResLoad(FALSE);
28  163 --
28  164 --          p := PByte(segRsrc^);
28  165 --          i := PInteger(p)^;
28  166 --          p := PByte(Ord(p) + 2);
28  167 --
28  168 --          total := 0;
28  169 --
28  170 --          WHILE i > 0 DO
28  171 1-              BEGIN
28  172 --              BlockMove(Ptr(p), @s, p^+1);
28  173 --
28  174 --              p := PByte(Ord(p) + p^ + 1);
28  175 --              i := i - 1;
28  176 --
28  177 --              seg := GetNamedResource('CODE', s);
28  178 --              IF seg <> NIL THEN
28  179 --                  total := total + SizeResource(seg) + 8;
28  180 -1              END;
28  181 --
28  182 --          AddSegSizes := total;
28  183 --
28  184 --          SetResLoad(oldResLoad);
28  185 --
28  186 --          HUnlock(segRsrc);
28  187 -0 A    END { AddSegSizes };
28  188 --
28  189 --
28  190 --      {$S MAMain}
28  191 --      {$IFC qTrace}{$D+}{$ENDC}
28  192 --      {-----------------------------------+
28  193 --      |   BuildAllReserves                |
28  194 --      +-----------------------------------}
28  195 -- A    PROCEDURE BuildAllReserves;
28  196 --      VAR
28  197 --          oldPerm:    BOOLEAN;
```

```
28  198 --
28  199 0- A   BEGIN
28  200 --              { set the permanent flag to ensure that the code reserve is
28  201 --                  actually allocated and not given up to the low space reserve }
28  202 --          oldPerm := gPermAllocation;
28  203 --          gPermAllocation := TRUE;
28  204 --
28  205 --              { make sure code reserve is OK }
28  206 --          BuildCodeReserve(kGZMaxAlloc, FALSE);
28  207 --
28  208 --              { reallocate the low space handle, if necessary }
28  209 --          IF gMemReserve^ = NIL THEN
28  210 --              ReallocHandle(gMemReserve, gSzMemReserve);
28  211 --
28  212 --              { reset the permanent flag }
28  213 --          gPermAllocation := oldPerm;
28  214 -0 A   END { BuildAllReserves };
28  215 --      {$IFC qTrace}{$D++}{$ENDC}
28  216 --
28  217 --
28  218 --      {$S MAMain}
28  219 --      {$IFC qTrace}{$D+}{$ENDC}
28  220 --      {-----------------------------------+
28  221 --      |   BuildCodeReserve                |
28  222 --      +-----------------------------------}
28  223 -- A   PROCEDURE BuildCodeReserve(allocLim: LONGINT; fromGZ: BOOLEAN);
28  224 --      VAR
28  225 --          needed:     Size;
28  226 --          avail:      Size;
28  227 --          canPurge:   Handle;
28  228 --
28  229 0- A   BEGIN
28  230 --          gOKCodeReserve := TRUE; { default value }
28  231 --
28  232 --      {$IFC qDebug}
28  233 --          gReserveShortfall := 0;
28  234 --
28  235 --          IF NOT gPermAllocation THEN
28  236 --              MacsbugStr('BuildCodeReserve called with gPermAllocation = FALSE');
28  237 --      {$ENDC qDebug}
28  238 --
28  239 --          IF NOT gReserveExists THEN
28  240 1-              BEGIN
28  241 --              gReserveExists := TRUE; { default value }
28  242 --
28  243 --                  { free the current code reserve }
28  244 --              EmptyHandle(gCodeReserve);
28  245 --
28  246 --                  { compute amt actually needed }
28  247 --              needed := Min(gSzCodeReserve - TotalTempSize(FALSE, canPurge) - 8,
28  248 --                            allocLim);
28  249 --
28  250 --              IF needed > 0 THEN
28  251 2-                  BEGIN
28  252 --                      { make as much memory available as possible }
28  253 --                  IF HandleIsEligible(gMemReserve) THEN
28  254 --                      EmptyHandle(gMemReserve);
28  255 --
28  256 --                  IF fromGZ THEN  { Never purge or compact from GrowZone }
28  257 --                      avail := allocLim
28  258 --                  ELSE
28  259 3-                      BEGIN
28  260 --                      PurgeMem(needed);
28  261 --                      avail := CompactMem(needed);
28  262 -3                      END;
28  263 --
28  264 --                  IF avail < needed THEN  { could not get the whole reserve }
28  265 3-                      BEGIN
28  266 --                      {$IFC qDebug}
28  267 --                      gReserveShortfall := needed - avail;
28  268 --                      {$ENDC}
28  269 --
28  270 --                      gOKCodeReserve := FALSE;
28  271 --                      gReserveExists := FALSE;
28  272 --
28  273 --                      needed := avail;    { get the most we can }
28  274 -3                      END;
28  275 --
28  276 --                  ReallocHandle(gCodeReserve, needed);
28  277 -2                  END;
28  278 -1              END;
28  279 -0 A   END { BuildCodeReserve };
28  280 --      {$IFC qTrace}{$D++}{$ENDC}
28  281 --
28  282 --
28  283 --      {$S MAMain}
28  284 --      {-----------------------------------+
28  285 --      |   CheckReserve                    |
28  286 --      +-----------------------------------}
28  287 -- A   FUNCTION  CheckReserve: BOOLEAN;
28  288 0- A   BEGIN
28  289 --          BuildAllReserves;
28  290 --          CheckReserve := gOKCodeReserve;
28  291 -0 A   END { CheckReserve };
28  292 --
28  293 --
28  294 --      {$IFC qDebug}
28  295 --      {$S MAMain}
28  296 --      {-----------------------------------+
```

```
28  297 --      |   CheckRsrcUsage                  |
28  298 --      +-----------------------------------}
28  299 -- A   PROCEDURE CheckRsrcUsage;
28  300 --      VAR
28  301 --          sz:         LONGINT;
28  302 --          h:          Handle;
28  303 --          s:          Str255;
28  304 0- A    BEGIN
28  305 --          sz := TotalTempSize(TRUE, h);
28  306 --          IF sz > gMaxLockedRsrc THEN
28  307 1-              BEGIN
28  308 --              gMaxLockedRsrc := sz;
28  309 --              IF gRsrcReport THEN
28  310 2-                  BEGIN
28  311 --                  NumToString(gMaxLockedRsrc, s);
28  312 --                  s := Concat('  == New maximum resources usage: ', s, ' ==');
28  313 --                  ProgramReport(s, gMemMgtBreak);
28  314 -2                  END;
28  315 -1              END;
28  316 -0 A    END { CheckRsrcUsage };
28  317 --      {$ENDC qDebug}
28  318 --
28  319 --
28  320 --      {$IFC qDebug}
28  321 --      {$S MADebug}
28  322 --      {-----------------------------------+
28  323 --      |   DoChangeReserve                 |
28  324 --      +-----------------------------------}
28  325 -- A    PROCEDURE DoChangeReserve(alter: BOOLEAN;
28  326 --                                VAR codeReserve, codeShort, lowSpaceReserve: LONGINT;
28  327 --                                VAR gotCode, gotLowSpace: BOOLEAN);
28  328 --
28  329 --      VAR
28  330 --          x:  LONGINT;
28  331 --          s:  Str255;
28  332 --
28  333 0- A    BEGIN
28  334 --          IF alter THEN
28  335 1-              BEGIN
28  336 --              Write('code reserve size = ', gSzCodeReserve:1, '  ');
28  337 --              IF gOKCodeReserve THEN
28  338 --                  Writeln(' (OK)')
28  339 --              ELSE
28  340 --                  Writeln(' (gone)');
28  341 --
28  342 --              Write('low space reserve size = ', gSzMemReserve:1, '  ');
28  343 --              IF gMemReserve^ <> NIL THEN
28  344 --                  Writeln(' (OK)')
28  345 --              ELSE
28  346 --                  Writeln(' (gone)');
28  347 --
28  348 --              Writeln;
28  349 --
28  350 --              Write('New code reserve (-1 = no change): ');
28  351 --              Readln(x);
28  352 --              IF x >= 0 THEN
28  353 --                  codeReserve := x
28  354 --              ELSE
28  355 --                  codeReserve := gSzCodeReserve;
28  356 --
28  357 --              Write('New low space reserve (-1 = no change): ');
28  358 --              Readln(x);
28  359 --              IF x >= 0 THEN
28  360 --                  lowSpaceReserve := x
28  361 --              ELSE
28  362 --                  lowSpaceReserve := gSzMemReserve;
28  363 --
28  364 --              Write('Reset max resource usage (Y or N) [N]? ');
28  365 --              Readln(s);
28  366 --              IF s <> '' THEN
28  367 --                  IF (s[1] = 'y') OR (s[1] = 'Y') THEN
28  368 2-                      BEGIN
28  369 --                      gMaxLockedRsrc := 0;
28  370 -2                      END;
28  371 --
28  372 --              Writeln;
28  373 --
28  374 --              SetReserveSize(codeReserve, lowSpaceReserve);
28  375 -1              END
28  376 --          ELSE
28  377 --              BuildAllReserves;
28  378 --
28  379 --          codeReserve := gSzCodeReserve;
28  380 --          codeShort := gReserveShortfall;
28  381 --          lowSpaceReserve := gSzMemReserve;
28  382 --          gotCode := gOKCodeReserve;
28  383 --          gotLowSpace := gMemReserve^ <> NIL;
28  384 -0 A    END { DoChangeReserve };
28  385 --      {$ENDC qDebug}
28  386 --
28  387 --
28  388 --      {$IFC qTrace}{$D+}{$ENDC}
28  389 --      {$S MAInit}
28  390 --      {-----------------------------------+
28  391 --      |   DoInitUMemory                   |
28  392 --      +-----------------------------------}
28  393 -- A    PROCEDURE DoInitUMemory (VAR sizeTempReserve, sizeLowSpaceReserve: Size);
28  394 --
28  395 --          { Called from InitUMemory so that InitUMemory can be in the main segment
```

```
28  396 --              and this code can be in another segment. }
28  397 --
28  398 --      TYPE
28  399 --          pINTEGER =  ^INTEGER;
28  400 --          hMem =          ^pMem;
28  401 --          pMem =          ^Mem;
28  402 --          Mem =           RECORD
28  403 --                          codeVal,
28  404 --                          lowSpaceVal,
28  405 --                          stackVal: LONGINT;
28  406 --                          END;
28  407 --
28  408 --      VAR
28  409 --          i:              INTEGER;
28  410 --          oldResLoad:     BOOLEAN;
28  411 --          seg:            Handle;
28  412 --          aZone:          THz;
28  413 --          StackTot:       LONGINT;
28  414 --          h:              Handle;
28  415 --
28  416 0- A    BEGIN
28  417 --          { Initialize memory management globals }
28  418 --          gPermAllocation := FALSE;
28  419 --          gMemReserve := NewHandle(0);
28  420 --          gSzMemReserve := 0;
28  421 --          gCodeReserve := NewHandle(0);
28  422 --          gSzCodeReserve := 0;
28  423 --          gGZPurgeNotify := NIL;
28  424 --          gOKCodeReserve := TRUE;
28  425 --          gReserveExists := FALSE;
28  426 --      {$IFC qDebug}
28  427 --          gSegReport := FALSE;
28  428 --          gUnloadAllSegs := TRUE;
28  429 --      {$ENDC}
28  430 --
28  431 --          aZone := ApplicZone;
28  432 --          aZone^.flags := BOR(aZone^.flags, $0400);
28  433 --              { set the Memory Manager bit that says to always call the
28  434 --                  Grow Zone proc, even in "non-critical" situations }
28  435 --
28  436 --          SetGrowZone(@GrowZoneProc);
28  437 --
28  438 --          { Create a list of code segment handles.  This assumes that (1) code
28  439 --              segments have consecutive ids starting with 1  }
28  440 --
28  441 --          i := CountResources('CODE') - 1;    { Can't use Count1Resources since this
28  442 --                                                  must work on 64K ROMs. }
28  443 --
28  444 --          gCodeSegs := HHandleList(NewHandle(i * SIZEOF(Handle)));
28  445 --
28  446 --          gNoOfSegments := 0;
28  447 --          oldResLoad := GetResLoad;
28  448 --          SetResLoad(FALSE);
28  449 --          WHILE i > 0 DO
28  450 1-              BEGIN
28  451 --              seg := GetResource('CODE', i);
28  452 --              gCodeSegs^^[i] := seg;
28  453 --              IF seg <> NIL THEN
28  454 --                  gNoOfSegments := gNoOfSegments + 1;
28  455 --              i := i -1;
28  456 -1              END;
28  457 --          SetResLoad(oldResLoad);
28  458 --
28  459 --          SetHandleSize(Handle(gCodeSegs), SIZEOF(Handle) * gNoOfSegments);
28  460 --
28  461 --          gNonResidents := gCodeSegs;
28  462 --          IF HandToHand(Handle(gNonResidents)) <> noErr THEN ; { ??? }
28  463 --
28  464 --          gSegNeedsUnloading := HBoolArray(NewHandle(SIZEOF(BOOLEAN) * gNoOfSegments));
28  465 --          FOR i := 1 TO gNoOfSegments DO
28  466 --              gSegNeedsUnloading^^[i] := TRUE;
28  467 --
28  468 --          gNonResidents^^[1] := NIL; { Main is always non resident }
28  469 --          gSegNeedsUnloading^^[1] := FALSE;
28  470 --
28  471 --      { init the gSysMemList }
28  472 --          gSysMemList := HHandleList(NewHandle(0));
28  473 --          AddAllRsrc('LDEF', gSysMemList);
28  474 --          AddAllRsrc('CDEF', gSysMemList);
28  475 --          AddAllRsrc('MDEF', gSysMemList);
28  476 --          AddAllRsrc('WDEF', gSysMemList);
28  477 --          AddAllRsrc('PACK', gSysMemList);
28  478 --
28  479 --      { Compute memory slop needed }
28  480 --          sizeTempReserve := 0;
28  481 --          sizeLowSpaceReserve := 0;
28  482 --          stackTot := 0;
28  483 --
28  484 --          FOR i := 1 TO CountResources('seg!') DO
28  485 1-              BEGIN
28  486 --              h := GetIndResource('seg!', i);
28  487 --              sizeTempReserve := sizeTempReserve + AddSegSizes(h);
28  488 --              ReleaseResource(h);
28  489 -1              END;
28  490 --
28  491 --          FOR i := 1 TO CountResources('mem!') DO
28  492 1-              BEGIN
28  493 --              h := GetIndResource('mem!', i);
28  494 --              WITH hMem(h)^^ DO
```

```
28  495 2-                  BEGIN
28  496 --                  sizeTempReserve := sizeTempReserve + codeVal;
28  497 --                  sizeLowSpaceReserve := sizeLowSpaceReserve + lowSpaceVal;
28  498 --                  stackTot := stackTot + stackVal;
28  499 -2                  END;
28  500 --              ReleaseResource(h);
28  501 -1              END;
28  502 --
28  503 --          SetStackSpace(stackTot);
28  504 --
28  505 --          MaxApplZone;
28  506 --
28  507 --          gApp1MemList := NIL;
28  508 --          gApp2MemList := NIL;
28  509 --
28  510 --          { Set up the LoadSeg patch }
28  511 --
28  512 --      {$IFC (NOT qNeedsROM128K) OR qDebug}
28  513 --      {$IFC NOT qDebug}       { If debugging, patch the trap anyway}
28  514 --          IF NOT gConfiguration.hasROM128K THEN
28  515 --      {$ENDC}
28  516 --              FailOSErr(PatchTrap(gSegLoadPatch, $A9F0, @ALoadMacAppSeg));    { LoadSeg }
28  517 --      {$ENDC}
28  518 --
28  519 -0 A    END { DoInitUMemory };
28  520 --      {$IFC qTrace}{$D++}{$ENDC}
28  521 --
28  522 --
28  523 --      {$S MAMain}
28  524 --      {-------------------------------------+
28  525 --      |   FailNoReserve                     |
28  526 --      +-------------------------------------}
28  527 -- A    PROCEDURE FailNoReserve;
28  528 0- A   BEGIN
28  529 --          IF NOT CheckReserve THEN
28  530 --              Failure(memFullErr, 0);
28  531 -0 A    END { FailNoReserve };
28  532 --
28  533 --
28  534 --      {$S MAMain}
28  535 --      {-------------------------------------+
28  536 --      |   FailSpaceIsLow                    |
28  537 --      +-------------------------------------}
28  538 -- A    PROCEDURE FailSpaceIsLow;
28  539 0- A   BEGIN
28  540 --          IF MemSpaceIsLow THEN
28  541 --              Failure(memFullErr, 0);
28  542 -0 A    END { FailSpaceIsLow };
28  543 --
28  544 --
28  545 --      {$IFC qTrace}{$D+}{$ENDC}
28  546 --      {$S MAMain}
28  547 --      {-------------------------------------+
28  548 --      |   GetReserveSize                    |
28  549 --      +-------------------------------------}
28  550 -- A    PROCEDURE GetReserveSize (VAR szCodeReserve, szMemReserve: SIZE);
28  551 --
28  552 0- A   BEGIN
28  553 --          szCodeReserve := gSzCodeReserve;
28  554 --          szMemReserve := gSzMemReserve;
28  555 -0 A    END { GetMemReserveSize };
28  556 --      {$IFC qTrace}{$D++}{$ENDC}
28  557 --
28  558 --
28  559 --      {$IFC qTrace}{$D+}{$ENDC} { no %_BP/%_EP allowed in here, because we cannot call to any
28  560 --                                  other segment from this procedure }
28  561 --      {$S Main} { must be in Main segment because we call this in order to make the resident segment resident }
28  562 --      {-------------------------------------+
28  563 --      |   GetSegNumber                      |
28  564 --      +-------------------------------------}
28  565 -- A    FUNCTION  GetSegNumber(aProc: ProcPtr): INTEGER;
28  566 --      TYPE
28  567 --          PInteger = ^INTEGER;
28  568 --
28  569 --      VAR
28  570 --          i:          INTEGER;
28  571 --          jt:         LONGINT;
28  572 --          segNum:     INTEGER;
28  573 --          seg:        Handle;
28  574 --          segStart:   LONGINT;
28  575 --
28  576 0- A   BEGIN
28  577 --          IF PInteger(aProc)^ = $4EF9 THEN { loaded  segment }
28  578 --              GetSegNumber := PInteger(ORD(aProc)-2)^
28  579 --          ELSE IF PInteger(aProc)^ = $3F3C THEN { unloaded segment }
28  580 --              GetSegNumber := PInteger(ORD(aProc)+2)^
28  581 --          ELSE { routine that computed @proc was in same segment as the proc }
28  582 1-              BEGIN
28  583 --              {$IFC qDebug}
28  584 --              ProgramBreak('GetSegNumber was not passed an jump table address');
28  585 --              {$ENDC}
28  586 --              GetSegNumber := 0;
28  587 -1              END;
28  588 -0 A    END { GetSegNumber };
28  589 --      {$IFC qTrace}{$D++}{$ENDC}
28  590 --
28  591 --
28  592 --      {$S MAMain}
28  593 --      {$IFC qTrace}{$D+}{$ENDC}
```

```
28  594 --      {-----------------------------------+
28  595 --      |   GetSegSize                      |
28  596 --      +-----------------------------------}
28  597 -- A    FUNCTION  GetSegSize(segnum: INTEGER): Size;
28  598 --
28  599 --      VAR
28  600 --          curResLoad:     BOOLEAN;
28  601 0- A    BEGIN
28  602 --          curResLoad := GetResLoad;
28  603 --          SetResLoad(FALSE);
28  604 --          GetSegSize := SizeResource(GetResource('CODE', segnum));
28  605 --          SetResLoad(curResLoad);
28  606 -0 A    END { GetSegSize };
28  607 --      {$IFC qTrace}{$D++}{$ENDC}
28  608 --
28  609 --
28  610 --      {$S Main}
28  611 --      {$IFC qTrace}{$D+}{$ENDC}
28  612 --      {-----------------------------------+
28  613 --      |   GrowZoneProc                    |
28  614 --      +-----------------------------------}
28  615 -- A    FUNCTION  GrowZoneProc(needed: Size): LONGINT;
28  616 --      VAR
28  617 --          result:         LONGINT;
28  618 --          canPurge:       Handle;
28  619 --          codeSize:       Size;
28  620 --          reserveSize:    LONGINT;
28  621 --
28  622 0- A    BEGIN
28  623 --          SetupA5;
28  624 --
28  625 --          result := 0; { default is to fail }
28  626 --
28  627 --              { on a temp alloc, free all code slack immediately }
28  628 --          IF NOT gPermAllocation THEN
28  629 --              IF HandleIsEligible(gCodeReserve) THEN
28  630 1-                  BEGIN
28  631 --                  EmptyHandle(gCodeReserve);
28  632 --                  gReserveExists := FALSE;
28  633 --                  result := 1;
28  634 -1                  END;
28  635 --
28  636 --          IF result = 0 THEN { try harder: see if we can purge a code segment
28  637 --                               or reduce the code reserve handle }
28  638 1-              BEGIN
28  639 --                  { compute size of resources currently in memory }
28  640 --
28  641 --              codeSize := TotalTempSize(FALSE, canPurge);
28  642 --
28  643 --                  { see if the code reserve handle is too large }
28  644 --
28  645 --              IF HandleIsEligible(gCodeReserve) THEN
28  646 --              { we have a code reserve handle; this implies permanent allocation,
28  647 --                  otherwise the handle would have been emptied above }
28  648 2-                  BEGIN
28  649 --                  reserveSize := GetHandleSize(gCodeReserve);
28  650 --
28  651 --                      { the following test is an optimization to avoid calling
28  652 --                          BuildCodeReserve if there is no hope of reducing
28  653 --                          the code reserve handle }
28  654 --                  IF codeSize + reserveSize + 8 > gSzCodeReserve THEN
28  655 3-                      BEGIN                           { reserve is too big }
28  656 --                      gReserveExists := FALSE;
28  657 --                      { this should lower the code reserve }
28  658 --                      BuildCodeReserve(reserveSize, TRUE);
28  659 --
28  660 --                          { see if we succeeded in freeing some memory }
28  661 --                      IF gCodeReserve^ = NIL THEN
28  662 --                          result := 1
28  663 --                      ELSE IF GetHandleSize(gCodeReserve) < reserveSize THEN
28  664 --                          result := 1;
28  665 -3                      END;
28  666 -2                  END;
28  667 --
28  668 --              IF result = 0 THEN
28  669 --                  IF canPurge <> NIL THEN
28  670 --                          { got something; only purge it if this is temporary
28  671 --                              OR we know there is too much code in memory already }
28  672 --                      IF NOT gPermAllocation OR (gCodeReserve^ = NIL) THEN
28  673 2-                          BEGIN
28  674 --                          IF gGZPurgeNotify <> NIL THEN
28  675 --                              CallNotify(canPurge, gGZPurgeNotify);
28  676 --
28  677 --                          reserveSize := GetHandleSize(canPurge);
28  678 --                          HPurge(canPurge);
28  679 --                          EmptyHandle(canPurge);
28  680 --                          gReserveExists := FALSE;
28  681 --
28  682 --                          IF gPermAllocation THEN { don't free too much however }
28  683 --                              BuildCodeReserve(reserveSize, TRUE);
28  684 --
28  685 --                          result := 1;
28  686 -2                          END;
28  687 -1              END;
28  688 --
28  689 --          IF result = 0 THEN { last ditch attempt: free emergency reserve }
28  690 --              IF HandleIsEligible(gMemReserve) THEN
28  691 1-                  BEGIN
28  692 --                  EmptyHandle(gMemReserve);
```

```
28  693 --                  result := 1;
28  694 -1                  END;
28  695 --
28  696 --          GrowZoneProc := result;
28  697 --
28  698 --          RestoreA5;
28  699 -0 A    END { GrowZoneProc };
28  700 --      {$IFC qTrace}{$D++}{$ENDC}
28  701 --
28  702 --
28  703 --      {$S Main}
28  704 --      {$IFC qTrace}{$D+}{$ENDC}
28  705 --      {-----------------------------------+
28  706 --      |   HandleIsEligible                |
28  707 --      +-----------------------------------}
28  708 -- A    FUNCTION  HandleIsEligible(h: Handle): BOOLEAN;
28  709 0- A    BEGIN
28  710 --          IF h^ = NIL THEN
28  711 --              HandleIsEligible := FALSE
28  712 --          ELSE
28  713 --              HandleIsEligible := (h<>GZMoveHand) AND (h<>GZRootHand);
28  714 -0 A    END { HandleIsEligible };
28  715 --      {$IFC qTrace}{$D++}{$ENDC}
28  716 --
28  717 --
28  718 --      {$IFC qTrace}{$D+}{$ENDC}
28  719 --      {$S Main}       { Must be in main segment and called from main segment }
28  720 --      {-----------------------------------+
28  721 --      |   InitUMemory                     |
28  722 --      +-----------------------------------}
28  723 -- A    FUNCTION  InitUMemory: BOOLEAN;
28  724 --
28  725 --      VAR
28  726 --          codeRes,
28  727 --          lowSpaceRes:    Size;
28  728 --
28  729 0- A    BEGIN
28  730 --          DoInitUMemory(codeRes, lowSpaceRes);
28  731 --          UnloadAllSegments;  { get init segment(s) out of middle of heap, so
28  732 --                                SetReserveSize has maximum space to work with }
28  733 --
28  734 --          SetReserveSize(codeRes, lowSpaceRes);
28  735 --          InitUMemory := gOKCodeReserve;
28  736 -0 A    END { InitUMemory };
28  737 --      {$IFC qTrace}{$D++}{$ENDC}
28  738 --
28  739 --
28  740 --      {$IFC qTrace}{$D+}{$ENDC} { no %_BP/%_EP allowed in here, because we cannot call to any
28  741 --                                  other segment from this procedure }
28  742 --      {$S Main} { must be in Main segment }
28  743 --      {-----------------------------------+
28  744 --      |   LoadMacAppSegment               |
28  745 --      +-----------------------------------}
28  746 -- A    FUNCTION  LoadMacAppSegment(segNum: INTEGER): LONGINT;
28  747 --
28  748 --          { This function patches LoadSeg.  It is called from the assembly-language
28  749 --              routine AMacAppLoadSeg, which is the actual patch setup by PatchTrap.
28  750 --              AMacAppLoadSeg saves D0-D1-A0/A1 and sets up the stack by
28  751 --              1) allocating space for LoadMacAppSegment's result, and
28  752 --              2) pushing a copy of LoadSeg's parameter onto the stack for use
28  753 --              by LoadMacAppSegment. }
28  754 --
28  755 --      {$IFC qDebug}
28  756 --      VAR
28  757 --          id:             INTEGER;
28  758 --          kind:           ResType;
28  759 --          segName:        Str255;
28  760 --      {$ENDC}
28  761 --
28  762 0- A    BEGIN
28  763 --          SetupA5;        { ***** Called from trap patches *****}
28  764 --
28  765 --          LoadMacAppSegment := gSegLoadPatch.oldTrapAddr;
28  766 --
28  767 --          IF PreloadSegment(segNum) THEN ; { ??? what to do if fails ??? }
28  768 --
28  769 --          gSegNeedsUnloading^^[segNum] := TRUE;
28  770 --
28  771 --      {$IFC qDebug}
28  772 --          IF gSegReport THEN
28  773 1-              BEGIN
28  774 --              { Cause the debugger to break at the start of the next routine. }
28  775 --              gReportNext := TRUE;
28  776 --              GetResInfo(gCodeSegs^^[segNum], id, kind, segName);
28  777 --              NumToString(segNum, gReportInfo);
28  778 --              gReportInfo := CONCAT('  *** Segment Loaded:  ', gReportInfo, ' ', segName);
28  779 --              gSingleStep := gMemMgtBreak;
28  780 -1              END;
28  781 --      {$ENDC}
28  782 --
28  783 --          RestoreA5;
28  784 --
28  785 -0 A    END { LoadMacAppSegment };
28  786 --      {$IFC qTrace}{$D++}{$ENDC}
28  787 --
28  788 --
28  789 --      {$S Main}       { Must be in Main segment }
28  790 --      {-----------------------------------+
28  791 --      |   LoadResidentSegments            |
```

```
28  792 --      +----------------------------------}
28  793 -- A    PROCEDURE LoadResidentSegments;
28  794 --
28  795 --      VAR
28  796 --          resIndex:       INTEGER;
28  797 --          i:              INTEGER;
28  798 --          offset:         INTEGER;
28  799 --          nameList:       Handle;
28  800 --          segNumber:      INTEGER;
28  801 --          p:              PByte;
28  802 --          name:           Str255;
28  803 --          seg:            Handle;
28  804 --          theType:        ResType;
28  805 --
28  806 0- A    BEGIN
28  807 --          FOR resIndex := 1 TO CountResources('res!') DO
28  808 1-              BEGIN
28  809 --              nameList := GetIndResource('res!', resIndex);
28  810 --              HNoPurge(nameList);
28  811 --
28  812 --              offset := 2;
28  813 --              FOR i := 1 TO PInteger(nameList^)^ DO
28  814 2-                  BEGIN
28  815 --                  p := PByte(ORD4(nameList^) + offset);
28  816 --                  BlockMove(Ptr(p), @name, p^+1);
28  817 --                  offset := offset + LENGTH(name) + 1;
28  818 --
28  819 --                  seg := GetNamedResource('CODE', name);
28  820 --                  IF seg <> NIL THEN
28  821 3-                      BEGIN
28  822 --                      GetResInfo(seg, segNumber, theType, name);
28  823 --                      SetResidentSegment(segNumber, TRUE);
28  824 -3                      END;
28  825 -2                  END;
28  826 --
28  827 --              HPurge(nameList);
28  828 --              ReleaseResource(nameList);
28  829 -1              END;
28  830 -0 A    END {LoadResidentSegments};
28  831 --
28  832 --
28  833 --      {$S MAMain}
28  834 --      {----------------------------------+
28  835 --      |   MemSpaceIsLow                  |
28  836 --      +----------------------------------}
28  837 -- A    FUNCTION  MemSpaceIsLow: BOOLEAN;
28  838 0- A    BEGIN
28  839 --          BuildAllReserves;
28  840 --
28  841 --          MemSpaceIsLow := gMemReserve^ = NIL;
28  842 -0 A    END { MemSpaceIsLow };
28  843 --
28  844 --
28  845 --      {$IFC qTrace}{$D+}{$ENDC}
28  846 --      {$S MAMain}
28  847 --      {----------------------------------+
28  848 --      |   NewPermHandle                  |
28  849 --      +----------------------------------}
28  850 -- A    FUNCTION  NewPermHandle(logicalSize: Size): Handle;
28  851 --      VAR
28  852 --          priorPerm:  BOOLEAN;
28  853 --
28  854 0- A    BEGIN
28  855 --          priorPerm := PermAllocation(TRUE);
28  856 --          NewPermHandle := NewHandle(logicalSize);
28  857 --          priorPerm := PermAllocation(priorPerm);
28  858 -0 A    END { NewPermHandle };
28  859 --      {$IFC qTrace}{$D++}{$ENDC}
28  860 --
28  861 --
28  862 --      {$IFC qTrace}{$D+}{$ENDC}
28  863 --      {$S MAMain}
28  864 --      {----------------------------------+
28  865 --      |   PermAllocation                 |
28  866 --      +----------------------------------}
28  867 -- A    FUNCTION  PermAllocation(permanent: BOOLEAN): BOOLEAN;
28  868 --      VAR
28  869 --          b:  BOOLEAN;
28  870 --
28  871 0- A    BEGIN
28  872 --          PermAllocation := gPermAllocation;
28  873 --
28  874 --          IF permanent <> gPermAllocation THEN
28  875 1-              BEGIN
28  876 --              gPermAllocation := permanent;
28  877 --
28  878 --              IF permanent THEN
28  879 --                  BuildCodeReserve(kGZMaxAlloc, FALSE);
28  880 -1              END;
28  881 -0 A    END { PermAllocation };
28  882 --      {$IFC qTrace}{$D++}{$ENDC}
28  883 --
28  884 --
28  885 --      {$IFC qTrace}{$D+}{$ENDC} { no %_BP/%_EP allowed in here, because we cannot call to any
28  886 --                                  other segment from this procedure }
28  887 --      {$S Main} { must be in Main segment }
28  888 --      {----------------------------------+
28  889 --      |   PreloadSegment                 |
28  890 --      +----------------------------------}
```

```
28  891 -- A   FUNCTION  PreloadSegment(segNum: INTEGER): BOOLEAN;
28  892 --     TYPE
28  893 --         HSegHeader = ^PSegHeader;
28  894 --         PSegHeader = ^SegHeader;
28  895 --         SegHeader = RECORD
28  896 --             jtOffset:   INTEGER;
28  897 --             jtCount:    INTEGER;
28  898 --             END;
28  899 --         PInteger = ^INTEGER;
28  900 --
28  901 --     VAR
28  902 --         seg:    Handle;
28  903 --         err:    OSErr;
28  904 0- A   BEGIN
28  905 --     {$IFC qDebug}
28  906 --         IF gPermAllocation THEN
28  907 1-             BEGIN
28  908 --             Writeln('segment # = ', segNum:1);
28  909 --             ProgramBreak('Trying to load a segment with PermAllocation = TRUE.');
28  910 -1             END;
28  911 --     {$ENDC}
28  912 --
28  913 --         seg := GetResource('CODE', segNum);
28  914 --
28  915 --         IF seg = NIL THEN
28  916 --             PreloadSegment := FALSE
28  917 --         ELSE
28  918 1-             BEGIN
28  919 --             PreloadSegment := TRUE;
28  920 --
28  921 --             IF NOT BTst(GetHandleBits(seg), 7) THEN { not yet locked }
28  922 2-                 BEGIN
28  923 --                 MoveHHi(seg); { ??? check MemErr ??? }
28  924 --                 HLock(seg);
28  925 -2                 END;
28  926 -1             END;
28  927 -0 A   END { PreloadSegment };
28  928 --     {$IFC qTrace}{$D++}{$ENDC}
28  929 --
28  930 --
28  931 --     {$S MAMain}
28  932 --     {-----------------------------------+
28  933 --     |   RemHandle                       |
28  934 --     +-----------------------------------}
28  935 -- A   PROCEDURE RemHandle(h: Handle; toList: HHandleList);
28  936 --     VAR
28  937 --         offset: LONGINT;
28  938 0- A   BEGIN
28  939 --         offset := Munger(Handle(toList), 0, @h, SIZEOF(Handle), @h, 0);
28  940 -0 A   END { RemHandle };
28  941 --
28  942 --
28  943 --     {$S MAMain}
28  944 --     {$IFC qTrace}{$D+}{$ENDC}
28  945 --     {-----------------------------------+
28  946 --     |   ScanHandles                     |
28  947 --     +-----------------------------------}
28  948 -- A   PROCEDURE ScanHandles(PROCEDURE DoToHandle(h: Handle));
28  949 --
28  950 --         { Calls DoToHandle for each handle in gCodeSegs, gSysMemList,
28  951 --           gAppl1MemList and gApp2MemList. }
28  952 --
28  953 --         {-------------------------------+
28  954 --         |   ScanList                    |
28  955 --         +-------------------------------}
28  956 -- B       PROCEDURE ScanList(list: HHandleList);
28  957 --         TYPE
28  958 --             PtrHandle   = ^Handle;
28  959 --
28  960 --         VAR
28  961 --             i:      INTEGER;
28  962 --             p:      PtrHandle;
28  963 0- B       BEGIN
28  964 --             i := GetHandleSize(Handle(list)) DIV SIZEOF(Handle);
28  965 --
28  966 --             p := PtrHandle(list^);
28  967 --             WHILE i > 0 DO
28  968 1-                 BEGIN
28  969 --                 DoToHandle(p^);
28  970 --                 p := PtrHandle(Ord(p) + SIZEOF(Handle));
28  971 --                 i := i - 1;
28  972 -1                 END;
28  973 -0 B       END { ScanList };
28  974 --
28  975 0- A   BEGIN
28  976 --         ScanList(gCodeSegs);
28  977 --         IF gAppl1MemList <> NIL THEN
28  978 --             ScanList(gAppl1MemList);
28  979 --         ScanList(gSysMemList);
28  980 --         IF gApp2MemList <> NIL THEN
28  981 --             ScanList(gApp2MemList);
28  982 -0 A   END { ScanHandles };
28  983 --     {$IFC qTrace}{$D++}{$ENDC}
28  984 --
28  985 --
28  986 --     {$IFC qTrace}{$D+}{$ENDC}
28  987 --     {$S MAMain}
28  988 --     {-----------------------------------+
28  989 --     |   SetReserveSize                  |
```

```
28  990 --      +-----------------------------------}
28  991 -- A    PROCEDURE SetReserveSize(forCode, forOther: Size);
28  992 --      VAR
28  993 --          oldPerm:    BOOLEAN;
28  994 0- A    BEGIN
28  995 --          gSzCodeReserve := forCode;
28  996 --          gSzMemReserve := forOther;
28  997 --
28  998 --          { Since the numbers have changed, make sure we start from scratch. }
28  999 --          gReserveExists := FALSE;
28 1000 --          EmptyHandle(gMemReserve);
28 1001 --
28 1002 --          BuildAllReserves;
28 1003 -0 A    END { SetReserveSize };
28 1004 --      {$IFC qTrace}{$D++}{$ENDC}
28 1005 --
28 1006 --
28 1007 --      {$IFC qTrace}{$D+}{$ENDC} { no %_BP/%_EP allowed in here, because we cannot call to any
28 1008 --                                  other segment from this procedure }
28 1009 --      {$S Main} { must be in Main segment }
28 1010 --      {-----------------------------------+
28 1011 --      |   SetResidentSegment              |
28 1012 --      +-----------------------------------}
28 1013 -- A    PROCEDURE SetResidentSegment(segNum: INTEGER; makeResident: BOOLEAN);
28 1014 0- A    BEGIN
28 1015 --          IF segNum > 1 THEN { can't modify the blank segment (# 1) }
28 1016 --              IF makeResident THEN
28 1017 1-                  BEGIN
28 1018 --                  gNonResidents^^[segNum] := NIL;
28 1019 --                  gSegNeedsUnloading^^[segNum] := FALSE;
28 1020 --                  IF PreloadSegment(segnum) THEN ; { ??? what if this fails ??? }
28 1021 -1                  END
28 1022 --              ELSE
28 1023 1-                  BEGIN
28 1024 --                  gNonResidents^^[segNum] := gCodeSegs^^[segNum];
28 1025 --                  gSegNeedsUnloading^^[segNum] := TRUE;
28 1026 -1                  END;
28 1027 -0 A    END { SetResidentSegment };
28 1028 --      {$IFC qTrace}{$D++}{$ENDC}
28 1029 --
28 1030 --
28 1031 --      {$S MAInit}
28 1032 --      {$IFC qTrace}{$D+}{$ENDC}
28 1033 --      {-----------------------------------+
28 1034 --      |   SetStackSpace                   |
28 1035 --      +-----------------------------------}
28 1036 -- A    PROCEDURE SetStackSpace(numBytes: LONGINT);
28 1037 --
28 1038 --      CONST
28 1039 --          CurStackBase    =   $0908;
28 1040 --
28 1041 --      VAR
28 1042 --          curLimit:   LONGINT;
28 1043 --          newLimit:   LONGINT;
28 1044 --
28 1045 0- A    BEGIN
28 1046 --          newLimit := PLongInt(CurStackBase)^ - numBytes;
28 1047 --
28 1048 --          IF Ord(GetApplLimit) > newLimit THEN
28 1049 --              SetApplLimit(Ptr(newLimit));
28 1050 -0 A    END { SetStackSpace };
28 1051 --      {$IFC qTrace}{$D++}{$ENDC}
28 1052 --
28 1053 --
28 1054 --      {$S MAMain}
28 1055 --      {$IFC qTrace}{$D+}{$ENDC}
28 1056 --      {-----------------------------------+
28 1057 --      |   TotalTempSize                   |
28 1058 --      +-----------------------------------}
28 1059 -- A    FUNCTION  TotalTempSize(justLocked: BOOLEAN; VAR canPurge: Handle): Size;
28 1060 --      VAR
28 1061 --          total:      Size;
28 1062 --          applZone:   THz;
28 1063 --
28 1064 --          {-------------------------------+
28 1065 --          |   TotalUp                     |
28 1066 --          +-------------------------------}
28 1067 -- B        PROCEDURE TotalUp(h: Handle);
28 1068 --          VAR
28 1069 --              hIsLocked:  BOOLEAN;
28 1070 0- B        BEGIN
28 1071 --              IF h^ <> NIL THEN { in memory already }
28 1072 --                  IF HandleZone(h) = applZone THEN { in application heap }
28 1073 1-                      BEGIN
28 1074 --                      HNoPurge(h);
28 1075 --
28 1076 --                      hIsLocked := BTst(GetHandleBits(h), 7);
28 1077 --
28 1078 --                      IF NOT justLocked OR hIsLocked THEN
28 1079 --                          total := total + GetHandleSize(h) + 8;
28 1080 --                                  { add in the size plus heap overhead }
28 1081 --
28 1082 --                      IF (NOT hIsLocked) THEN
28 1083 --                          IF canPurge = NIL THEN
28 1084 --                              IF HandleIsEligible(h) THEN
28 1085 --                                  canPurge := h;
28 1086 -1                      END;
28 1087 -0 B        END { TotalUp };
28 1088 --
```

```
28 1089 0- A   BEGIN
28 1090 --         canPurge := NIL;
28 1091 --         total := 0;
28 1092 --         applZone := ApplicZone;
28 1093 --
28 1094 --         ScanHandles(TotalUp);
28 1095 --
28 1096 --         TotalTempSize := total;
28 1097 -0 A   END { TotalTempSize };
28 1098 --     {$IFC qTrace}{$D++}{$ENDC}
28 1099 --
28 1100 --
28 1101 --     {$IFC qTrace}{$D+}{$ENDC} { no %_BP/%_EP allowed in here, because we cannot call to any
28 1102 --                                 other segment from this procedure }
28 1103 --     {$S Main} { must be in Main segment }
28 1104 --     {-----------------------------------+
28 1105 --     |   UnloadAllSegments               |
28 1106 --     +-----------------------------------}
28 1107 -- A   PROCEDURE UnloadAllSegments;
28 1108 --
28 1109 --     CONST
28 1110 --         CurJTOffset =   $0934;
28 1111 --
28 1112 --     VAR
28 1113 --         i:              LONGINT;
28 1114 --         seg:            Handle;
28 1115 --         base:           LONGINT;
28 1116 --         oldResLoad:     BOOLEAN;
28 1117 --
28 1118 0- A   BEGIN
28 1119 --     {$IFC qDebug}
28 1120 --         CheckRsrcUsage;
28 1121 --     {$ENDC}
28 1122 --
28 1123 --     {$IFC qDebug}
28 1124 --         IF gUnloadAllSegs THEN
28 1125 --     {$ENDC}
28 1126 1-             BEGIN
28 1127 --             {$%+}
28 1128 --             base := %_GetA5 + PInteger(CurJTOffset)^;
28 1129 --             {$%-}
28 1130 --
28 1131 --             FOR i := 2 TO gNoOfSegments DO      { 1st segment is never unloaded }
28 1132 --                 IF gSegNeedsUnloading^^[i] THEN
28 1133 2-                     BEGIN
28 1134 --                     seg := gNonResidents^^[i];
28 1135 --                     IF seg <> NIL THEN
28 1136 --                         IF seg^ <> NIL THEN
28 1137 --                             UnloadSeg(Ptr(base + HInteger(seg)^^ + 2));
28 1138 --                     gSegNeedsUnloading^^[i] := FALSE;
28 1139 -2                     END;
28 1140 --
28 1141 --     {$IFC qDebug}
28 1142 --             IF gSegReport THEN
28 1143 --                 ProgramReport('  *** Just unloaded all segments ***', gMemMgtBreak);
28 1144 --     {$ENDC}
28 1145 -1             END;
28 1146 -0 A   END { UnloadAllSegments };
28 1147 --     {$IFC qTrace}{$D++}{$ENDC}
28 1148 --
28 1149 --
28 1150 --     {$IFC qDebug}
28 1151 --     {$S MAInspect}
28 1152 --     {-----------------------------------+
28 1153 --     |   WriteReserves                   |
28 1154 --     +-----------------------------------}
28 1155 -- A   PROCEDURE WriteReserves;
28 1156 --
28 1157 0- A   BEGIN
28 1158 --         WrLblPtr('Temporary reserve (gCodeReserve)', gCodeReserve); Writeln;
28 1159 --         WrLblPtr('Low-memory reserve (gMemReserve)', gMemReserve);  Writeln;
28 1160 -0 A   END { WriteReserves };
28 1161 --     {$ENDC}
27  368 --
27  369 --     END { UMemory }.
```

```
29    1 --    {$P}
29    2 --    { UMenuSetup.p }
29    3 --    { Copyright © 1984-1988 Apple Computer Inc.  All rights reserved. }
29    4 --
29    5 --    { Conditional compile flags used by this unit:
29    6 --
29    7 --        qDebug      True includes debugging code (writelns and ProgramBreak),
29    8 --                    false ignores this code.
29    9 --
29   10 --        qTrace          True surrounds trap patches with $D+ and $D++ (because
29   11 --                        the default is $D++), false skips these directives.
29   12 --
29   13 --        The settings of $D, $R and $N are imported from the UMAUtil interface.
29   14 --     }
29   15 --
29   16 --
29   17 --    UNIT UMenuSetup;
29   18 --
29   19 --    {
29   20 --                        T H E O R Y   O F   O P E R A T I O N
29   21 --
29   22 --        This unit provides two features:  It implements a command numbering
29   23 --        system that is independent of menu/item placement, and implements a
29   24 --        framework for optimizing menu setups.
29   25 --
29   26 --        The command numbering system works by assigning commands a unique
29   27 --        integer number, and providing a mechanism for mapping command numbers
29   28 --        to menu/item pairs.  A set of routines are provide to manipulate
29   29 --        commands via their command number rather than their menu and item
29   30 --        numbers.
29   31 --
29   32 --        Each command can be assigned an integer command number from 1 to
29   33 --        32767.  To associate a command number with a menu item, menu resources
29   34 --        are defined with the 'cmnu' resource type--not the 'MENU' type.  The
29   35 --        'cmnu' type is just like the 'MENU' type, with an additional command
29   36 --        number field for each menu item.  The program PostRez converts the
29   37 --        'cmnu' output of Rez into equivalent 'MENU' resources and one 'mntb'
29   38 --        resource.  The 'mntb' resource maps command numbers to menu/item pairs.
29   39 --
29   40 --        The items of some menus cannot be determined until run-time.  The font
29   41 --        menu is an example.  In such cases, no command number is assigned by
29   42 --        you.  Instead, the command number is equal to -(256 * menu + item).
29   43 --
29   44 --        The procedure CmdToMenuItem converts a command number to a menu id
29   45 --        and item number by the following method:
29   46 --
29   47 --        - If the command number is positive, the command table is searched
29   48 --          for the number.  If it is found, CmdToMenuItem returns the corre-
29   49 --          sponding menu id and item number.  Otherwise the menu and item are
29   50 --          zero.
29   51 --
29   52 --        - If the command number is negative, the menu number is the upper eight
29   53 --          bits, and the item number the lower eight bits, of the absolute value
29   54 --          of the command number.  (This implies that menu id's must be <= 127
29   55 --          and item numbers must be <= 255.)
29   56 --
29   57 --        The function CmdFromMenuItem converts menu/items to command numbers by
29   58 --        the following method:
29   59 --
29   60 --        - If the item number is positive, the command table is search for the
29   61 --          given menu/item pair.  If found, the corresponding command number is
29   62 --          returned.  If not found, the number returned is equal to
29   63 --          -(256 * menu + item).
29   64 --
29   65 --        - If the item number is negative, the number returned is equal to
29   66 --          -item number.
29   67 --
29   68 --        Note that CmdToMenuItem and CmdFromMenuItem work regardless of whether
29   69 --        the command is actually installed in the menu bar.  MacApp takes
29   70 --        advantage of this feature by having a set of generic "buzz" commands,
29   71 --        whose primary use is to set the name of the Undo command (e.g. 'Undo
29   72 --        Drawing').
29   73 --
29   74 --        The second feature of this unit is to optimize menu setups (changing
29   75 --        the appearance of menus and items).  Normally, each call to CheckItem,
29   76 --        SetItemStyle and SetCmdIcon in turn calls CalcMenuSize because the width
29   77 --        or height of the menu may have changed.  If menu setups are done all at
29   78 --        one time, then CalcMenuSize can be deferred until the end of the setup
29   79 --        process.  The procedure PerformMenuSetup implements this mechanism.  It
29   80 --        relies on the fact that menu setups are done all at once, and that you
29   81 --        will call SetCmdName, SetItemStyle and SetCmdIcon instead of SetItem,
29   82 --        SetItemStyle and SetCmdIcon.  It works only for menus whose id is from
29   83 --        1 to mLastMenu.  Menus with IDs greater than mLastMenu are not affected
29   84 --        by this scheme.
29   85 --
29   86 --        PerformMenuSetup accepts one parameter, a procedure which actually
29   87 --        implements the menu appearance changes.  SetupMenus performs the
29   88 --        following steps:
29   89 --
29   90 --        - Before calling your menu setup procedure, StartupMenuSetup is called.
29   91 --          It disables all menus and items, remembers the current
29   92 --          menu proc and enabled flags of each menu, and sets the MenuProc of
29   93 --          each menu to gHNullMenuProc, thereby disabling CalcMenuSize.
29   94 --        - Your procedure is called.  It in turn calls SetCmdName,
29   95 --          SetCmdStyle, SetCmdIcon, Enable, EnableCheck, or a
29   96 --          Toolbox routine (provided it isn't SetItem, SetItemStyle or
29   97 --          SetCmdIcon).
29   98 --        - Menus with at least one enabled item are enabled, CalcMenuSize is
```

```
29   99 --          called for those menus that need it, each menu's menuproc is restored,
29  100 --          and DrawMenuBar is called if the enabled state of any menu changed.
29  101 --      }
29  102 --
29  103 --  INTERFACE
29  104 --
29  105 --  {-----------------------------------+
29  106 --  |   Uses                            |
29  107 --  +-----------------------------------}
29  108 --  USES
29  109 --      {$LOAD MacIntf.LOAD}
29  110 --          MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
29  111 --      {$LOAD}
29  112 --          UMAUtil,
29  113 --          UFailure,
29  114 --          UMemory
29  115 --  {$IFC qTrace}
29  116 --          ,UTrace
29  117 --  {$ENDC}
29  118 --          ;
29  119 --
29  120 --  {-----------------------------------+
29  121 --  |   Global constants                |
29  122 --  +-----------------------------------}
29  123 --  CONST
29  124 --      kIDMNTBbyCmdNumber  =       0;      { menu command number table }
29  125 --
29  126 --      { Predefined menu IDs;
29  127 --          MacApp reads in menus with IDs [mApple..mLastInstalled], until a break
29  128 --              in the sequence, and installs them in the menu bar.
29  129 --          MacApp reads in menus with IDs[mLastInstalled+1..mLastMenu], until a
29  130 --              break in the sequence, but does not install them.
29  131 --
29  132 --      MacApp appends the desk accessories to the mApple menu. }
29  133 --
29  134 --      mApple          =       1;  { MacApp does not disable any commands in
29  135 --                                      this menu }
29  136 --      mFile           =       2;
29  137 --      mEdit           =       3;
29  138 --
29  139 --      mDebug          =     900;  { read in if debugging is turned on;  number
29  140 --                                      corresponds with debugging command
29  141 --                                      numbers }
29  142 --      mLastMenu       =      63;  { MacApp disables menu titles
29  143 --                                      mFile..mLastMenu generically }
29  144 --
29  145 --  {-----------------------------------+
29  146 --  |   Global types                    |
29  147 --  +-----------------------------------}
29  148 --  TYPE
29  149 --
29  150 --      CmdNumber = INTEGER;
29  151 --
29  152 --  {-----------------------------------+
29  153 --  |   Global variables                |
29  154 --  +-----------------------------------}
29  155 --  VAR
29  156 --      gMenusAreSetup:     BOOLEAN;            { set FALSE after every event; set
29  157 --                                              TRUE by SetupTheMenus, which is called at
29  158 --                                              Idle Begin; if your DoIdle changes the
29  159 --                                              target or makes other changes that would
29  160 --                                              alter the appearance of menus, you must set
29  161 --                                              gMenusAreSetup to FALSE there. }
29  162 --      gRedrawMenuBar:     BOOLEAN;            { if TRUE, then DrawMenuBar will be
29  163 --                                              called by TApplication.SetupTheMenus.  If
29  164 --                                              you have menus that are not handled by
29  165 --                                              MacApp, your implementation(s) of
29  166 --                                              DoSetupMenus can set this to TRUE to force
29  167 --                                              the menu bar to be redrawn. }
29  168 --      gHNullMenuProc:     Handle;             { Handle to null menu proc }
29  169 --  {$IFC qTrace}
29  170 --      gTraceSetupMenus:   BOOLEAN;
29  171 --  {$ENDC}
29  172 --
29  173 --  {-----------------------------------+
29  174 --  |   Global routines                 |
29  175 --  +-----------------------------------}
29  176 --  { The following routines all recognize command numbers of the form
29  177 --      -(256 * menu + item) and -(256 * menu) }
29  178 --  { The former is used when there is no command number; the latter to
29  179 --      enable/disable a whole menu (rare) }
29  180 --
29  181 --  FUNCTION  CmdFromMenuItem(menu, item: INTEGER): CmdNumber;
29  182 --      { Returns the command number for the given menu ID and item number.
29  183 --
29  184 --          If the item number is positive, the command table is search for the
29  185 --          given menu/item pair.  If found, the corresponding command number
29  186 --          is returned.  If not found, the number returned is equal to
29  187 --          -(256 * menu + item).
29  188 --
29  189 --          If the item number is negative, the number returned is equal to
29  190 --          -item number.
29  191 --       }
29  192 --
29  193 --  PROCEDURE CmdToMenuItem(aCmd: CmdNumber; VAR menu, item: INTEGER);
29  194 --      { CmdToMenuItem returns the menu ID and item number of the given
29  195 --          command.
29  196 --
29  197 --          If aCmd is positive, the command table is searched for aCmd.
```

```
29  198 --              If it is found, CmdToMenuItem returns the corresponding menu ID and
29  199 --              item number.  Otherwise menu and item are set to zero.
29  200 --
29  201 --              If aCmd is negative, the menu number is the upper eight bits, and
29  202 --              the item number the lower eight bits, of the absolute value of aCmd.
29  203 --              (This implies that menu id's must be <= 127 and item numbers must be
29  204 --              <= 255.)
29  205 --          }
29  206 --
29  207 --      PROCEDURE CmdToName(aCmd: CmdNumber; VAR menuText: Str255);
29  208 --          { Return the text of the menu item for the given command. }
29  209 --
29  210 --      PROCEDURE EachMenuDo (PROCEDURE DoToMenu(aMenuHandle: MenuHandle);
29  211 --                            includeHierarchical: BOOLEAN);
29  212 --          { Sequences thru each menu in menu bar with ID in [0..mLastMenu], calling
29  213 --              DoToMenu for each menu.  Handles mDebug as a special case. }
29  214 --
29  215 --      PROCEDURE Enable(aCmd: CmdNumber; canDo: BOOLEAN);
29  216 --          { Enable or disable a command }
29  217 --
29  218 --      PROCEDURE EnableCheck(aCmd: CmdNumber; canDo: BOOLEAN; checkIt: BOOLEAN);
29  219 --          { Enable/Disable and check/uncheck a command }
29  220 --
29  221 --      FUNCTION  GetResMenu(menuResID: INTEGER): MenuHandle;
29  222 --          { Utility that just does GetResource('MENU', resourceID), so that it
29  223 --              works for menus not currently in the menu bar.  (You cannot call
29  224 --              GetMenu more than once for the same menu, so you can use this
29  225 --              instead.) }
29  226 --
29  227 --      PROCEDURE InitUMenuSetup;
29  228 --          { Initializes this unit, and must be called before using any other part of
29  229 --              this unit.  Loads the command table and sets up the null menu proc. }
29  230 --
29  231 --      PROCEDURE MAInsertMenu (theMenu: MenuHandle; beforeID: INTEGER);
29  232 --          { Equivalent to the Menu Manager's InsertMenu with the addition that the
29  233 --              MAInsertMenu looks for an 'mctb' resource whose id is the menu's id, and
29  234 --              if it is present applies the 'mctb' to the menu.}
29  235 --
29  236 --      PROCEDURE NeedCalcMenuSize(aMenuHandle: MenuHandle);
29  237 --          { This procedure is called as part of the menu setup process and
29  238 --              indicates that the given menu needs CalcMenuSize.  You must
29  239 --              explicitly call this if you use change a menu by means other than
29  240 --              those supplied in this unit (e.g. you insert or delete items from
29  241 --              a menu).  Failure to do so will result in improperly sized menus. }
29  242 --
29  243 --      PROCEDURE SetCmdIcon(aCmd: CmdNumber; menuIcon: Byte);
29  244 --          { You should call this (instead of the Toolbox's SetCmdIcon) for menus
29  245 --              that are in the range [1..mLastMenu].  In addition to changing the
29  246 --              menu this tells MacApp that the menu's size needs to be recomputed. }
29  247 --
29  248 --      PROCEDURE SetCmdName(aCmd: CmdNumber; menuText: Str255);
29  249 --          { You should call this (instead of the Toolbox's SetItem) for menus that
29  250 --              are in the range [1..mLastMenu].  In addition to changing the menu
29  251 --              this tells MacApp that the menu's size needs to be recomputed. }
29  252 --
29  253 --      PROCEDURE SetIndCmdName(aCmd: CmdNumber; rsrcID, strIndex: INTEGER);
29  254 --          { Does a GetIndString with the rsrcID & strIndex; and then a
29  255 --              SetCmdName. }
29  256 --
29  257 --      PROCEDURE SetMenuState(aCmd: CmdNumber;
29  258 --              rsrcID, falseBuzzItem, trueBuzzItem: INTEGER; stateVariable: BOOLEAN);
29  259 --          {   IF stateVariable THEN
29  260 --                  SetIndCmdName(aCmd, rsrcID, trueBuzzItem)
29  261 --              ELSE
29  262 --                  SetIndCmdName(aCmd, rsrcID, falseBuzzItem);
29  263 --          }
29  264 --
29  265 --      PROCEDURE SetStyle(aCmd: CmdNumber; aStyle: Style);
29  266 --          { You should call this (instead of the Toolbox's SetItemStyle) for menus
29  267 --              that are in the range [1..mLastMenu].  In addition to changing the
29  268 --              menu this tells MacApp that the menu's size needs to be recomputed. }
29  269 --
29  270 --      PROCEDURE PerformMenuSetup (PROCEDURE TheMenuSetterUpper);
29  271 --
29  272 --      IMPLEMENTATION
29  273 --
29  274 --      {$I UMenuSetup.incl.p}
```

```
30    1 --      {$P}
30    2 --      { UMenuSetup.incl.p }
30    3 --      { Copyright © 1984-1988 by Apple Computer Inc.  All rights reserved. }
30    4 --
30    5 --      {-----------------------------------+
30    6 --      |   Local types                     |
30    7 --      +-----------------------------------}
30    8 --      TYPE
30    9 --          { the following must match the declaration in the PostRez tool }
30   10 --          MenuCmdRecord =
30   11 --              RECORD
30   12 --                theCmdNumber:         INTEGER;
30   13 --                theMenuNumber:        INTEGER;
30   14 --                theItemNumber:        INTEGER;
30   15 --              END;
30   16 --
30   17 --          CmdArray = ARRAY[1..4000] OF MenuCmdRecord;
30   18 --          PCmdArray = ^CmdArray;
30   19 --          CmdTable = ^PCmdArray;
30   20 --
30   21 --      {-----------------------------------+
30   22 --      |   Local variables                 |
30   23 --      +-----------------------------------}
30   24 --      VAR
30   25 --          gCmdTable:          CmdTable;           { maps CmdNumber <--> (menu,item) }
30   26 --          gSizeCmdTable:      INTEGER;            { number of records in gCmdTable }
30   27 --
30   28 --
30   29 --      {-----------------------------------+
30   30 --      |   Forward declarations            |
30   31 --      +-----------------------------------}
30   32 -- A    PROCEDURE NullMenuProc(message: INTEGER; aMenuHandle: MenuHandle;
30   33 --                             VAR menuRect: Rect; hitPt: Point;
30   34 --                             VAR whichItem: INTEGER); FORWARD;
30   35 --
30   36 --
30   37 --      (*****************************************************************************************)
30   38 --      (*      G l o b a l   R o u t i n e s                                                    *)
30   39 --      (*****************************************************************************************)
30   40 --      {-----------------------------------+
30   41 --      |   GetMenuList                     |
30   42 --      +-----------------------------------}
30   43 -- A    FUNCTION  GetMenuList: Handle; INLINE $2EB8, $0A1C;
30   44 --      {
30   45 --          MOVE.L      $A1C,(SP)
30   46 --       }
30   47 --
30   48 --
30   49 --      {$S MARes} { must be in resident segment because we mark what ever segment
30   50 --                   contains this function resident }
30   51 --      {$IFC qTrace}{$D+}{$ENDC}
30   52 --      {-----------------------------------+
30   53 --      |   CmdFromMenuItem                 |
30   54 --      +-----------------------------------}
30   55 -- A    FUNCTION  CmdFromMenuItem(menu, item: INTEGER): CmdNumber;
30   56 --          { Given a menuID/item # return the appropriate command number.  IF there
30   57 --            is no such command number, return   -BOR(BSL(menu, 8), item).  If the
30   58 --            item number is <0 then assume that it is a negative command number. }
30   59 --
30   60 --      VAR
30   61 --          i:      INTEGER;
30   62 --          p:      PCmdArray;
30   63 0- A    BEGIN
30   64 --          IF item < 0 THEN
30   65 --              CmdFromMenuItem := -item
30   66 --          ELSE
30   67 1-              BEGIN
30   68 --              IF item > 0 THEN
30   69 2-                  BEGIN
30   70 --                  p := gCmdTable^;
30   71 --
30   72 --                  FOR i := 1 TO gSizeCmdTable DO
30   73 --                      WITH p^[i] DO
30   74 --                          IF (menu = theMenuNumber) AND (item = theItemNumber) THEN
30   75 3-                              BEGIN
30   76 --                              CmdFromMenuItem := theCmdNumber;
30   77 --                              Exit(CmdFromMenuItem);
30   78 -3                              END;
30   79 -2                  END;
30   80 --      {$IFC qDebug}
30   81 --              IF (menu > 127) OR (item > 255) THEN
30   82 2-                  BEGIN
30   83 --                  Writeln('menu = ', menu:1, '  item = ', item:1);
30   84 --                  Writeln('Menu/item number is too big for a negative command number!');
30   85 --                  ProgramBreak('Try using a negative item number instead.');
30   86 -2                  END;
30   87 --      {$ENDC}
30   88 --
30   89 --              CmdFromMenuItem := -BOR(BSL(menu, 8), item);
30   90 -1              END;
30   91 -0 A    END;
30   92 --      {$IFC qTrace}{$D++}{$ENDC}
30   93 --
30   94 --
30   95 --      {$S MARes}
30   96 --      {$IFC qTrace}{$D+}{$ENDC}
30   97 --      {-----------------------------------+
30   98 --      |   CmdToMenuItem                   |
```

```
30   99 --      +-----------------------------------}
30  100 -- A   PROCEDURE CmdToMenuItem(aCmd: CmdNumber; VAR menu, item: INTEGER);
30  101 --      VAR
30  102 --          i:      INTEGER;
30  103 --          p:      PCmdArray;
30  104 --          low:    INTEGER;
30  105 --          high:   INTEGER;
30  106 --          test:   INTEGER;
30  107 0- A    BEGIN
30  108 --          IF aCmd < 0 THEN
30  109 1-              BEGIN
30  110 --              menu := BSR(-aCmd, 8);
30  111 --              item := BAND(-aCmd, 255);
30  112 -1              END
30  113 --          ELSE
30  114 1-              BEGIN
30  115 --              low := 1;
30  116 --              high := gSizeCmdTable;
30  117 --                  { Loop Invariant: unsearched range is [low..high] inclusive }
30  118 --
30  119 --              p := gCmdTable^;
30  120 --              WHILE low <= high DO
30  121 2-                  BEGIN
30  122 --                  test := BSR(low + high, 1);     { (low + high) DIV 2 }
30  123 --                  WITH p^[test] DO
30  124 --                      IF aCmd = theCmdNumber THEN
30  125 3-                          BEGIN
30  126 --                          menu := theMenuNumber;
30  127 --                          item := theItemNumber;
30  128 --                          Exit(CmdToMenuItem);
30  129 -3                          END
30  130 --                      ELSE IF aCmd < theCmdNumber THEN
30  131 --                          high := test - 1
30  132 --                      ELSE
30  133 --                          low := test + 1;
30  134 -2                  END;
30  135 --
30  136 --              { not found }
30  137 --              menu := 0;
30  138 --              item := 0;
30  139 -1              END;
30  140 -0 A    END;
30  141 --      {$IFC qTrace}{$D++}{$ENDC}
30  142 --
30  143 --
30  144 --      {$S MARes}
30  145 --      {-----------------------------------+
30  146 --      |   CmdToName                       |
30  147 --      +-----------------------------------}
30  148 -- A   PROCEDURE CmdToName(aCmd: CmdNumber; VAR menuText: Str255);
30  149 --      VAR
30  150 --          aMenu:      INTEGER;
30  151 --          anItem:     INTEGER;
30  152 --          mHandle:    MenuHandle;
30  153 0- A    BEGIN
30  154 --          menuText := '';
30  155 --
30  156 --          CmdToMenuItem(aCmd, aMenu, anItem);
30  157 --          mHandle := GetResMenu(aMenu);
30  158 --          IF mHandle <> NIL THEN
30  159 --              GetItem(mHandle, anItem, menuText);
30  160 -0 A    END;
30  161 --
30  162 --
30  163 --      {-----------------------------------+
30  164 --      |   EachMenuDo                      |
30  165 --      +-----------------------------------}
30  166 -- A   PROCEDURE EachMenuDo (PROCEDURE DoToMenu(aMenuHandle: MenuHandle);
30  167 --                           includeHierarchical: BOOLEAN);
30  168 --          { Calls DoToMenu for each menu in the menu bar with ID, including the
30  169 --              hierarchical and popup menus if includeHierarchical is TRUE. }
30  170 --
30  171 --      TYPE
30  172 --              { See Inside Mac Volume 5, Menu Manager, 'Menu Mgr Data Structures' }
30  173 --
30  174 --              MLRecord = RECORD
30  175 --                              mHandle:    MenuHandle;
30  176 --                              rightCD:    INTEGER;
30  177 --                              END;
30  178 --
30  179 --              HMenuList = ^PMenuList;
30  180 --              PMenuList = ^AMenuList;
30  181 --              AMenuList = RECORD
30  182 --                              sizeX6:     INTEGER;    { # of menus * 6 }
30  183 --                              rightCd:    INTEGER;
30  184 --                              notUsed:    INTEGER;
30  185 --                              { Variable number of menus }
30  186 --                              menus:      ARRAY[1..1000] OF MLRecord;
30  187 --                              { Variable number of hierarchical menus follows }
30  188 --                          END;
30  189 --      VAR
30  190 --          i:              INTEGER;
30  191 --          ml:             HMenuList;
30  192 --          noOfMenus:      INTEGER;
30  193 --          menu:           MenuHandle;
30  194 --          noOfHierMenus:  INTEGER;
30  195 --          hMenuPtr:       LongInt;
30  196 --          lastHierMenu:   INTEGER;
30  197 --          menurec:        MLRecord;
```

```
30  198 0- A   BEGIN
30  199 --         ml := HMenuList(GetMenuList);
30  200 --         noOfMenus := ml^^.sizeX6 DIV 6;
30  201 --
30  202 --         FOR i := 1 TO noOfMenus DO
30  203 1-             BEGIN
30  204 --             menu := ml^^.menus[i].mHandle;
30  205 --     {$IFC qDebug}
30  206 --             IF (menu^^.menuID = mDebug) THEN
30  207 2-                 BEGIN
30  208 --                 menu^^.menuID := 0;      { store info about debug menu in index 0 }
30  209 --                 DoToMenu(menu);
30  210 --                 menu^^.menuID := mDebug;
30  211 -2                 END
30  212 --             ELSE
30  213 --     {$ENDC}
30  214 --             DoToMenu(menu);
30  215 -1             END;
30  216 --
30  217 --         { Now do hierarchial menus}
30  218 --         IF gConfiguration.hasHierarchicalMenus THEN
30  219 1-             BEGIN
30  220 --             { Get pointer to first hierarchical menu }
30  221 --             HMenuPtr := ORD4(ml^) + ml^^.sizeX6 + 12;
30  222 --
30  223 --             { get offset from HMenuPtr to last hierarchical menu }
30  224 --             lastHierMenu := PInteger(ORD4(ml^) + ml^^.sizeX6 + 6)^;
30  225 --             noOfHierMenus := lastHierMenu DIV 6;
30  226 --             FOR i := 1 TO noOfHierMenus DO
30  227 2-                 BEGIN
30  228 --                 BlockMove(Pointer(HMenuPtr), @menuRec, 6);
30  229 --                 WITH menuRec DO
30  230 --                 DoToMenu(mHandle);
30  231 --                 HMenuPtr := HMenuPtr + 6;
30  232 -2                 END;
30  233 -1             END;
30  234 --
30  235 -0 A   END;
30  236 --
30  237 --
30  238 --     {$S MARes}
30  239 --     {$IFC qTrace}{$D+}{$ENDC}
30  240 --     {-----------------------------------+
30  241 --     |   Enable                          |
30  242 --     +-----------------------------------}
30  243 -- A   PROCEDURE Enable(aCmd: CmdNumber; canDo: BOOLEAN);
30  244 --     VAR
30  245 --         menu, item:     INTEGER;
30  246 --         aMenuHandle:    MenuHandle;
30  247 0- A   BEGIN
30  248 --     {$IFC qTrace}
30  249 --         IF gTracing AND gTraceSetupMenus THEN
30  250 --             WriteLn('..... Enable(', aCmd:1, ', ', gBoolString[canDo], ')');
30  251 --     {$ENDC}
30  252 --         CmdToMenuItem(aCmd, menu, item);
30  253 --         aMenuHandle := GetMHandle(menu);
30  254 --         IF aMenuHandle <> NIL THEN
30  255 --             IF canDo THEN
30  256 --                 EnableItem(aMenuHandle, item)
30  257 --             ELSE
30  258 --                 DisableItem(aMenuHandle, item);
30  259 -0 A   END;
30  260 --     {$IFC qTrace}{$D++}{$ENDC}
30  261 --
30  262 --
30  263 --     {$S MARes}
30  264 --     {$IFC qTrace}{$D+}{$ENDC}
30  265 --     {-----------------------------------+
30  266 --     |   EnableCheck                     |
30  267 --     +-----------------------------------}
30  268 -- A   PROCEDURE EnableCheck(aCmd: CmdNumber; canDo, checkIt: BOOLEAN);
30  269 --     VAR
30  270 --         menu, item:     INTEGER;
30  271 --         aMenuHandle:    MenuHandle;
30  272 0- A   BEGIN
30  273 --     {$IFC qTrace}
30  274 --         IF gTracing AND gTraceSetupMenus THEN
30  275 --             WriteLn('..... EnableCheck(', aCmd:1, ', ', gBoolString[canDo], ', ', gBoolString[checkIt], ')');
30  276 --     {$ENDC}
30  277 --         CmdToMenuItem(aCmd, menu, item);
30  278 --         aMenuHandle := GetMHandle(menu);
30  279 --         IF aMenuHandle <> NIL THEN
30  280 1-             BEGIN
30  281 --             IF canDo THEN
30  282 --                 EnableItem(aMenuHandle, item)
30  283 --             ELSE
30  284 --                 DisableItem(aMenuHandle, item);
30  285 --             CheckItem(aMenuHandle, item, checkIt);
30  286 -1             END;
30  287 -0 A   END;
30  288 --     {$IFC qTrace}{$D++}{$ENDC}
30  289 --
30  290 --
30  291 --     {$S MARes}
30  292 --     {-----------------------------------+
30  293 --     |   GetResMenu                      |
30  294 --     +-----------------------------------}
30  295 -- A   FUNCTION  GetResMenu(menuResID: INTEGER): MenuHandle;
30  296 0- A   BEGIN
```

```
30  297 --          GetResMenu := MenuHandle(GetResource('MENU', menuResID));
30  298 -0 A    END;
30  299 --
30  300 --
30  301 --      {$S MAInit}
30  302 --      {-----------------------------------+
30  303 --      |   InitUMenuSetup                  |
30  304 --      +-----------------------------------}
30  305 -- A    PROCEDURE InitUMenuSetup;
30  306 --
30  307 --      TYPE
30  308 --          PJMP    =       ^JmpRec;
30  309 --          JmpRec  =       RECORD
30  310 --                              opcode: INTEGER;
30  311 --                              target: Ptr;
30  312 --                          END;
30  313 --
30  314 --      {$IFC qDebug}
30  315 --      VAR
30  316 --          i:                  INTEGER;
30  317 --          previousCmdNumber:  INTEGER;
30  318 --          aString:            Str255;
30  319 --      {$ENDC}
30  320 --
30  321 0- A    BEGIN
30  322 --          { read in the command table }
30  323 --          gCmdTable := CmdTable(GetResource('mntb', kIDMNTBbyCmdNumber));
30  324 --          FailNil(gCmdTable);
30  325 --          gSizeCmdTable := GetHandleSize(Handle(gCmdTable)) DIV SIZEOF(MenuCmdRecord);
30  326 --
30  327 --      {$IFC qDebug}
30  328 --          previousCmdNumber := MAXINT;
30  329 --          FOR i := 1 TO gSizeCmdTable DO
30  330 1-              BEGIN
30  331 --              IF gCmdTable^^[i].theCmdNumber = previousCmdNumber THEN
30  332 2-                  BEGIN
30  333 --                  NumToString(previousCmdNumber, aString);
30  334 --                  ProgramBreak(CONCAT('Command #', aString, ' is duplicated in the command table.'));
30  335 -2                  END;
30  336 --              previousCmdNumber := gCmdTable^^[i].theCmdNumber;
30  337 -1              END;
30  338 --      {$ENDC}
30  339 --
30  340 --          gHNullMenuProc := NewPermHandle(6);
30  341 --          FailNIL(gHNullMenuProc);
30  342 --          WITH PJMP(gHNullMenuProc^)^ DO
30  343 1-              BEGIN
30  344 --              opcode := $4EF9;            { JMP }
30  345 --              target := @NullMenuProc;
30  346 -1              END;
30  347 --          { expect 68020 cache to be flushed before this is used }
30  348 -0 A    END { InitUMenuSetup };
30  349 --
30  350 --
30  351 --      {$S MARes}
30  352 --      {-----------------------------------+
30  353 --      |   MAInsertMenu                    |
30  354 --      +-----------------------------------}
30  355 -- A    PROCEDURE MAInsertMenu (theMenu: MenuHandle; beforeID: INTEGER);
30  356 --      TYPE
30  357 --          MenuCRsrc = RECORD
30  358 --                          numEntries:     INTEGER;
30  359 --                          data:           ARRAY [1..1092] OF MCEntry;
30  360 --                      END;
30  361 --          PMenuCRsrc = ^MenuCRsrc;
30  362 --          HMenuCRsrc = ^PMenuCRsrc;
30  363 --      VAR
30  364 --          theColorRsrc:       HMenuCRsrc;
30  365 0- A    BEGIN
30  366 --          InsertMenu(theMenu, beforeID);
30  367 --          { Since only GetMenu automatically loads the appropriate color information,
30  368 --              and (sigh) since we can only call GetMenu once, and if you call DeleteMenu
30  369 --              all the good color stuff goes away (double sigh) we'll have to help out
30  370 --              the Menu Manager by doing its job for it }
30  371 --          IF gConfiguration.hasColorQD THEN
30  372 1-              BEGIN
30  373 --              theColorRsrc := HMenuCRsrc(GetResource('mctb', theMenu^^.menuID));
30  374 --              IF theColorRsrc <> NIL THEN
30  375 2-                  BEGIN
30  376 --                  HLock(Handle(theColorRsrc));
30  377 --                      WITH theColorRsrc^^ DO
30  378 --                          SetMCEntries(numEntries, @data[1]);
30  379 --                  HUnlock(Handle(theColorRsrc));
30  380 --                  ReleaseResource(Handle(theColorRsrc));
30  381 -2                  END;
30  382 -1              END;
30  383 -0 A    END;
30  384 --
30  385 --
30  386 --      {$S MARes}
30  387 --      {-----------------------------------+
30  388 --      |   NeedCalcMenuSize                |
30  389 --      +-----------------------------------}
30  390 -- A    PROCEDURE NeedCalcMenuSize(aMenuHandle: MenuHandle);
30  391 0- A    BEGIN
30  392 --          WITH aMenuHandle^^ DO
30  393 --              IF menuProc = gHNullMenuProc THEN
30  394 --                  menuWidth := 0;
30  395 -0 A    END;
```

```
30  396 --
30  397 --
30  398 --      {$S Main}
30  399 --          { a null menuProc that is used to inhibit re-calculating the menu's size after each
30  400 --              call to EnableItem, CheckItem, etc. }
30  401 --      {$IFC qTrace}{$D+}{$ENDC}
30  402 --      {-----------------------------------+
30  403 --      |   NullMenuProc                    |
30  404 --      +-----------------------------------}
30  405 -- A    PROCEDURE NullMenuProc(message: INTEGER; aMenuHandle: MenuHandle; VAR menuRect: Rect;
30  406 --                             hitPt: Point; VAR whichItem: INTEGER);
30  407 0- A    BEGIN
30  408 --          aMenuHandle^^.menuWidth := 0;
30  409 -0 A    END;
30  410 --      {$IFC qTrace}{$D++}{$ENDC}
30  411 --
30  412 --
30  413 --      {$S MARes}
30  414 --      {-----------------------------------+
30  415 --      |   PerformMenuSetup                |
30  416 --      +-----------------------------------}
30  417 -- A    PROCEDURE PerformMenuSetup (PROCEDURE TheMenuSetterUpper);
30  418 --
30  419 --      TYPE
30  420 --          EnableArray = ARRAY [0..mLastMenu] OF BOOLEAN;      { Saved enable flags }
30  421 --          SavePrArray = ARRAY [0..mLastMenu] OF Handle;       { Saved menu procs }
30  422 --
30  423 --      VAR
30  424 --          aMenuHandle:    MenuHandle;
30  425 --          wasEnabled:     EnableArray;
30  426 --          savedProcs:     SavePrArray;
30  427 --
30  428 --
30  429 --          {-------------------------------+
30  430 --          |   StartMenuSetup              |
30  431 --          +-------------------------------}
30  432 -- B        PROCEDURE StartMenuSetup(aMenuHandle: MenuHandle);
30  433 --          VAR
30  434 --              item:               INTEGER;
30  435 --              theCmd:             CHAR;
30  436 0- B        BEGIN
30  437 --              WITH aMenuHandle^^ DO
30  438 --                  IF (menuID >= 0) & (menuID <= mLastMenu) & (menuID <> mApple) THEN
30  439 1-                      BEGIN
30  440 --                      { Remember the menu itself was enabled, and disable the menu
30  441 --                          and all of its items. }
30  442 --                      wasEnabled[menuID] := Odd(enableFlags);
30  443 --                      enableFlags := 0;
30  444 --
30  445 --                      { Save the menu's menuproc and set it to the NullMenuProc, so that
30  446 --                          CalcMenuSize is disabled (will do ÇalcMenuSize at end of setup). }
30  447 --                      savedProcs[menuID] := menuProc; { See comment below }
30  448 --                      menuProc := gHNullMenuProc;
30  449 --
30  450 --                      { Uncheck all items. }
30  451 --                      FOR item := 1 TO CountMItems(aMenuHandle) DO
30  452 --                          { Make sure we don't check items with sub-menus }
30  453 --                          IF gConfiguration.hasHierarchicalMenus THEN
30  454 2-                              BEGIN
30  455 --                              GetItemCmd(aMenuHandle, item, theCmd);
30  456 --                              IF theCmd <> CHR($1B) THEN
30  457 --                                  CheckItem(aMenuHandle, item, FALSE);
30  458 -2                              END
30  459 --                          ELSE
30  460 --                              CheckItem(aMenuHandle, item, FALSE);
30  461 -1                      END;
30  462 -0 B        END { StartMenuSetup };
30  463 --
30  464 --          {-------------------------------+
30  465 --          |   EndMenuSetup                |
30  466 --          +-------------------------------}
30  467 -- B        PROCEDURE EndMenuSetup (aMenuHandle: MenuHandle);
30  468 --          VAR
30  469 --              newFlags: LONGINT;
30  470 --              menuID:     INTEGER;
30  471 0- B        BEGIN
30  472 --              WITH aMenuHandle^^ DO
30  473 --                  IF (menuID >= 0) & (menuID <= mLastMenu) & (menuID <> mApple) THEN
30  474 1-                      BEGIN
30  475 --                      newFlags := enableFlags;
30  476 --                      { If any items are enabled, enable the menu }
30  477 --                      IF newFlags <> 0 THEN
30  478 2-                          BEGIN
30  479 --                          newFlags := BOR(1, newFlags);
30  480 --                          enableFlags := newFlags;
30  481 -2                          END;
30  482 --
30  483 --                      { If the menu's enabled state changed, we have to draw the menu bar. }
30  484 --                      IF Odd(newFlags) <> wasEnabled[menuID] THEN
30  485 --                          gRedrawMenuBar := TRUE;
30  486 --
30  487 --                      { Restore the menuproc. }
30  488 --                      menuProc := savedProcs[menuID];
30  489 --
30  490 --                      { menuWidth set to 0 by routines that require CalcMenuSize, by
30  491 --                          calling NeedCalcMenu. }
30  492 --                      IF menuWidth = 0 THEN
30  493 --                          CalcMenuSize(aMenuHandle);
30  494 -1                      END;
```

```
30  495 -0 B       END { EndMenuSetup };
30  496 --
30  497 0- A   BEGIN
30  498 --         EachMenuDo(StartMenuSetup, TRUE);
30  499 --         TheMenuSetterUpper;
30  500 --         EachMenuDo(EndMenuSetup, TRUE);
30  501 --
30  502 --         IF gRedrawMenuBar THEN
30  503 1-             BEGIN
30  504 --             DrawMenuBar;
30  505 --             gRedrawMenuBar := FALSE;
30  506 -1             END;
30  507 --
30  508 --         gMenusAreSetup := TRUE;
30  509 -0 A   END { PerformMenuSetup };
30  510 --
30  511 --
30  512 --     {$S MARes}
30  513 --     {$IFC qTrace}{$D+}{$ENDC}
30  514 --     {-----------------------------------+
30  515 --     |   SetCmdIcon                      |
30  516 --     +-----------------------------------}
30  517 -- A   PROCEDURE SetCmdIcon(aCmd: CmdNumber; menuIcon: Byte);
30  518 --     VAR
30  519 --         menu, item:     INTEGER;
30  520 --         aMenuHandle:    MenuHandle;
30  521 0- A   BEGIN
30  522 --     {$IFC qTrace}
30  523 --         IF gTracing AND gTraceSetupMenus THEN
30  524 --             WriteLn('..... SetCmdIcon(', aCmd:1, ',', menuIcon:1, ')');
30  525 --     {$ENDC}
30  526 --         CmdToMenuItem(aCmd, menu, item);
30  527 --         aMenuHandle := GetResMenu(menu);
30  528 --         IF aMenuHandle <> NIL THEN
30  529 --             SetItemIcon(aMenuHandle, item, menuIcon);
30  530 -0 A   END;
30  531 --     {$IFC qTrace}{$D++}{$ENDC}
30  532 --
30  533 --
30  534 --     {$S MARes}
30  535 --     {$IFC qTrace}{$D+}{$ENDC}
30  536 --     {-----------------------------------+
30  537 --     |   SetCmdName                      |
30  538 --     +-----------------------------------}
30  539 -- A   PROCEDURE SetCmdName(aCmd: CmdNumber; menuText: Str255);
30  540 --     VAR
30  541 --         menu, item:     INTEGER;
30  542 --         aMenuHandle:    MenuHandle;
30  543 0- A   BEGIN
30  544 --     {$IFC qTrace}
30  545 --         IF gTracing AND gTraceSetupMenus THEN
30  546 --             WriteLn('..... SetCmdName(', aCmd:1, ', ''', menuText, ''')');
30  547 --     {$ENDC}
30  548 --         CmdToMenuItem(aCmd, menu, item);
30  549 --         aMenuHandle := GetResMenu(menu);
30  550 --         IF aMenuHandle <> NIL THEN
30  551 --             SetItem(aMenuHandle, item, menuText);
30  552 -0 A   END;
30  553 --     {$IFC qTrace}{$D++}{$ENDC}
30  554 --
30  555 --
30  556 --     {$S MARes}
30  557 --     {$IFC qTrace}{$D+}{$ENDC}
30  558 --     {-----------------------------------+
30  559 --     |   SetIndCmdName                   |
30  560 --     +-----------------------------------}
30  561 -- A   PROCEDURE SetIndCmdName(aCmd: CmdNumber; rsrcID, strIndex: INTEGER);
30  562 --     VAR
30  563 --         s:  Str255;
30  564 0- A   BEGIN
30  565 --         GetIndString(s, rsrcID, strIndex);
30  566 --         SetCmdName(aCmd, s);
30  567 -0 A   END;
30  568 --     {$IFC qTrace}{$D++}{$ENDC}
30  569 --
30  570 --
30  571 --     {$S MARes}
30  572 --     {$IFC qTrace}{$D+}{$ENDC}
30  573 --     {-----------------------------------+
30  574 --     |   SetMenuState                    |
30  575 --     +-----------------------------------}
30  576 -- A   PROCEDURE SetMenuState(aCmd: CmdNumber;
30  577 --                 rsrcID, falseBuzzItem, trueBuzzItem: INTEGER; stateVariable: BOOLEAN);
30  578 --     VAR
30  579 --         buzzItem:   INTEGER;
30  580 0- A   BEGIN
30  581 --         IF stateVariable THEN
30  582 --             buzzItem := trueBuzzitem
30  583 --         ELSE
30  584 --             buzzItem := falseBuzzItem;
30  585 --
30  586 --         SetIndCmdName(aCmd, rsrcID, buzzItem);
30  587 -0 A   END;
30  588 --     {$IFC qTrace}{$D++}{$ENDC}
30  589 --
30  590 --
30  591 --     {$S MARes}
30  592 --     {$IFC qTrace}{$D+}{$ENDC}
30  593 --     {-----------------------------------+
```

```
30  594 --      |   SetStyle                          |
30  595 --      +-------------------------------------}
30  596 -- A    PROCEDURE SetStyle(aCmd: CmdNumber; aStyle: Style);
30  597 --      VAR
30  598 --          menu, item:     INTEGER;
30  599 --          aMenuHandle:    MenuHandle;
30  600 0- A    BEGIN
30  601 --      {$IFC qTrace}
30  602 --          IF gTracing AND gTraceSetupMenus THEN
30  603 --              WriteLn('..... SetStyle(', aCmd:1, ',', QDByte(aStyle):1, ')');
30  604 --      {$ENDC}
30  605 --          CmdToMenuItem(aCmd, menu, item);
30  606 --          aMenuHandle := GetResMenu(menu);
30  607 --          IF aMenuHandle <> NIL THEN
30  608 --              SetItemStyle(aMenuHandle, item, aStyle);
30  609 -0 A    END;
30  610 --      {$IFC qTrace}{$D++}{$ENDC}
29  275 --
29  276 --      END { UMenuSetup }.
```

```
31    1 --    {$P}
31    2 --    { UObject.p }
31    3 --    { Copyright © 1984-1988  Apple Computer, Inc.  All rights reserved. }
31    4 --    { Converted for MPW; 13 May 1986 }
31    5 --
31    6 --    UNIT UObject;
31    7 --
31    8 --    INTERFACE
31    9 --
31   10 --    {-----------------------------------+
31   11 --    |   Uses                            |
31   12 --    +-----------------------------------}
31   13 --    USES
31   14 --        {$LOAD MacIntf.LOAD}
31   15 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
31   16 --        {$LOAD}
31   17 --            UMAUtil,
31   18 --            UFailure,
31   19 --            UMemory
31   20 --        {$IFC qWWNeeded},
31   21 --            UWritelnWindow
31   22 --        {$ENDC}
31   23 --        {$IFC qTrace},
31   24 --            UTrace
31   25 --        {$ENDC};
31   26 --
31   27 --    {-----------------------------------+
31   28 --    |   Global constants                |
31   29 --    +-----------------------------------}
31   30 --    CONST
31   31 --        kFailNone       =   1;
31   32 --        kFailAbstract   =   2;
31   33 --        kFailCoercion   =   3;
31   34 --
31   35 --        kMethodTrap     =   $A9F8;
31   36 --
31   37 --    {$IFC qDebug}
31   38 --        kInvalidObj = '*NotObj*';   { return value from LookupObjName
31   39 --                                                                  if not an object}
31   40 --    {$ENDC}
31   41 --
31   42 --    {-----------------------------------+
31   43 --    |   Global types                    |
31   44 --    +-----------------------------------}
31   45 --    TYPE
31   46 --
31   47 --    {$IFC qDebug}
31   48 --            { Format of object name table }
31   49 --        ObjNameRec = RECORD
31   50 --            objID:      INTEGER;
31   51 --            objName:    String8;
31   52 --            END;
31   53 --        HObjNameTbl = ^PObjNameTbl;
31   54 --        PObjNameTbl = ^AObjNameTbl;
31   55 --        AObjNameTbl = ARRAY[1..2500] OF ObjNameRec;
31   56 --    {$ENDC qDebug}
31   57 --
31   58 --
31   59 --        {-----------------------------------+
31   60 --        |   TObject                         |
31   61 --        +-----------------------------------}
31   62 --        TObject = OBJECT
31   63 --
31   64 --            FUNCTION  ShallowClone: TObject;
31   65 --                { Lowest level method for copying an object; should not be
31   66 --                    overridden except in very unusual cases.  Simply calls
31   67 --                    HandToHand to copy the object data. }
31   68 --
31   69 --            FUNCTION  Clone: TObject;
31   70 --                { Defaults to calling ShallowClone; can be overridden to copy
31   71 --                    objects refered to by fields. }
31   72 --
31   73 --            PROCEDURE ShallowFree;
31   74 --                { Lowest level method for freeing an object; should not be
31   75 --                    overridden except in very unusual cases.  Simply calls
31   76 --                    DisposHandle to free the object data. }
31   77 --
31   78 --            PROCEDURE Free;
31   79 --                { Defaults to calling ShallowFree; can be overridden to free objects
31   80 --                    refered to by fields. }
31   81 --
31   82 --    {$IFC qDebug}
31   83 --            FUNCTION  TypeName: String8;
31   84 --                { Returns the first 8 character's of an object's class name. }
31   85 --
31   86 --            PROCEDURE Inspect;
31   87 --                { Called from the debugger;
31   88 --                    TObject.Inspect displays the fields of the object in the writeln
31   89 --                    window by calling the object's Fields method, passing InspectField
31   90 --                    as the DoToField routine.  You can override this if you want to
31   91 --                    display any additional information in the writeln window when
31   92 --                    an object is inspected with the MacApp debugger. }
31   93 --
31   94 --            PROCEDURE TObject.GetInspectorName (VAR inspectorName: Str255);
31   95 --
31   96 --            PROCEDURE TObject.Fields (PROCEDURE DoToField (fieldName: Str255;
31   97 --                                                           fieldAddr: Ptr;
31   98 --                                                           fieldType: INTEGER));
```

```
31   99 --     {$ENDC}
31  100 --             END { TObject };
31  101 --
31  102 --
31  103 --     {-------------------------------------+
31  104 --     |   Global routines                   |
31  105 --     +-------------------------------------}
31  106 --     PROCEDURE FreeObject(obj: TObject);
31  107 --         { IF obj <> NIL THEN obj.Free; useful for freeing an object that might
31  108 --             sometimes be NIL. }
31  109 --
31  110 --
31  111 --     {$IFC qDebug}
31  112 --     PROCEDURE IdUobject;
31  113 --         { Writeln UObject compile date }
31  114 --
31  115 --     PROCEDURE WrLblField (fieldName: Str255; fieldAddr: Ptr; fieldType: INTEGER);
31  116 --         { Writes the given field in the writeln window in the form
31  117 --             WRITE(fieldName, '=', valueAsString).}
31  118 --
31  119 --     PROCEDURE InspectObject(obj: Handle);
31  120 --         { Called by debugger to inspect an object. }
31  121 --
31  122 --     FUNCTION IsObject (obj: TObject): BOOLEAN;
31  123 --         { Returns true if obj references a real object, false if obj is NIL or
31  124 --             a non-object reference. }
31  125 --
31  126 --     PROCEDURE BuildObjNameTable;
31  127 --         { Builds the object name table.  FindObjID can't be used until after
31  128 --             this has been called. }
31  129 --
31  130 --     FUNCTION  FindObjID(anObjName: String8): INTEGER;
31  131 --         { Given an object name, return its id.  Relies on gObjNameTable. }
31  132 --
31  133 --     PROCEDURE LookupObjName(objID: INTEGER; VAR objName: String8);
31  134 --         { Given an object id, return the object's name.  Does not need
31  135 --             gObjNameTable. }
31  136 --     {$ENDC qDebug}
31  137 --
31  138 --
31  139 --     PROCEDURE OBJFail(error: INTEGER); { Called by UTrace }
31  140 --
31  141 --     {$%+        Enable '%' in identifiers}
31  142 --
31  143 --     PROCEDURE %_Pgml;
31  144 --         { The Pascal compiler generates code to call this procedure before
31  145 --             the first unit is initialized. }
31  146 --
31  147 --     FUNCTION  %_OBCHK(ordObject, jumpTablePtr: LONGINT): LONGINT;
31  148 --         { Called before the first unit is gInitialized.  This returns its first
31  149 --             argument IF it is NIL or passes a class-membership check; ELSE calls
31  150 --             OBJFail  }
31  151 --
31  152 --     { Calls to following generated by compiler for New(obj) }
31  153 --     PROCEDURE %_OBNEW(VAR obj: LONGINT; jumpTablePtr: LONGINT; itsSize: INTEGER);
31  154 --         { when type of obj in same unit }
31  155 --
31  156 --     PROCEDURE %_OBDISP(VAR obj: Handle);
31  157 --
31  158 --     {$%-        Disable '%' in identifiers}
31  159 --
31  160 --     IMPLEMENTATION
31  161 --
31  162 --     {$%+        Enable '%' in identifiers}
31  163 --
31  164 --     {$I UObject.incl.p}
```

```
32    1 --      {$P}
32    2 --      { UObject.incl.p }
32    3 --      { Copyright © 1984-1988 by Apple Computer, Inc.  All rights reserved. }
32    4 --
32    5 --
32    6 --      {-----------------------------------+
32    7 --      |   Local variables                 |
32    8 --      +-----------------------------------}
32    9 --      {$IFC qDebug}
32   10 --      VAR
32   11 --          gObjNameTable:      HObjNameTbl;    { handle to table of object names }
32   12 --          gInspectLinePos:    INTEGER;        { Used to do line breaks when inspecting fields. }
32   13 --      {$ENDC}
32   14 --
32   15 --
32   16 --      {$R-        Turn range checking off}
32   17 --
32   18 --      {$SETC qMacApp := TRUE}
32   19 --      {$I ObjLib.incl.p}
33    1 --      {ObjLib.incl.p}
33    2 --      {Copyright © 1984-1988  Apple Computer, Inc.  All rights reserved.}
33    3 --
33    4 --      {This is an alternate library for Object Pascal, Assembler, etc users who
33    5 --       do not wish to use MacApp.  MacApp users should use UObject instead}
33    6 --
33    7 --      {-----------------------------------+
33    8 --      |   External declarations           |
33    9 --      +-----------------------------------}
33   10 -- A    PROCEDURE OBJFail(error: INTEGER); EXTERNAL;
33   11 -- A    FUNCTION  %_InObj(ordObject, jumpTablePtr: LONGINT): BOOLEAN; EXTERNAL;
33   12 -- A    PROCEDURE %_SetClassIndex(ordObject, jumpTablePtr: LONGINT); EXTERNAL;
33   13 --
33   14 -- A    PROCEDURE %_RTS1; EXTERNAL;
33   15 -- A    PROCEDURE %_InitObj; EXTERNAL;
33   16 --
33   17 --      {$SETC qInspector := TRUE}
33   18 --      {$IFC qInspector}
33   19 -- A    PROCEDURE AddObjectToInspector (theObject: TObject); EXTERNAL;
33   20 -- A    PROCEDURE RemoveObjectFromInspector (theObject: TObject); EXTERNAL;
33   21 --      {$ENDC}
33   22 --
33   23 --      (*****************************************************************************************)
33   24 --      (*      G l o b a l   R o u t i n e s                                                    *)
33   25 --      (*****************************************************************************************)
33   26 --      {$IFC qTrace}{$D+}{$ENDC}
33   27 --      {$S Main}
33   28 --      {-----------------------------------+
33   29 --      |   %_Pgm1                          |
33   30 --      +-----------------------------------}
33   31 --
33   32 --      {$J+}
33   33 --      VAR
33   34 --          %_METHCACHE:    Ptr;
33   35 --      {$J-}
33   36 --
33   37 -- A    PROCEDURE %_Pgm1;
33   38 --          {The Pascal compiler generates code to call this procedure auto-
33   39 --              matically, before initializing the units and starting the
33   40 --              application's main program.  This function must always work on 64K
33   41 --              ROMs.}
33   42 --
33   43 0- A    BEGIN
33   44 --          {Important note for understanding why this is here:
33   45 --
33   46 --              In an unoptimized program, %_InitObj gets called, which does nothing
33   47 --              but return.  However, in an optimized program, the Linker changes
33   48 --              this to call %_InitOptObj, was does do something.  So don't be
33   49 --              fooled into believing that the next statement is useless.}
33   50 --
33   51 --          %_MethCache := NewPtr (256 * 8);    { Used for non-opt, thrown away in opt. }
33   52 --          %_InitObj;                          {A no-op for non-optimized code}
33   53 -0 A    END {%_Pgm1};
33   54 --
33   55 --      {$IFC qMacApp}
33   56 --      {$S MAMain}
33   57 --      {$ENDC}
33   58 --      {-----------------------------------+
33   59 --      |   %_OBNEW                         |
33   60 --      +-----------------------------------}
33   61 -- A    PROCEDURE %_OBNEW (VAR obj: LONGINT; jumpTablePtr: LONGINT; itsSize: INTEGER);
33   62 --          {$IFC qDebug}
33   63 --          CONST initVal = $3F21;
33   64 --
33   65 --          VAR i:      INTEGER;    {must be first local and 2 bytes in size}
33   66 --              p:      PInteger;
33   67 --              n:      String8;
33   68 --              s:      Str255;
33   69 --          {$ENDC}
33   70 --
33   71 0- A    BEGIN
33   72 --      {$IFC qDebug}
33   73 --          IF gAskAboutAlloc AND CanReadLn THEN
33   74 1-              BEGIN
33   75 --
33   76 --                  GetMethodName(%_GetA6+4, s);
33   77 --                  LookupObjName(jumpTablePtr-%_GetA5, n);
33   78 --
33   79 --                  Writeln('Within ', s, ', trying to make a ''', n, '''.');
```

```
33   80 --              IF ReadYesNo('     Return NIL (Y or N) [N]? ') THEN
33   81 2-                  BEGIN
33   82 --                  obj := 0;
33   83 --                  EXIT(%_OBNEW);
33   84 -2                  END;
33   85 -1              END;
33   86 --      {$ENDC qDebug}
33   87 --
33   88 --      {$IFC qMacApp}
33   89 --          obj := ORD(NewPermHandle(itsSize));
33   90 --      {$ELSEC}
33   91 --          obj := ORD(NewHandle(itsSize));
33   92 --      {$ENDC}
33   93 --
33   94 --          IF obj <> 0 THEN
33   95 1-              BEGIN
33   96 --
33   97 --              {$IFC qDebug}
33   98 --              {Initialize the object to $3F213F21...}
33   99 --              p := PInteger(Handle(obj)^);
33  100 --              FOR i := 1 TO itsSize DIV 2 DO
33  101 2-                  BEGIN
33  102 --                  p^ := initVal;
33  103 --                  p := PInteger(ORD(p) + 2);
33  104 -2                  END;
33  105 --              {$ENDC}
33  106 --
33  107 --              {Install class index}
33  108 --              %_SetClassIndex(obj, jumpTablePtr);
33  109 --
33  110 --      {$IFC qInspector}
33  111 --              AddObjectToInspector(TObject(obj));
33  112 --      {$ENDC}
33  113 -1              END;
33  114 -0 A   END {%_OBNEW};
33  115 --
33  116 --
33  117 --      {$IFC qMacApp}
33  118 --      {$S MAMain}
33  119 --      {$ENDC}
33  120 --      {-----------------------------------+
33  121 --      |   %_OBDISP                        |
33  122 --      +-----------------------------------}
33  123 -- A   PROCEDURE %_OBDISP(VAR obj: Handle);
33  124 0- A   BEGIN
33  125 --          DisposHandle(obj);
33  126 -0 A   END {%_OBDISP};
33  127 --
33  128 --
33  129 --      {$IFC qMacApp}
33  130 --      {$S MAMain}
33  131 --      {$ENDC}
33  132 --      {-----------------------------------+
33  133 --      |   %_OBCHK                         |
33  134 --      +-----------------------------------}
33  135 -- A   FUNCTION %_OBCHK(ordObject, jumpTablePtr: LONGINT): LONGINT;
33  136 0- A   BEGIN
33  137 --          %_OBCHK := ordObject;
33  138 --          IF ordObject <> 0 THEN
33  139 --              IF NOT %_InObj(ordObject, jumpTablePtr) THEN
33  140 1-                  BEGIN
33  141 --                  OBJFail(kFailCoercion);
33  142 --                  EXIT(%_OBCHK);
33  143 -1                  END;
33  144 -0 A   END {%_OBCHK};
33  145 --      {$IFC qTrace}{$D++}{$ENDC}
33  146 --
33  147 --
33  148 --      (*****************************************************************************************)
33  149 --      (*      T O b j e c t                                                                    *)
33  150 --      (*****************************************************************************************)
33  151 --      {$IFC qMacApp}
33  152 --      {$S MAMain}
33  153 --      {$ENDC}
33  154 --      {-----------------------------------+
33  155 --      |   ShallowClone                    |
33  156 --      +-----------------------------------}
33  157 -- A   FUNCTION  TObject.ShallowClone;
33  158 --
33  159 --          VAR result:     Handle;
33  160 --              oldPerm:    BOOLEAN;
33  161 --              err:        OSErr;
33  162 --
33  163 --      {$IFC qDebug}
33  164 --              n:          String8;
33  165 --              s:          Str255;
33  166 --              c:          String8;
33  167 --              m:          String8;
33  168 --      {$ENDC}
33  169 --
33  170 0- A   BEGIN
33  171 --      {$IFC qDebug}
33  172 --          IF gAskAboutAlloc THEN
33  173 1-              BEGIN
33  174 --              n := TypeName;
33  175 --
33  176 --              GetProcName(%_GetA6+4, c, m);
33  177 --
33  178 --              IF (c = 'TOBJECT ') AND (m = 'CLONE   ') THEN {report about caller of TObject.Clone, instead}
```

```
33  179 --                GetProcName(PLongInt(%_GetA6)^ + 4, c, m);
33  180 --
33  181 --            IF c = kSpace8 THEN
33  182 --                s := m
33  183 --            ELSE
33  184 --                s := CONCAT(c, '.', m);
33  185 --
33  186 --            Writeln('Within ', s, ', trying to clone a ''', n, '''.');
33  187 --
33  188 --            IF ReadYesNo('     Return NIL (Y or N) [N]? ') THEN
33  189 2-                BEGIN
33  190 --                ShallowClone := NIL;
33  191 --                EXIT(ShallowClone);
33  192 -2                END;
33  193 -1            END;
33  194 --     {$ENDC}
33  195 --
33  196 --         result := Handle(SELF);
33  197 --
33  198 --     {$IFC qMacApp}
33  199 --         oldPerm := PermAllocation(TRUE);
33  200 --     {$ENDC}
33  201 --
33  202 --         err := HandToHand(result);
33  203 --
33  204 --     {$IFC qMacApp}
33  205 --         oldPerm := PermAllocation(oldPerm);
33  206 --     {$ENDC}
33  207 --
33  208 --         IF err <> noErr THEN
33  209 --             result := NIL
33  210 --     {$IFC qInspector}
33  211 --         ELSE
33  212 --             AddObjectToInspector(TObject(result))
33  213 --     {$ENDC}
33  214 --             ;
33  215 --
33  216 --         ShallowClone := TObject(result);
33  217 -0 A   END {TObject.ShallowClone};
33  218 --
33  219 --
33  220 --     {$IFC qMacApp}
33  221 --     {$S MAMain}
33  222 --     {$ENDC}
33  223 --     {-----------------------------------+
33  224 --     |   Clone                           |
33  225 --     +-----------------------------------}
33  226 -- A   FUNCTION  TObject.Clone;
33  227 0- A   BEGIN
33  228 --         Clone := ShallowClone;
33  229 -0 A   END {TObject.Clone};
33  230 --
33  231 --
33  232 --     {$IFC qMacApp}
33  233 --     {$S MAMain}
33  234 --     {$ENDC}
33  235 --     {-----------------------------------+
33  236 --     |   ShallowFree                     |
33  237 --     +-----------------------------------}
33  238 -- A   PROCEDURE TObject.ShallowFree;
33  239 0- A   BEGIN
33  240 --         DisposHandle(Handle(SELF));
33  241 -0 A   END {TObject.ShallowFree};
33  242 --
33  243 --
33  244 --     {$IFC qMacApp}
33  245 --     {$S MAMain}
33  246 --     {$ENDC}
33  247 --     {-----------------------------------+
33  248 --     |   Free                            |
33  249 --     +-----------------------------------}
33  250 -- A   PROCEDURE TObject.Free;
33  251 0- A   BEGIN
33  252 --     {$IFC qInspector}
33  253 --         RemoveObjectFromInspector(SELF);
33  254 --     {$ENDC}
33  255 --         ShallowFree;
33  256 -0 A   END {TObject.Free};
33  257 --
33  258 --
33  259 --     {$IFC qInspector}
33  260 --     {$S MAInspector}
33  261 --     {-----------------------------------+
33  262 --     |   GetInspectorName                |
33  263 --     +-----------------------------------}
33  264 -- A   PROCEDURE TObject.GetInspectorName (VAR inspectorName: Str255);
33  265 0- A   BEGIN
33  266 -0 A   END {TObject.GetInspectorName};
33  267 --     {$ENDC}
32   20 --
32   21 --
32   22 --     (*****************************************************************************************)
32   23 --     (*      G l o b a l   r o u t i n e s                                                    *)
32   24 --     (*****************************************************************************************)
32   25 --     {$IFC qMacApp AND qDebug}
32   26 --     {$IFC qTrace}{$D+}{$ENDC}
32   27 --     {$S MAInit}
32   28 --     {-----------------------------------+
32   29 --     |   BuildObjNameTable               |
```

```
32   30 --     +---------------------------------}
32   31 -- A   PROCEDURE BuildObjNameTable;
32   32 --
32   33 --     CONST
32   34 --         empty = '12345678';
32   35 --
32   36 --     TYPE
32   37 --         ZPPAOC8 = ^ZPAOC8;
32   38 --         ZPAOC8 = PACKED ARRAY[0..7] OF Char;
32   39 --
32   40 --     VAR
32   41 --             i:          INTEGER;
32   42 --             objRec:     ObjNameRec;
32   43 --             methTable:  Handle;
32   44 --             p:          PInteger;
32   45 --             backP:      PInteger;
32   46 --             endMeth:    PInteger;
32   47 --             offset:     LONGINT;
32   48 --
32   49 0- A   BEGIN
32   50 --         gObjNameTable := HObjNameTbl(NewHandle(0));
32   51 --         methTable := GetNamedResource('CODE', '%_MethTables');
32   52 --         IF methTable <> NIL THEN
32   53 1-             BEGIN
32   54 --             p := PInteger(methTable^);
32   55 --             endMeth := PInteger(Ord(p) + SizeResource(methTable));
32   56 --
32   57 --             WHILE Ord(p) <= Ord(endMeth)-8 DO
32   58 2-                 BEGIN
32   59 --                 IF ZPPAOC8(p)^ = 'CLASINFO' THEN
32   60 3-                     BEGIN
32   61 --                         { we are at the end of the object info; search backwards }
32   62 --                     backP := p;
32   63 --                     objRec.objName := empty;
32   64 --
32   65 --                         { copy the object type name into the record }
32   66 --                     FOR i := 8 DOWNTO 1 DO
32   67 4-                         BEGIN
32   68 --                         backP := PInteger(Ord(backP) - 1);
32   69 --                         objRec.objName[i] := Chr(BAND(PByte(backP)^, 127));
32   70 -4                         END;
32   71 --
32   72 --                         { find the # of methods for this object type }
32   73 --                     backP := PInteger(Ord(backP) - 2);
32   74 --                     i := 0;
32   75 --                     WHILE i <> backP^ DO
32   76 4-                         BEGIN
32   77 --                         i := i + 1;
32   78 --                         backP := PInteger(Ord(backP) - 4);
32   79 -4                         END;
32   80 --
32   81 --                     backP := PInteger(Ord(backP) - 6);
32   82 --                     objRec.objID := backP^ + 2; { object ID is 2 more than jt offset }
32   83 --
32   84 --                         { add it to the list }
32   85 --                     offset := Munger(Handle(gObjNameTable), 0,
32   86 --                                 NIL, 0, @objRec, SIZEOF(objRec));
32   87 --
32   88 --                     p := PInteger(Ord(p) + 8);  { skip 'CLASINFO' }
32   89 -3                     END
32   90 --                 ELSE
32   91 --                     p := PInteger(Ord(p) + 2);
32   92 -2                 END;
32   93 -1             END;
32   94 -0 A   END { BuildObjNameTable };
32   95 --     {$IFC qTrace}{$D++}{$ENDC}
32   96 --     {$ENDC qMacApp AND qDebug}
32   97 --
32   98 --
32   99 --     {$IFC qDebug}
32  100 --     {$IFC qTrace}{$D+}{$ENDC}
32  101 --     {$S MADebug}
32  102 --     {---------------------------------+
32  103 --     |   FindObjID                     |
32  104 --     +---------------------------------}
32  105 -- A   FUNCTION  FindObjID(anObjName: String8): INTEGER;
32  106 --         VAR i:          INTEGER;
32  107 0- A   BEGIN
32  108 --             { pad objName to 8 chars and uppercase it }
32  109 --         anObjName := Copy(Concat(anObjName, kSpace8), 1, 8);
32  110 --         UprString8(anObjName);
32  111 --
32  112 --         i := GetHandleSize(Handle(gObjNameTable)) DIV SIZEOF(ObjNameRec);
32  113 --         WHILE i > 0 DO
32  114 1-             BEGIN
32  115 --             WITH gObjNameTable^^[i] DO
32  116 --                 IF anObjName = objName THEN
32  117 2-                     BEGIN
32  118 --                     FindObjId := objID;
32  119 --                     Exit(FindObjID);
32  120 -2                     END;
32  121 --             i := i - 1;
32  122 -1             END;
32  123 --
32  124 --         ProgramBreak(CONCAT('FindObjId: Can''t find object name ', anObjName));
32  125 --         FindObjID := 0;
32  126 --
32  127 -0 A   END { FindObjID };
32  128 --     {$IFC qTrace}{$D++}{$ENDC}
```

```
32  129 --      {$ENDC qDebug}
32  130 --
32  131 --
32  132 --      {$S MAMain}
32  133 --      {$IFC qTrace}{$D+}{$ENDC}
32  134 --      {-----------------------------------+
32  135 --      |   FreeObject                      |
32  136 --      +-----------------------------------}
32  137 -- A    PROCEDURE FreeObject(obj: TObject);
32  138 --
32  139 0- A    BEGIN
32  140 --          IF obj <> NIL THEN
32  141 --              obj.Free;
32  142 -0 A    END;
32  143 --      {$IFC qTrace}{$D++}{$ENDC}
32  144 --
32  145 --
32  146 --      {$IFC qDebug}
32  147 --      {$IFC qTrace}{$D+}{$ENDC}
32  148 --      {$S MAInspect}  { MacApp InitToolbox sets this segment resident }
32  149 --      {-------------------------------+
32  150 --      |   InspectField                |
32  151 --      +-------------------------------}
32  152 -- A    PROCEDURE InspectField (fieldName: Str255; fieldAddr: Ptr; fieldType: INTEGER);
32  153 --
32  154 --      VAR
32  155 --          s:      Str255;
32  156 --          x:      INTEGER;
32  157 --
32  158 0- A    BEGIN
32  159 --      {$IFC qWWNeeded}            { Can't do this if there is no writeln window }
32  160 --          IF fieldType <> bClass THEN
32  161 1-              BEGIN
32  162 --              FieldToString(fieldAddr, fieldType, s);
32  163 --              x := LENGTH(fieldName) + 1 + LENGTH(s);
32  164 --              IF gInspectLinePos + x + 2 >= gWWPerLine THEN
32  165 2-                  BEGIN
32  166 --                  WRITELN;
32  167 --                  gInspectLinePos := 0;
32  168 -2                  END
32  169 --              ELSE IF gInspectLinePos <> 0 THEN           { If not at the start of a line     }
32  170 2-                  BEGIN
32  171 --                  WRITE('  ');
32  172 --                  gInspectLinePos := gInspectLinePos + 2;
32  173 -2                  END;
32  174 --              WRITE(fieldName, '=', s);
32  175 --              gInspectLinePos := gInspectLinePos + x;
32  176 -1              END;
32  177 --      {$ENDC}
32  178 -0 A    END;
32  179 --      {$IFC qTrace}{$D++}{$ENDC}
32  180 --      {$ENDC qDebug}
32  181 --
32  182 --
32  183 --      {$IFC qDebug}
32  184 --      {$IFC qTrace}{$D+}{$ENDC}
32  185 --      {$S MAInspect}  { MacApp InitToolbox sets this segment resident }
32  186 --      {-----------------------------------+
32  187 --      |   InspectObject                   |
32  188 --      +-----------------------------------}
32  189 -- A    PROCEDURE InspectObject(obj: Handle);
32  190 --      VAR
32  191 --          hex:        String8;
32  192 --          x:          INTEGER;
32  193 --
32  194 0- A    BEGIN
32  195 --          IF IsObject(TObject(obj)) THEN
32  196 1-              BEGIN
32  197 --              x := 0;
32  198 --              HLock(Handle(obj));
32  199 --              TObject(obj).Inspect;
32  200 --              WRITELN;
32  201 --              HUnlock(Handle(obj));
32  202 -1              END
32  203 --          ELSE
32  204 1-              BEGIN
32  205 --              LintToHex(ORD(obj), hex, 6);
32  206 --              Writeln('$', hex, ' is not a TObject!');
32  207 -1              END;
32  208 -0 A    END { InspectObject };
32  209 --      {$IFC qTrace}{$D++}{$ENDC}
32  210 --      {$ENDC qDebug}
32  211 --
32  212 --
32  213 --      {$IFC qDebug}
32  214 --      {$IFC qTrace}{$D+}{$ENDC}
32  215 --      {$S MAInspect}
32  216 --      {-----------------------------------+
32  217 --      |   IsObject                        |
32  218 --      +-----------------------------------}
32  219 -- A    FUNCTION  IsObject (obj: TObject): BOOLEAN;
32  220 --      VAR
32  221 --          ptrTObject:     LONGINT;
32  222 0- A    BEGIN
32  223 --          IF NOT ODD(ORD4(obj)) THEN
32  224 1-              BEGIN
32  225 --              ptrTObject := %_GetA5 + FindObjID('TObject');
32  226 --              IsObject := %_InObj(ORD4(obj), ptrTObject);
32  227 -1              END
```

```
32  228 --          ELSE
32  229 --              IsObject := FALSE;
32  230 -0 A    END { IsObject };
32  231 --      {$IFC qTrace}{$D++}{$ENDC}
32  232 --      {$ENDC qDebug}
32  233 --
32  234 --
32  235 --      {$IFC qDebug}
32  236 --      {$IFC qTrace}{$D+}{$ENDC}
32  237 --      {$S MADebug}
32  238 --      {-----------------------------------+
32  239 --      |   LookupObjName                   |   Given A5-offset return Object Type Name
32  240 --      +-----------------------------------}
32  241 -- A    PROCEDURE LookupObjName(objID: INTEGER; VAR objName: String8);
32  242 0- A    BEGIN
32  243 --          TRCObjTypeName(objID, String8(objName));
32  244 -0 A    END { LookupObjName };
32  245 --      {$IFC qTrace}{$D++}{$ENDC}
32  246 --      {$ENDC qDebug}
32  247 --
32  248 --
32  249 --      {$IFC qDebug}
32  250 --      {$IFC qTrace}{$D+}{$ENDC}
32  251 --      {$S MAInspect}
32  252 --      {-----------------------------------+
32  253 --      |   WrLblField                      |   Given A5-offset return Object Type Name
32  254 --      +-----------------------------------}
32  255 -- A    PROCEDURE WrLblField (fieldName: Str255; fieldAddr: Ptr; fieldType: INTEGER);
32  256 0- A    BEGIN
32  257 --          gInspectLinePos := 0;
32  258 --          InspectField(fieldName, fieldAddr, fieldType);
32  259 --          gInspectLinePos := 0;
32  260 -0 A    END {WrLblField};
32  261 --      {$IFC qTrace}{$D++}{$ENDC}
32  262 --      {$ENDC qDebug}
32  263 --
32  264 --
32  265 --      (*****************************************************************************************)
32  266 --      (*      T O b j e c t                                                                    *)
32  267 --      (*****************************************************************************************)
32  268 --      {$IFC qDebug}
32  269 --      {$S MADebug}
32  270 --      {$IFC qTrace}{$D+}{$ENDC}
32  271 --      {-----------------------------------+
32  272 --      |   TypeName                        |
32  273 --      +-----------------------------------}
32  274 -- A    FUNCTION  TObject.TypeName: String8;
32  275 --      VAR
32  276 --          result:     String8;
32  277 0- A    BEGIN
32  278 --          LookupObjName(HInteger(SELF)^^, result);
32  279 --          TypeName := result;
32  280 -0 A    END;
32  281 --      {$IFC qTrace}{$D++}{$ENDC}
32  282 --
32  283 --
32  284 --      {$IFC qTrace}{$D+}{$ENDC}
32  285 --      {$S MAFields}
32  286 --      {-----------------------------------+
32  287 --      |   Fields                          |
32  288 --      +-----------------------------------}
32  289 -- A    PROCEDURE TObject.Fields (PROCEDURE DoToField (fieldName: Str255;
32  290 --                                                     fieldAddr: Ptr;
32  291 --                                                     fieldType: INTEGER));
32  292 0- A    BEGIN
32  293 --          DoToField('TObject', NIL, bClass);
32  294 -0 A    END;
32  295 --      {$IFC qTrace}{$D++}{$ENDC}
32  296 --
32  297 --
32  298 --      {$IFC qTrace}{$D+}{$ENDC}
32  299 --      {$S MAInspect}
32  300 --      {-----------------------------------+
32  301 --      |   Inspect                         |
32  302 --      +-----------------------------------}
32  303 -- A    PROCEDURE TObject.Inspect;
32  304 --      VAR
32  305 --          hex:        String8;
32  306 0- A    BEGIN
32  307 --          LIntToHex(ORD(SELF), hex, 6);
32  308 --          Writeln(TypeName, '     $', hex);
32  309 --          gInspectLinePos := 0;
32  310 --          Fields(InspectField);
32  311 -0 A    END;
32  312 --      {$IFC qTrace}{$D++}{$ENDC}
32  313 --      {$ENDC qDebug}
32  314 --
32  315 --
32  316 --      {$IFC qDebug}
32  317 --      {$S MADebug}
32  318 --      {-----------------------------------+
32  319 --      |   IDUobject                       |
32  320 --      +-----------------------------------}
32  321 -- A    PROCEDURE IDUobject;
32  322 0- A    BEGIN
32  323 --          Writeln('UObject of ', COMPDATE, ' @ ', COMPTIME);
32  324 -0 A    END;
32  325 --      {$ENDC}
31  165 --
```

```
31  166 --     END.
```

```
34     1 --     {$P}
34     2 --     { UPatch.p }
34     3 --     { Copyright 1985-1988 by Apple Computer, Inc.  All rights reserved. }
34     4 --
34     5 --     { Conditional compilation flags used by this unit:
34     6 --
34     7 --         qNeedsROM128K   True skips code for Mac XL and 64K ROMs,
34     8 --                         false leaves it in.
34     9 --
34    10 --         The settings of $D, $R and $N are imported from UMAUtil.
34    11 --      }
34    12 --
34    13 --     UNIT UPatch;
34    14 --
34    15 --     INTERFACE
34    16 --
34    17 --     {
34    18 --                     T H E O R Y   O F   O P E R A T I O N
34    19 --
34    20 --         This unit implements a general trap-patching mechanism.  Two types of
34    21 --         patches are implemented:
34    22 --
34    23 --         Normal Patch    The trap is patched such that it jumps to the routine
34    24 --                         of your choice, ignoring the routine it previously
34    25 --                         jumped to.  On a 128K ROM machine, the patch is made
34    26 --                         by setting the trap address to your routine.  On a 64K
34    27 --                         machine, a small block of code is allocated in the
34    28 --                         system heap:  The trap address is set to point to the
34    29 --                         block, and the block contains code to jump to your
34    30 --                         routine.
34    31 --
34    32 --         Tail Patch      The trap is patched such that it jumps to the
34    33 --                         routine of your choice, then jumps to the trap's
34    34 --                         routine.  Tail patches come in two flavors:  with
34    35 --                         no parameters and with one long-word parameter.  The
34    36 --                         patch is made by allocating a small block of code in
34    37 --                         the application heap (system heap for 64K ROMs).  The
34    38 --                         trap address is set to the block, and the block contains
34    39 --                         code to jump to your routine and return to the trap's
34    40 --                         original address.
34    41 --
34    42 --         References to patches are kept in TrapPatch records.  For each trap
34    43 --         patched you must define a variable of type TrapPatch.  This unit links
34    44 --         the TrapPatch records to form a linked list for facilitate unpatching
34    45 --         all patches without specifying each individual patch.
34    46 --      }
34    47 --
34    48 --     {-----------------------------------+
34    49 --     |   Uses                            |
34    50 --     +-----------------------------------}
34    51 --     USES
34    52 --         {$LOAD MacIntf.LOAD}
34    53 --             MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
34    54 --         {$LOAD}
34    55 --             UMAUtil;
34    56 --
34    57 --     {-----------------------------------+
34    58 --     |   Global types                    |
34    59 --     +-----------------------------------}
34    60 --     TYPE
34    61 --         TrapPatchPtr =      ^TrapPatch;
34    62 --         TrapPatch =         RECORD
34    63 --                             jmpPtr:         Ptr;            { Addr of sys heap block }
34    64 --                             trapNum:        INTEGER;        { trap # being patched }
34    65 --                             oldTrapAddr:    LONGINT;        { old trap address }
34    66 --                             nextPatch:      TrapPatchPtr;   { linked list of patches }
34    67 --                             END;
34    68 --
34    69 --
34    70 --     {-----------------------------------+
34    71 --     |   Global routines                 |
34    72 --     +-----------------------------------}
34    73 --     PROCEDURE InitUPatch;
34    74 --         { Call this once to set things up.  Initializes the linked list of
34    75 --             patches to NIL. }
34    76 --
34    77 --     FUNCTION  PatchTrap(VAR thePatch: TrapPatch; theTrapNum: INTEGER;
34    78 --             theRoutine: Ptr): INTEGER;
34    79 --         { PatchTrap patches the given trap to theRoutine.  On a 128K ROM
34    80 --             machine, we patch the trap's address directly.  On a 64K ROM
34    81 --             machine, we must set up a block in the system heap, patch the
34    82 --             trap to jump to the block, and insert code in the block to jump
34    83 --             to theRoutine. }
34    84 --
34    85 --     FUNCTION  TailPatch(VAR thePatch: TrapPatch; theTrapNum: INTEGER;
34    86 --             theRoutine: Ptr): INTEGER;
34    87 --         { TailPatch patches a trap such that it calls theRoutine first, then
34    88 --             executes the old trap routine.  theRoutine must be a procedure
34    89 --             with no arguments.  This type of patch is implemented by setting
34    90 --             up a block in the system heap, patching the trap to jump to the
34    91 --             block, and inserting code in the block that pushes the original
34    92 --             trap address on the stack and jumps to theRoutine.  Thus,
34    93 --             theRoutine returns to the original trap routine. }
34    94 --
34    95 --     FUNCTION  Tail1Patch(VAR thePatch: TrapPatch; theTrapNum: INTEGER;
34    96 --             theRoutine: Ptr): INTEGER;
34    97 --         { Tail1Patch is just like TailPatch, except that it is intended to patch
34    98 --             traps with a single long-word parameter.  The parameter passed to
```

```
34   99 --              the trap is also passed to theRoutine.  theRoutine must be a
34  100 --              procedure with one long-word parameter. }
34  101 --
34  102 --      PROCEDURE UnpatchTrap(VAR thePatch: TrapPatch);
34  103 --          { UnpatchTrap unpatches the given trap. }
34  104 --
34  105 --      PROCEDURE UnpatchAll;
34  106 --          { UnpatchAll unpatches all traps. }
34  107 --
34  108 --
34  109 --      IMPLEMENTATION
34  110 --
34  111 --      {$I UPatch.incl.p}
```

```
35    1 --      {$P}
35    2 --      { UPatch.incl.p }
35    3 --      { Copyright 1985-1988 by Apple Computer, Inc.  All rights reserved. }
35    4 --
35    5 --      {-----------------------------------+
35    6 --      |   Local types                     |
35    7 --      +-----------------------------------}
35    8 --      TYPE
35    9 --          { System heap block record for PatchTrap }
35   10 --          PTBlockPtr =    ^PTBlock;
35   11 --          PTBlock =       RECORD
35   12 --                              Jmp:            INTEGER;
35   13 --                              Routine:        Ptr;
35   14 --                          END;
35   15 --
35   16 --          { System heap block record for TailPatch }
35   17 --          TPBlockPtr =    ^TPBlock;
35   18 --          TPBlock =       RECORD
35   19 --                              MoveL:          INTEGER;
35   20 --                              OldTrapAddr:    LONGINT;
35   21 --                              Jmp:            INTEGER;
35   22 --                              Routine:        Ptr;
35   23 --                          END;
35   24 --
35   25 --          { System heap block record for Tail1Patch }
35   26 --          T1PBlockPtr =   ^T1PBlock;
35   27 --          T1PBlock =      RECORD
35   28 --                              MoveParameter:  LONGINT;
35   29 --                              MoveL:          INTEGER;
35   30 --                              OldTrapAddr:    LONGINT;
35   31 --                              Jmp:            INTEGER;
35   32 --                              Routine:        Ptr;
35   33 --                          END;
35   34 --
35   35 --
35   36 --      {-----------------------------------+
35   37 --      |   Local variables                 |
35   38 --      +-----------------------------------}
35   39 --      VAR
35   40 --          gPatchList:     TrapPatchPtr;           { Head of linked list of patches }
35   41 --
35   42 --      {*****************************************************************************************}
35   43 --      {*      G l o b a l   R o u t i n e s                                                    *}
35   44 --      {*****************************************************************************************}
35   45 --      {$IFC qTrace}{$D+}{$ENDC}       { Tracing code may not be initialized yet }
35   46 --      {$S MAInit}
35   47 --      {-----------------------------------+
35   48 --      |   InitUPatch                      |
35   49 --      +-----------------------------------}
35   50 -- A    PROCEDURE InitUPatch;
35   51 --
35   52 0- A    BEGIN
35   53 --          gPatchList := NIL;      { Initialize the linked list of patches. }
35   54 -0 A    END { InitUPatch };
35   55 --      {$IFC qTrace}{$D++}{$ENDC}
35   56 --
35   57 --
35   58 --      {$IFC qTrace}{$D+}{$ENDC}
35   59 --      {$S MAInit}
35   60 --      {-----------------------------------+
35   61 --      |   AllocBlock                      |
35   62 --      +-----------------------------------}
35   63 -- A    FUNCTION  AllocBlock (blockSize: INTEGER; trapNum: INTEGER;
35   64 --                           VAR thePatch: TrapPatch): Ptr;
35   65 --
35   66 --          { If blockSize is greater than zero, AllocBlock allocates a non-
35   67 --              relocatable block in the system heap, of size blockSize.  It then
35   68 --              fills all fields of thePatch.  If blockSize is zero, then no block
35   69 --              is allocated and AllocBlock fills fields of thePatch. }
35   70 --
35   71 --      VAR
35   72 --          theBlock:       Ptr;
35   73 --
35   74 0- A    BEGIN
35   75 --          IF blockSize > 0 THEN
35   76 --              IF gConfiguration.hasROM128K THEN
35   77 --                  theBlock := NewPtr(blockSize)
35   78 --              ELSE
35   79 --                  theBlock := NewSysPtr(blockSize)
35   80 --          ELSE
35   81 --              theBlock := NIL;
35   82 --
35   83 --          thePatch.jmpPtr := theBlock;
35   84 --          thePatch.oldTrapAddr := NGetTrapAddress(trapNum, GetTrapType(trapNum));
35   85 --          thePatch.trapNum := trapNum;
35   86 --          thePatch.nextPatch := gPatchList;
35   87 --          gPatchList := @thePatch;
35   88 --          AllocBlock := theBlock;
35   89 -0 A    END { AllocBlock };
35   90 --      {$IFC qTrace}{$D++}{$ENDC}
35   91 --
35   92 --
35   93 --      {$IFC qTrace}{$D+}{$ENDC}
35   94 --      {$S MAInit}
35   95 --      {-----------------------------------+
35   96 --      |   PatchTrap                       |
35   97 --      +-----------------------------------}
35   98 -- A    FUNCTION  PatchTrap (VAR thePatch: TrapPatch; theTrapNum: INTEGER;
```

```
35   99 --                          theRoutine: Ptr): OSErr;
35  100 --      VAR
35  101 --          patchBlock:     PTBlockPtr;
35  102 --
35  103 0- A    BEGIN
35  104 --      {$IFC NOT qNeedsROM128K}
35  105 --          IF NOT gConfiguration.hasROM128K THEN
35  106 1-              BEGIN
35  107 --
35  108 --              { Allocate the system heap block, and fill thePatch }
35  109 --              patchBlock := PTBlockPtr(AllocBlock(SIZEOF(PTBlock), theTrapNum, thePatch));
35  110 --
35  111 --              IF patchBlock <> NIL THEN
35  112 2-                  BEGIN
35  113 --                  { Stuff the code into the block }
35  114 --                  WITH patchBlock^ DO
35  115 3-                      BEGIN
35  116 --                      Jmp := $4EF9;                           { JMP       #Routine }
35  117 --                      Routine := theRoutine;
35  118 -3                     END;
35  119 --
35  120 --                  { Set the trap address to the block of code in the system heap }
35  121 --                  NSetTrapAddress(ORD4(patchBlock), theTrapNum, GetTrapType(theTrapNum));
35  122 -2                  END;
35  123 --
35  124 --              PatchTrap := MemError;
35  125 -1              END
35  126 --          ELSE
35  127 --      {$ENDC}
35  128 1-              BEGIN
35  129 --              { On 128K ROMs, setup the patch record and the trap address }
35  130 --              patchBlock := PTBlockPtr(AllocBlock(0, theTrapNum, thePatch));
35  131 --              NSetTrapAddress(ORD4(theRoutine), theTrapNum, GetTrapType(theTrapNum));
35  132 --              PatchTrap := noErr;
35  133 -1              END;
35  134 -0 A    END { PatchTrap };
35  135 --      {$IFC qTrace}{$D++}{$ENDC}
35  136 --
35  137 --
35  138 --      {$IFC qTrace}{$D+}{$ENDC}
35  139 --      {$S MAInit}
35  140 --      {-----------------------------------+
35  141 --      |   TailPatch                       |
35  142 --      +-----------------------------------}
35  143 -- A    FUNCTION  TailPatch (VAR thePatch: TrapPatch; theTrapNum: INTEGER;
35  144 --                          theRoutine: Ptr): OSErr;
35  145 --
35  146 --      VAR
35  147 --          patchBlock:     TPBlockPtr;
35  148 --
35  149 0- A    BEGIN
35  150 --          { Allocate the system heap block, and fill thePatch }
35  151 --          patchBlock := TPBlockPtr(AllocBlock(SIZEOF(TPBlock), theTrapNum, thePatch));
35  152 --
35  153 --          IF patchBlock <> NIL THEN
35  154 1-              BEGIN
35  155 --              { Stuff the code into the block }
35  156 --              WITH patchBlock^ DO
35  157 2-                  BEGIN
35  158 --                  MoveL := $2F3C;                         { MOVE.L    #OldTrapAddr,-(SP)  }
35  159 --                  OldTrapAddr := thePatch.OldTrapAddr;
35  160 --                  Jmp := $4EF9;                           { JMP       #Routine }
35  161 --                  Routine := theRoutine;
35  162 -2                  END;
35  163 --
35  164 --              { Set the trap address to the block of code in the system heap }
35  165 --              NSetTrapAddress(ORD4(patchBlock), theTrapNum, GetTrapType(theTrapNum));
35  166 -1              END;
35  167 --          TailPatch := MemError;
35  168 -0 A    END { TailPatch };
35  169 --      {$IFC qTrace}{$D++}{$ENDC}
35  170 --
35  171 --
35  172 --      {$S MAInit}
35  173 --      {$IFC qTrace}{$D+}{$ENDC}
35  174 --      {-----------------------------------+
35  175 --      |   TailPatch                       |
35  176 --      +-----------------------------------}
35  177 -- A    FUNCTION  TailPatch (VAR thePatch: TrapPatch; theTrapNum: INTEGER;
35  178 --                          theRoutine: Ptr): OSErr;
35  179 --
35  180 --      VAR
35  181 --          patchBlock:     T1PBlockPtr;
35  182 --
35  183 0- A    BEGIN
35  184 --          { Allocate the system heap block, and fill thePatch }
35  185 --          patchBlock := T1PBlockPtr(AllocBlock(SIZEOF(T1PBlock), theTrapNum, thePatch));
35  186 --
35  187 --          IF patchBlock <> NIL THEN
35  188 1-              BEGIN
35  189 --              { Stuff the code into the block }
35  190 --              WITH patchBlock^ DO
35  191 2-                  BEGIN
35  192 --                  MoveParameter := $2F2F0004;             { MOVE.L    4(SP),-(SP) }
35  193 --                  MoveL := $2F3C;                         { MOVE.L    #OldTrapAddr,-(SP) }
35  194 --                  OldTrapAddr := thePatch.OldTrapAddr;
35  195 --                  Jmp := $4EF9;                           { JMP       #Routine }
35  196 --                  Routine := theRoutine;
35  197 -2                  END;
```

```
35  198 --
35  199 --              { Set the trap address to the block of code in the system heap }
35  200 --              NSetTrapAddress(ORD4(patchBlock), theTrapNum, GetTrapType(theTrapNum));
35  201 -1              END;
35  202 --          TaillPatch := MemError;
35  203 -0 A    END { TaillPatch };
35  204 --      {$IFC qTrace}{$D++}{$ENDC}
35  205 --
35  206 --
35  207 --      {$IFC qTrace}{$D+}{$ENDC}
35  208 --      {$S MATerminate}
35  209 --      {-----------------------------------+
35  210 --      |   UnpatchTrap                     |
35  211 --      +-----------------------------------}
35  212 -- A    PROCEDURE UnpatchTrap (VAR thePatch: TrapPatch);
35  213 --
35  214 --      VAR
35  215 --          aPatch:     TrapPatchPtr;
35  216 --
35  217 0- A    BEGIN
35  218 --          { Unlink the patch from the linked list of patches }
35  219 --          IF @thePatch = gPatchList THEN
35  220 --              gPatchList := thePatch.nextPatch
35  221 --          ELSE
35  222 1-              BEGIN
35  223 --              aPatch := gPatchList;
35  224 --              WHILE aPatch^.nextPatch <> @thePatch DO
35  225 2-                  BEGIN
35  226 --                  aPatch := aPatch^.nextPatch;
35  227 --                  IF aPatch = NIL THEN
35  228 --                      { Couldn't find thePatch, so don't try to unpatch it. }
35  229 --                      EXIT(UnpatchTrap);
35  230 -2                  END;
35  231 --              aPatch^.nextPatch := thePatch.nextPatch;
35  232 -1              END;
35  233 --
35  234 --          WITH thePatch DO
35  235 1-              BEGIN
35  236 --              { If the patch allocated a block in the system heap, deallocate it }
35  237 --              IF jmpPtr <> NIL THEN
35  238 2-                  BEGIN
35  239 --                  DisposPtr(jmpPtr);
35  240 --                  jmpPtr := NIL;
35  241 -2                  END;
35  242 --
35  243 --              { Restore the trap address }
35  244 --              NSetTrapAddress(oldTrapAddr, trapNum, GetTrapType(trapNum));
35  245 -1              END;
35  246 --
35  247 -0 A    END { UnpatchTrap };
35  248 --      {$IFC qTrace}{$D++}{$ENDC}
35  249 --
35  250 --
35  251 --      {$IFC qTrace}{$D+}{$ENDC}
35  252 --      {$S MATerminate}
35  253 --      {-----------------------------------+
35  254 --      |   UnpatchAll                      |
35  255 --      +-----------------------------------}
35  256 -- A    PROCEDURE UnpatchAll;
35  257 --
35  258 0- A    BEGIN
35  259 --          WHILE gPatchList <> NIL DO
35  260 --              UnpatchTrap(gPatchList^);
35  261 -0 A    END { UnpatchAll };
35  262 --      {$IFC qTrace}{$D++}{$ENDC}
34  112 --
34  113 --      END.
34  114 --
```

```
36    1 --    {$P}
36    2 --    { UPrinting.p }
36    3 --    { Copyright © 1986-1988 by Apple Computer Inc.  All rights reserved. }
36    4 --
36    5 --    UNIT UPrinting;
36    6 --
36    7 --    INTERFACE
36    8 --
36    9 --    {-----------------------------------+
36   10 --    |   Uses                            |
36   11 --    +-----------------------------------}
36   12 --    USES
36   13 --        {$LOAD MacIntf.LOAD}
36   14 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
36   15 --        {$LOAD}
36   16 --    {$IFC qNeedsROM128K}
36   17 --            PrintTraps,
36   18 --    {$ELSEC}
36   19 --            MacPrint,
36   20 --    {$ENDC}
36   21 --        {$LOAD UMacApp.LOAD}
36   22 --            UMAUtil, UViewCoords, UFailure, UMemory, UMenuSetup, UObject, UList,
36   23 --            UAssociation, UMacApp;
36   24 --
36   25 --
36   26 --    { NOTE: This unit USEs the MacPrint unit, but client's own units and main
36   27 --            program do not need to USE MacPrint unless they make direct calls
36   28 --            to MacPrint code, which is unlikely }
36   29 --
36   30 --
36   31 --    {-----------------------------------+
36   32 --    |   Global constants                |
36   33 --    +-----------------------------------}
36   34 --    CONST
36   35 --
36   36 --      { Command constants }
36   37 --        cChangePrinterStyle =   192;    { Command arising from user's making changes
36   38 --                                            in Page Setup... dialog }
36   39 --        cShowBreaks         =   201;    { "Show Page Breaks" command }
36   40 --
36   41 --      { Printing-related command numbers which apply only when a view with a non-null
36   42 --            PrintHandler is in the target chain }
36   43 --    {$IFC FALSE}
36   44 --        cIncludeFrame       =   202;    { include frame when printing -- mainly for debugging }
36   45 --        cMargins            =   203;    { the Margins... command -- mainly for debugging }
36   46 --        cHeadings           =   204;    { Headings... command -- no such command at present }
36   47 --        cStartPgNum         =   205;    { Starting Page Number... command }
36   48 --        cTopToBottom        =   206;    { number pages top-to-bottom command }
36   49 --        cPrintPageNums      =   207;    { print page numbers at bottom of page }
36   50 --        cShowPageNums       =   211;    { show page numbers in corner of pages --
36   51 --                                            nonstandard; mainly for debugging }
36   52 --    {$ENDC}
36   53 --
36   54 --      { Dialogs }
36   55 --        phSpoolPrintDialog  =   257;    { the dialog shown when printing a file
36   56 --                                            (a misnomer - not just when spooled) }
36   57 --        phFinderPrintDialog =   258;    { the dialog shown when doing Finder Printing;
36   58 --                                            differs from prev. by addition of a
36   59 --                                            'Cancel All Printing' button }
36   60 --
36   61 --      { Alerts }
36   62 --        phNoPages           =   251;    { The "no pages within requested range" alert }
36   63 --
36   64 --      { Dialogs }
36   65 --        phWhichDoc          =   261;    { The "Preparing to print <docname>" dialog
36   66 --                                            used during Finder printing }
36   67 --
36   68 --
36   69 --     { Miscellaneous }
36   70 --        kUsualPages         =   MAXINT; { For methods that take a "pageNumber"
36   71 --                                            argument, this value indicates a
36   72 --                                            request for a general setting rather
36   73 --                                            than a setting customized to one
36   74 --                                            particular page }
36   75 --
36   76 --    {-----------------------------------+
36   77 --    |   Global types                    |
36   78 --    +-----------------------------------}
36   79 --    TYPE
36   80 --
36   81 --        PageAreas =
36   82 --            RECORD
36   83 --                thePaper:       Rect;   { physical page }
36   84 --                theInk:         Rect;   { printable page }
36   85 --                theMargins:     Rect;   { top-left positive; bottom-right negative }
36   86 --                theInterior:    Rect;   { rect into which view-subset is projected }
36   87 --            END;
36   88 --
36   89 --        {-----------------------------------+
36   90 --        |   TStdPrintHandler                |
36   91 --        +-----------------------------------}
36   92 --        TStdPrintHandler = OBJECT(TPrintHandler)
36   93 --
36   94 --            fPageAreas:         PageAreas;  { metrics for printing }
36   95 --            fDocument:          TDocument;  { the document printed by this handler }
36   96 --
36   97 --            fPrintExtent:       VRect;      { the part of the view's extent that is
36   98 --                                                to be printed by this print handler.
```

```
36   99 --                                          The default is the entire extent, but
36  100 --                                          it can be changed with the method
36  101 --                                          SetAreaToPrint. }
36  102 --
36  103 --          fFixedSizePages:    ARRAY [VHSelect] OF BOOLEAN;
36  104 --                                      { Indicates whether pages are fixed size
36  105 --                                          horzontally and vertically. }
36  106 --
36  107 --          fHPrint:            Handle;     { actually of THPrint, a MacPrint type.
36  108 --                                          The Print Record is 120 bytes long,
36  109 --                                          and may be shared among all
36  110 --                                          PrintHandlers in a document or may
36  111 --                                          be unique to each PrintHandler,
36  112 --                                          depending on the flag
36  113 --                                          TDocument.fSharePrintInfo }
36  114 --
36  115 --          fPageStrips:        Point;
36  116 --              { For printing a view which will take more than one page to print,
36  117 --                  the view is thought of as being divided up by vertical and
36  118 --                  horizontal lines ('page breaks') into a checkerboard of 'pages'.
36  119 --                  A vertical column, one page wide, of such pages is called a
36  120 --                  'vertical page strip', while a horizontal row, one page high,
36  121 --                  of such pages is called a 'horizontal page strip'.  The number
36  122 --                  of vertical page strips is represented by fPageStrips.v,
36  123 --                  and the number of horizontal page strips is represented by
36  124 --                  fPageStrips.h
36  125 --
36  126 --                  In a typical Word Processor application, there is only one
36  127 --                  vertical strip of pages, but many horizontal 'strips', each one
36  128 --                  only one page wide.  In a graphical application, there may be
36  129 --                  many vertical as well as horizontal strips of pages.
36  130 --
36  131 --                  fPageStrips is recomputed by method CalcPageStrips. }
36  132 --
36  133 --          fStartPage:         INTEGER;    { the page number for the topleftmost
36  134 --                                          corner of the view.  Normally 1,
36  135 --                                          but can be set to any positive
36  136 --                                          value }
36  137 --
36  138 --          fPrinterDev:        INTEGER;    { MacPrint's device number for printer;
36  139 --                                          use this at Print time for getting
36  140 --                                          font metrics in device coordinates }
36  141 --
36  142 --          fLastCheckedPrinter:
36  143 --                              LONGINT;    { The time that the user's chosen Printer
36  144 --                                          and my own idea of printer metrics
36  145 --                                          were last known to be in harmony }
36  146 --
36  147 --          fLastPrinterName:   STRING[63]; { The name of the current printer driver,
36  148 --                                          at fLastCheckedPrinter. }
36  149 --
36  150 --          fPageDirection:     VHSelect;   { Determines how page numbers are to
36  151 --                                          advance in a large view which is
36  152 --                                          more than one page wide and more
36  153 --                                          than one page high; v means
36  154 --                                          page 2 is BELOW page 1; h means
36  155 --                                          it's to its RIGHT.  Irrelevant if
36  156 --                                          the view is only one page wide or
36  157 --                                          only one page high }
36  158 --
36  159 --
36  160 --          fShowBreaks:        BOOLEAN;    { TRUE iff page breaks should be
36  161 --                                          displayed at the moment }
36  162 --
36  163 --          fFinderSetup:       BOOLEAN;    { whether, during Finder Printing, the
36  164 --                                          'Page Setup' dialog should be
36  165 --                                          presented to the user }
36  166 --
36  167 --          fFinderJobDialog:   BOOLEAN;    { whether, during Finder Printing, the
36  168 --                                          Print Job dialog should be
36  169 --                                          presented to the user (making it
36  170 --                                          seem like "Print...") or not
36  171 --                                          (making it seem like "Print One") }
36  172 --
36  173 --          fSquareDots:        BOOLEAN;    { if TRUE, then in the initial instance,
36  174 --                                          use 72 x 72 resolution by default
36  175 --                                          on ImageWriter; if FALSE, then full
36  176 --                                          80 x 72 is used.  This flag has no
36  177 --                                          significance if the default printer
36  178 --                                          at the time of creation of the
36  179 --                                          printHandler is not an ImageWriter }
36  180 --          fMinimalMargins:    BOOLEAN;    { If TRUE, page margins are maintained
36  181 --                                          such that exactly the complete
36  182 --                                          printable area of the page is used. }
36  183 --
36  184 --          fLastStrip:         Point;      { The last strip printed. }
36  185 --
36  186 --          fLastBreak:         VPoint;     { The coordinates of the last page break. }
36  187 --
36  188 --          fViewedRect:        VRect;      { The part of the view shown on the
36  189 --                                          "current" page.  Set by SetPage. }
36  190 --
36  191 --      { Initialization/Termination }
36  192 --          PROCEDURE TStdPrintHandler.IStdPrintHandler(itsDocument: TDocument; itsView: TView;
36  193 --                      itsSquareDots, itsHFixedSize, itsVFixedSize: BOOLEAN);
36  194 --              { Initializes the printHandler and installs it in the view; if
36  195 --                  'itsSquareDots' is TRUE, then a default print record for the
36  196 --                  ImageWriter world will use square  dots ('tall adjusted') rather
36  197 --                  than rectangular ones; this sacrifices a bit of resolution but
```

```
36  198 --                  ensures that graphics will appear undistorted on the printed
36  199 --                  page }
36  200 --
36  201 --          PROCEDURE TStdPrintHandler.Free; OVERRIDE;
36  202 --              { If relevant, frees the PrintInfo record before freeing SELF }
36  203 --
36  204 --          PROCEDURE TStdPrintHandler.Terminate; OVERRIDE;
36  205 --              { Called automatically by MacApp (for the printHandler installed in
36  206 --                  the global variable "gPrintHandler" in UMacApp) at the end of
36  207 --                  processing to shut down MacPrint code. }
36  208 --
36  209 --      { Page Strips }
36  210 --
36  211 --          PROCEDURE TStdPrintHandler.CalcPageStrips (VAR pageStrips: Point); OVERRIDE;
36  212 --              { Recalculate the number of strips of pages in each dimension }
36  213 --
36  214 --          PROCEDURE TStdPrintHandler.CalcViewPerPage(VAR amtPerPage: VPoint); OVERRIDE;
36  215 --              { Decide how much view (h and v) normally gets mapped into page
36  216 --                  interiors }
36  217 --
36  218 --          FUNCTION  TStdPrintHandler.BreakFollowing(vhs: VHSelect;
36  219 --                          prevBreak: VCoordinate; VAR automatic: BOOLEAN
36  220 --                          ): VCoordinate; OVERRIDE;
36  221 --            { Returns the location of the page break which follows the page break
36  222 --              located at 'prevBreak', in direction vhs; returns automatic = TRUE
36  223 --              if the page-break is thought of as an 'automatic' rather than a
36  224 --              'manual' (user-specified) one.  Note that page-breaks in dimension
36  225 --              'v' are drawn as vertical lines, those in dimension 'h' as
36  226 --              horizontal lines.  Generic support at present is only provided for
36  227 --              'automatic' page breaks; the 'automatic' parameter in
36  228 --              BreakFollowing is a hook to allow clients to support manual page
36  229 --              breaks as well }
36  230 --
36  231 --          PROCEDURE TStdPrintHandler.EachBreak(vhs: VHSelect; includeLast: BOOLEAN;
36  232 --                  FUNCTION DoToBreak(loc: VCoordinate; automatic: BOOLEAN): BOOLEAN);
36  233 --            { Iterates through all page-breaks in the dimension given by vhs, or
36  234 --              until DoToBreak returns true; the trailing edge of the view (i.e.,
36  235 --              right or bottom edge) is given as the final break if 'includeLast'
36  236 --              is TRUE. The function 'DoToBreak' is called once for each page break
36  237 --              in the chosen dimension }
36  238 --
36  239 --          PROCEDURE TStdPrintHandler.GetBreakCoord(vhs: VHSelect;
36  240 --                  whichBreak: INTEGER; VAR loc: VCoordinate);
36  241 --            { Returns, in 'loc', the coordinate of the 'whichBreak-th' page break
36  242 --              in dimension vhs.  For example, GetBreakCoord(v, 5, loc) will return,
36  243 --              in 'loc', the horizontal coordinate of the vertical line
36  244 --              (page-break) which marks the boundary the area of the view which
36  245 --              will be mapped into the 4th vertical strip of pages and the area
36  246 --              of the view which will be mapped into the 5th vertical strip of
36  247 --              pages }
36  248 --
36  249 --          FUNCTION TStdPrintHandler.PageToStrip (pageNumber: INTEGER): Point;
36  250 --              { Returns the horizontal and vertical strip for the given page. }
36  251 --
36  252 --          FUNCTION TStdPrintHandler.StripToPage (hStrip, vStrip: INTEGER): INTEGER;
36  253 --              { Returns the page number for the given page strip. }
36  254 --
36  255 --          FUNCTION  TStdPrintHandler.PointToPageStrip(pointInView: VPoint): Point;
36  256 --              { Returns the page strip for a given point in view space. }
36  257 --
36  258 --
36  259 --      { General Utility }
36  260 --
36  261 --          PROCEDURE TStdPrintHandler.GetDocName(VAR docName: Str255);
36  262 --              { Returns the name of the document being printed, for use in
36  263 --                  the print dialog. }
36  264 --
36  265 --          PROCEDURE TStdPrintHandler.DoInMacPrint(PROCEDURE WhatToDo);
36  266 --            { If printing code is available, open the Print Driver, then call
36  267 --              procedure "WhatToDo", then close the Print Driver.  If printing
36  268 --              code not available, does nothing }
36  269 --
36  270 --          PROCEDURE TStdPrintHandler.OpenPrintShop;
36  271 --              { Attempts to open the print shop by calling PrOpen.  Signals
36  272 --                  failure if an error occurs.  Called from DoInMacPrint. }
36  273 --
36  274 --          PROCEDURE TStdPrintHandler.ClosePrintShop;
36  275 --              { Closes the print shop by calling PrClose.  DOES NOT signal
36  276 --                  failure if an error occurs, only reports it to the debug
36  277 --                  window. }
36  278 --
36  279 --          FUNCTION  TStdPrintHandler.MaxPageNumber: INTEGER; OVERRIDE;
36  280 --              { Returns the highest page number which would be printed if the user
36  281 --                  requested that "all pages" be printed }
36  282 --
36  283 --
36  284 --      { Actual Printing }
36  285 --          { Basic flow of control at print time:
36  286 --              Print
36  287 --                  CheckPrinter
36  288 --                  For each page:
36  289 --                    FocusOnBorder
36  290 --                    DrawPageInterior
36  291 --                    FocusOnBorder
36  292 --                    AdornPage }
36  293 --
36  294 --          PROCEDURE TStdPrintHandler.AdornPage;
36  295 --             { Draw things on page # 'aPageNumber' which do not arise from the
36  296 --                  View but rather from page considerations; examples: Headers
```

```
36  297 --                  and Footers; page numbers; frames drawn around inset text;
36  298 --                  row and column headers for spreadsheets. }
36  299 --
36  300 --          PROCEDURE TStdPrintHandler.ChooseSpoolFile(VAR spoolFileName: Str255;
36  301 --                  VAR spoolVRefNum: INTEGER; VAR pagesPerSubjob: INTEGER);
36  302 --              { Select the filename and volume to use for spooling; default chooses
36  303 --                  empty string on vol 0, which results in MacPrint's automatically
36  304 --                  choosing the local Boot volume }
36  305 --
36  306 --          PROCEDURE TStdPrintHandler.DrawPageInterior;
36  307 --              { Make the QuickDraw calls which will result in the 'Interior' of
36  308 --                  page # 'aPageNumber' being drawn.  The 'Interior' is those
36  309 --                  things on the page which arise from drawing the View associated
36  310 --                  with the PrintHandler }
36  311 --
36  312 --          PROCEDURE TStdPrintHandler.FocusOnBorder;
36  313 --              { Set up GrafPort, clipping, etc., prior to calling AdornPage }
36  314 --
36  315 --          PROCEDURE TStdPrintHandler.FocusOnInterior; OVERRIDE;
36  316 --              { Set up the GrafPort, clipping, etc., prior to calling
36  317 --                  DrawPageInterior }
36  318 --
36  319 --          PROCEDURE TStdPrintHandler.LocatePageInterior(pageNumber: INTEGER;
36  320 --                  VAR loc: Point); OVERRIDE;
36  321 --              { Determine where to locate the top-left corner of the Interior of
36  322 --                  the given page nuamber }
36  323 --
36  324 --          FUNCTION TStdPrintHandler.OneSubJob(
36  325 --                  subjobFirstPage, subjobLastPage: INTEGER;
36  326 --                  justSpool: BOOLEAN; partialJob: BOOLEAN;
36  327 --                  VAR ranOutOfSpace: BOOLEAN; VAR lastPageTried: INTEGER;
36  328 --                  VAR proceed: BOOLEAN): TCommand;
36  329 --              { Processes one subjob for printing.  Where appropriate, a print job
36  330 --                  is divided automatically into one or more 'subjobs', so that a
36  331 --                  long document can be printed even if there is very limited
36  332 --                  spool space available on the disk.  This is a generalization
36  333 --                  of the 'spool-a-page/print-a-page' technique.  TStdPrintHandler
36  334 --                  will use a subjob size as large as can be handled by current
36  335 --                  conditions, and will only fall back to more subjobs, each of
36  336 --                  smaller size, if it experiences problems running with a larger
36  337 --                  subjob size }
36  338 --
36  339 --          FUNCTION TStdPrintHandler.Print(itsCmdNumber: CmdNumber; VAR proceed: BOOLEAN): TCommand; OVERRIDE;
36  340 --              { Carries out printing for the print handler. }
36  341 --
36  342 --          PROCEDURE TStdPrintHandler.PrintPage(aPageNumber: INTEGER);
36  343 --              { Print the indicated page }
36  344 --
36  345 --          PROCEDURE TStdPrintHandler.SetPage(aPageNumber: INTEGER);
36  346 --              { Dispatches to TView.DoSetInterior and TView.DoSetPageOffset }
36  347 --
36  348 --          PROCEDURE TStdPrintHandler.SetPageInterior(pageNumber: INTEGER); OVERRIDE;
36  349 --              { Responsible for installing the correct values into
36  350 --                fPageAreas.theInterior, representing the 'interior' of the page
36  351 --                in View coordinates }
36  352 --
36  353 --          PROCEDURE TStdPrintHandler.SetPageOffset(coord: VPoint); OVERRIDE;
36  354 --              { Given the view-coordinate of the top-left-most point of the view
36  355 --                  which is mapped into the interior of this page, calculate and
36  356 --                  set the offset for the print code. }
36  357 --
36  358 --
36  359 --      { Formatting for printing and other Setup methods }
36  360 --
36  361 --          PROCEDURE TStdPrintHandler.CheckPrinter; OVERRIDE;
36  362 --              { Check to see if user has changed printer specifications; if so,
36  363 --                  react }
36  364 --
36  365 --          PROCEDURE TStdPrintHandler.InstallMargins(newMargins: Rect;
36  366 --                  areMinimalMargins: BOOLEAN);
36  367 --              { Install the (possibly new) margins specified; if
36  368 --                  'areMinimalMargins' is TRUE, then the minimal-margins scheme is
36  369 --                  used; in this scheme, margins are always maintained at exactly
36  370 --                  the size necessary to have the page Interior coincide exactly
36  371 --                  with the printable area of the page }
36  372 --
36  373 --          PROCEDURE TStdPrintHandler.PrinterChanged; OVERRIDE;
36  374 --              { After printer specifications have changed, absorb the new metrics
36  375 --                  and reformat the view if necessary }
36  376 --
36  377 --          PROCEDURE TStdPrintHandler.RedoPageBreaks; OVERRIDE;
36  378 --              { Recalculate the page breaks }
36  379 --
36  380 --          PROCEDURE TStdPrintHandler.Reset; OVERRIDE;
36  381 --              { Reset print record to default }
36  382 --
36  383 --          PROCEDURE TStdPrintHandler.SetDefaultPrintInfo;
36  384 --              { Create a PrintInfo record and initialize it to default values }
36  385 --
36  386 --          PROCEDURE TStdPrintHandler.SetPrintExtent;
36  387 --              { Sets the part of the print handler's view that is printed
36  388 --                  by this print handler, by calling the view's GetPrintExtent. }
36  389 --
36  390 --          PROCEDURE TStdPrintHandler.SetMargins;
36  391 --              { Install the desired page margins }
36  392 --
36  393 --          PROCEDURE TStdPrintHandler.ValidatePrintRecord(VAR didChange: BOOLEAN);
36  394 --              { Make certain that the current Print Record is compatible with the
36  395 --                  user's current printer choice; 'didChange' is set to TRUE
```

```
36  396 --                      iff the kind of printer changed }
36  397 --
36  398 --
36  399 --          { Menu Commands }
36  400 --
36  401 --              FUNCTION  TStdPrintHandler.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
36  402 --                  { Handles 'show page breaks', among others }
36  403 --
36  404 --              PROCEDURE TStdPrintHandler.DoSetupMenus; OVERRIDE;
36  405 --                  { Sets up menus for 'show page breaks' and company }
36  406 --
36  407 --
36  408 --          { Finder Printing }
36  409 --
36  410 --              FUNCTION TStdPrintHandler.SetupForFinder: BOOLEAN; OVERRIDE;
36  411 --                  { Sets up the print record for finder printing.  If fFinderPageSetup
36  412 --                      is true, then PosePageSetupDialog is called.  If fFinderJobDialog
36  413 --                      is true, then PoseJobDialog is called.  Otherwise, prJobMerge is
36  414 --                      called to merge the job record from the }
36  415 --
36  416 --              FUNCTION SetupPrintOne: BOOLEAN;
36  417 --                  { Sets up the print record for a "print one" type of operation,
36  418 --                      one where the entire document or view is printed and no
36  419 --                      job dialog takes place.  Returns true if we should proceed. }
36  420 --
36  421 --
36  422 --          { Screen Feedback }
36  423 --
36  424 --              PROCEDURE TStdPrintHandler.DrawPrintFeedback(area: Rect); OVERRIDE;
36  425 --                  { Gateway to all screen feedback related to pagination issues }
36  426 --
36  427 --              PROCEDURE TStdPrintHandler.DrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
36  428 --                                                        loc: VCoordinate; automatic: BOOLEAN); OVERRIDE;
36  429 --                  { Draws a page break on the screen. }
36  430 --
36  431 --              PROCEDURE TStdPrintHandler.InvalPageFeedback;
36  432 --                  { Invalidates (forces redraw) all page-related screen feedback }
36  433 --
36  434 --              FUNCTION TStdPrintHandler.ShowsOnScreen: BOOLEAN;
36  435 --                  { Returns TRUE only if the printHandler has an fView which is
36  436 --                      in an open window. }
36  437 --
36  438 --
36  439 --          { Dialogs relating to printing }
36  440 --
36  441 --              FUNCTION TStdPrintHandler.PoseJobDialog: BOOLEAN;
36  442 --                  { Pose the 'Print Job' dialog }
36  443 --
36  444 --              FUNCTION TStdPrintHandler.PosePageSetupDialog(
36  445 --                      VAR proceed: BOOLEAN; isUndoable: BOOLEAN): TCommand;
36  446 --                  { Undertake the 'Page Setup' dialog; return a command object to be
36  447 --                      performed if relevant }
36  448 --
36  449 --              PROCEDURE TStdPrintHandler.PosePrintDialog;
36  450 --                  { Put up the 'Printing in Progress' dialog }
36  451 --
36  452 --              PROCEDURE TStdPrintHandler.ShowDocBeingPrinted (entering: BOOLEAN);
36  453 --                  { Put up (or remove) a window whose title is the document's title,
36  454 --                      because the Print Manager gets the document title from the
36  455 --                      front window. }
36  456 --
36  457 --              PROCEDURE TStdPrintHandler.BanishPrintDialog;
36  458 --
36  459 --          { Debugging }
36  460 --
36  461 --      {$IFC qDebug}
36  462 --              PROCEDURE TStdPrintHandler.IdentifySoftware; OVERRIDE;
36  463 --                  { Shows the version of the Printing Unit being used; display is in
36  464 --                      the Debug window }
36  465 --
36  466 --              PROCEDURE TStdPrintHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
36  467 --                                                      fieldAddr: Ptr;
36  468 --                                                      fieldType: INTEGER)); OVERRIDE;
36  469 --      {$ENDC}
36  470 --              END;
36  471 --
36  472 --
36  473 --          {-----------------------------------+
36  474 --          |   TPrintStyleChangeCommand        |
36  475 --          +-----------------------------------}
36  476 --          TPrintStyleChangeCommand = OBJECT(TCommand)
36  477 --
36  478 --              { This command is used to allow changes made by the user in the
36  479 --                  'Page Setup...' dialog to be undoable }
36  480 --
36  481 --          { Fields }
36  482 --
36  483 --              fStdPrintHandler:   TStdPrintHandler;   { Associated print handler }
36  484 --
36  485 --              fOldHPrint:         Handle;             { The old print record, before the change }
36  486 --              fNewHPrint:         Handle;             { The new print record, after the change }
36  487 --
36  488 --
36  489 --          { Initialization and Termination }
36  490 --
36  491 --              PROCEDURE TPrintStyleChangeCommand.IPrintStyleChangeCommand(
36  492 --                      itsPrintHandler: TStdPrintHandler);
36  493 --              PROCEDURE TPrintStyleChangeCommand.Free; OVERRIDE;
36  494 --
```

```
36  495 --
36  496 --          { Command Execution }
36  497 --
36  498 --              PROCEDURE TPrintStyleChangeCommand.DoIt; OVERRIDE;
36  499 --              PROCEDURE TPrintStyleChangeCommand.UndoIt; OVERRIDE;
36  500 --              PROCEDURE TPrintStyleChangeCommand.RedoIt; OVERRIDE;
36  501 --      {$IFC qDebug}
36  502 --          { Debugging }
36  503 --              PROCEDURE TPrintStyleChangeCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
36  504 --                                                          fieldAddr: Ptr;
36  505 --                                                          fieldType: INTEGER)); OVERRIDE;
36  506 --
36  507 --      {$ENDC}
36  508 --              END;
36  509 --
36  510 --      {-----------------------------------+
36  511 --      |   Global variables                |
36  512 --      +-----------------------------------}
36  513 --      VAR
36  514 --          gBreaksPenState:    PenState;   { penState to used to draw page
36  515 --                                            breaks }
36  516 --          gCancelAllPrinting: BOOLEAN;    { set to TRUE if during Finder Printing user
36  517 --                                            selects "Cancel All" }
36  518 --          gStdPageMargins:    Rect;       { "Standard" left, top, right, bottom page
36  519 --                                            margins, in screen resolution -
36  520 --                                            one inch all round by default }
36  521 --          gPrintDialog:       DialogPtr;  { Used for all printing-related dialogs }
36  522 --
36  523 --          gFinderHPrint:      Handle;     { Used for finder printing, to do PrJobMerge
36  524 --                                            when printing more than one document. }
36  525 --
36  526 --      {-----------------------------------+
36  527 --      |   Global routines                 |
36  528 --      +-----------------------------------}
36  529 --        { Initialization }
36  530 --
36  531 --          PROCEDURE InitPrinting;
36  532 --              { Call once, at application start-up time.  You can interrogate global
36  533 --                  variable 'gCouldPrint' at any time to determine whether printing
36  534 --                  code is actually accessible }
36  535 --
36  536 --
36  537 --        { NOTE: None of the remaining global procedures in this unit are intended to
36  538 --              be called directly by the client; they are in the Interface only for
36  539 --              use of TPrintHandler subclasses }
36  540 --
36  541 --          PROCEDURE EmpowerChooser(allowIt: BOOLEAN);
36  542 --              { Set or clear the bit that controls access to the Choose Printer desk
36  543 --                  accessory }
36  544 --
36  545 --
36  546 --      IMPLEMENTATION
36  547 --
36  548 --      {$I UPrinting.incl.p}
```

```
37    1 --     {$P}
37    2 --     { UPrinting.incl.p }
37    3 --     { Copyright © 1986-1988 by Apple Computer, Inc.  All rights reserved. }
37    4 --
37    5 --     (*
37    6 --         Segmentation strategy:
37    7 --
37    8 --         PrintRes                Resident
37    9 --         PrintMain               Resident and used at initialization
37   10 --         PrintActual             Used only during Actual Printing (imaging/spooling)
37   11 --         PrintImage              Used during imaging only
37   12 --         PrintSpool              Used only during printing of a Spool File
37   13 --         PrintDebug              Debugging code
37   14 --         PrintFinder             Code only ever accessed from Finder Printing
37   15 --         PrintInit               One-time Initialization Code
37   16 --         PrintOpen               Code accessed when opening new document
37   17 --         PrintNonRes             General non-resident code
37   18 --         PrintDoCommand          Code for Page Setup command
37   19 --         PrintTerminate          One-time-only code, called only at Termination time
37   20 --
37   21 --     *)
37   22 --
37   23 --
37   24 --     {-----------------------------------+
37   25 --     |   Local constants                 |
37   26 --     +-----------------------------------}
37   27 --     CONST
37   28 --
37   29 --         kNeverInitialized = MAXINT; { Pseudo-value for fPrinterDev field of TStdPrintHandler; used to
37   30 --                                       trigger an initial PrinterChanged call }
37   31 --
37   32 --         kPrintDriverName = $E000;   { STR resource in System file which contains
37   33 --                                       name of the current print driver }
37   34 --
37   35 --     {-----------------------------------+
37   36 --     |   Local variables                 |
37   37 --     +-----------------------------------}
37   38 --     VAR
37   39 --         gPPrPort:       TPPrPort;   { Printer Port; used during printing }
37   40 --
37   41 --
37   42 --
37   43 --     (*****************************************************************************************)
37   44 --     (*      G l o b a l   R o u t i n e s                                                    *)
37   45 --     (*****************************************************************************************)
37   46 --     {$S PrintActual}
37   47 --     {-----------------------------------+
37   48 --     |   CheckButton                     |
37   49 --     +-----------------------------------}
37   50 -- A   PROCEDURE CheckButton;
37   51 --     { This procedure handles events in the 'Print Dialogs' which give the user
37   52 --     'Cancel', and, in the case of printing from the Finder, 'Cancel All Printing. }
37   53 --     CONST
37   54 --         myDlgMask = mDownMask + mUpMask + updateMask;
37   55 --     VAR
37   56 --         box:                Rect;
37   57 --         savedPort:          GrafPtr;
37   58 --         aDialog:            DialogPtr;
37   59 --         anEvent:            EventRecord;
37   60 --         item:               INTEGER;
37   61 --         itemType:           INTEGER;
37   62 --         b:                  BOOLEAN;
37   63 --         printState:         BOOLEAN;
37   64 --         savedPrintHandler:  TPrintHandler;
37   65 0- A   BEGIN
37   66 --         GetPort(savedPort);     { Save MacPrint's GrafPort }
37   67 --         IF GetNextEvent(myDlgMask, anEvent) THEN
37   68 1-             BEGIN
37   69 --             { Workaround for LaserWriter driver bug:  it tries to set GhostWindow
37   70 --                 so that its status window is invisible.  Unfortunately, it doesn't
37   71 --                 always set it, so IsDialogEvent returns FALSE (the status window
37   72 --                 is frontmost).  If ours isn't already in front, force it there. }
37   73 --             IF gPrintDialog <> FrontWindow THEN
37   74 --                 gApplication.SelectWMgrWindow(gPrintDialog);
37   75 --             IF IsDialogEvent(anEvent) THEN
37   76 2-                 BEGIN
37   77 --                 b := DialogSelect(anEvent, aDialog, item);
37   78 --                 IF b AND (aDialog = gPrintDialog) THEN
37   79 3-                     CASE item OF
37   80 --                         1:  PrSetError(iPrAbort);           { Cancel }
37   81 --
37   82 4-                         2:  BEGIN                           { Cancel All Printing }
37   83 --                             PrSetError(iPrAbort);
37   84 --                             gCancelAllPrinting := TRUE;
37   85 -4                             END;
37   86 --
37   87 -3                         END;{ case }
37   88 -2                 END;
37   89 --     {$IFC FALSE}        {??? This doesn't seem to work }
37   90 --             ELSE    { not a dialog event - check for updates and activates }
37   91 --                 IF (anEvent.what = updateEvt) OR (anEvent.what = activateEvt) THEN
37   92 2-                     BEGIN
37   93 --                     printState := gPrinting;
37   94 --                     gPrinting := FALSE;
37   95 --                     savedPrintHandler := gCurrPrintHandler;
37   96 --                     gCurrPrintHandler := NIL;
37   97 --                     gApplication.HandleEvent(anEvent);
37   98 --                     gPrinting := printState;
```

```
37   99 --                    gCurrPrintHandler := savedPrintHandler;
37  100 -2                   END;
37  101 --      {$ENDC}
37  102 -1              END;
37  103 --          SetPort(savedPort);      { Restore MacPrint's Grafport }
37  104 -0 A    END;
37  105 --
37  106 --
37  107 --      {$S PrintActual}
37  108 --      {-----------------------------------+
37  109 --      |   ChkPrintErr                     |
37  110 --      +-----------------------------------}
37  111 -- A    PROCEDURE ChkPrintErr(VAR err: OSErr; VAR proceed: BOOLEAN; VAR ranOutOfSpace: BOOLEAN);
37  112 --      { Checks for a MacPrint error, but only if 'proceed' is TRUE on entry; specially
37  113 --          handles one particular error (sets return parameter 'ranOutOfSpace'
37  114 --          if error code indicates disk-space problem while saving a spool file). }
37  115 0- A    BEGIN
37  116 --          IF proceed THEN
37  117 1-              BEGIN
37  118 --              err := PrError;
37  119 --              IF err <> noErr THEN
37  120 2-                  BEGIN
37  121 --      {$IFC qDebug}
37  122 --                  IF gDebugPrinting THEN
37  123 --                      WriteLn('Error from PrError is ', err:1);
37  124 --      {$ENDC}
37  125 --                  proceed := FALSE;
37  126 --                  IF err =  -1 { iPrSavPFil } THEN
37  127 --                      ranOutOfSpace := TRUE;
37  128 -2                  END;
37  129 -1              END;
37  130 -0 A    END;
37  131 --
37  132 --
37  133 --      {$S PrintMain}  { Not the best place, but called at both Init and Term }
37  134 --      {-----------------------------------+
37  135 --      |   EmpowerChooser                  |
37  136 --      +-----------------------------------}
37  137 -- A    PROCEDURE EmpowerChooser(allowIt: BOOLEAN);  { Poke low-memory to enable or disable access to 'Choose Printer'
37  138 --                                                        Desk Acceessory }
37  139 --      CONST
37  140 --          ChooserByte =   $946;
37  141 --
37  142 0- A    BEGIN
37  143 --          IF allowIt THEN
37  144 --              BitSet(Ptr(ChooserByte), 0)   { set bit 7 of $946 to 1 to enable, to 0 to disable }
37  145 --          ELSE
37  146 --              BitClr(Ptr(ChooserByte), 0)
37  147 -0 A    END;
37  148 --
37  149 --
37  150 --      {$S PrintRes}
37  151 --      {-----------------------------------+
37  152 --      |   GetDriverName                   |
37  153 --      +-----------------------------------}
37  154 -- A    PROCEDURE GetDriverName (VAR driverName: Str255);
37  155 --      VAR
37  156 --          driverHandle:   StringHandle;
37  157 0- A    BEGIN
37  158 --          driverHandle := GetString(kPrintDriverName);    { Get current driver }
37  159 --
37  160 --          { Be a little cautious, in case we accidentally pick up something
37  161 --              which is not what we expect, or we can't find it. }
37  162 --          IF (driverHandle <> NIL) & (LENGTH(driverHandle^^) < 64) THEN
37  163 --              driverName := driverHandle^^
37  164 --          ELSE
37  165 --              driverName := '';
37  166 -0 A    END;
37  167 --
37  168 --
37  169 --      {$S PrintInit}
37  170 --      {-----------------------------------+
37  171 --      |  RealInitPrinting                 |
37  172 --      +-----------------------------------}
37  173 -- A    PROCEDURE RealInitPrinting;
37  174 --      VAR
37  175 --          err:                OSErr;
37  176 --          aStdPrintHandler:   TStdPrintHandler;
37  177 0- A    BEGIN
37  178 --          gCancelAllPrinting := FALSE;
37  179 --          gPrintDialog := NIL;
37  180 --          gFinderHPrint := NIL;
37  181 --
37  182 --          SetRect(gStdPageMargins, 72, 72, -72, -72);   { 1" margins std default }
37  183 --
37  184 --          WITH gBreaksPenState DO
37  185 1-              BEGIN
37  186 --              pnLoc := Point(0);
37  187 --              pnSize := Point($00020002);
37  188 --              pnMode := patCopy;
37  189 --              StuffHex(@pnPat, 'CC663399CC663399');
37  190 -1              END;
37  191 --
37  192 --          gCouldPrint := TRUE;
37  193 --
37  194 --          IF NOT gChooserOK THEN  { disable printer-change in the chooser }
37  195 --              EmpowerChooser(FALSE);
37  196 --
37  197 --          IF gPrintHandler = gNullPrintHandler THEN
```

```
37  198 1-              BEGIN                       { Install a StdPrintHandler in UMacApp global variable gPrintHandler }
37  199 --              New(aStdPrintHandler);
37  200 --              FailNIL(aStdPrintHandler);
37  201 --              aStdPrintHandler.IStdPrintHandler(NIL, NIL, TRUE, TRUE, TRUE);
37  202 --              aStdPrintHandler.fFinderJobDialog := TRUE;
37  203 --              gPrintHandler := aStdPrintHandler;
37  204 -1              END;
37  205 --
37  206 -0 A    END;
37  207 --
37  208 --
37  209 --      {$S Main}
37  210 --      { Initialize the printing unit.  Call exactly once, after calling
37  211 --          InitToolBox, and after stuffing gChooserOK to FALSE if you do not wish
37  212 --          to allow your users ever to use 'Choose Printer' while within your
37  213 --          application }
37  214 --      {-----------------------------------+
37  215 --      |   InitPrinting                    |
37  216 --      +-----------------------------------}
37  217 -- A    PROCEDURE InitPrinting;
37  218 0- A    BEGIN
37  219 --          RealInitPrinting;
37  220 --          UnloadAllSegments;
37  221 -0 A    END;
37  222 --
37  223 --
37  224 --      {$S PrintSpool}
37  225 --      {-----------------------------------+
37  226 --      |   PrintSpoolFile                  |
37  227 --      +-----------------------------------}
37  228 -- A    PROCEDURE PrintSpoolFile(anHPrint: Handle; VAR err: OSErr; VAR proceed: BOOLEAN);
37  229 --      VAR
37  230 --          prStatus:   TPrStatus;
37  231 --          b:          BOOLEAN;
37  232 0- A    BEGIN
37  233 --          proceed := TRUE;
37  234 --          PRPicFile(THPrint(anHPrint), NIL, NIL, NIL, prStatus);
37  235 --          ChkPrintErr(err, proceed, b);
37  236 -0 A    END;
37  237 --
37  238 --      (*****************************************************************************************)
37  239 --      (*      T S t d P r i n t H a n d l e r                                                   *)
37  240 --      (*****************************************************************************************)
37  241 --      {$S PrintOpen}
37  242 --      {-----------------------------------+
37  243 --      |   IStdPrintHandler                |
37  244 --      +-----------------------------------}
37  245 -- A    PROCEDURE TStdPrintHandler.IStdPrintHandler(itsDocument: TDocument; itsView: TView;
37  246 --                                                  itsSquareDots, itsHFixedSize, itsVFixedSize: BOOLEAN);
37  247 --      VAR
37  248 --          fi:             FailInfo;
37  249 --
37  250 --          {-------------------------------+
37  251 --          |   HdlInitFailed               |
37  252 --          +-------------------------------}
37  253 -- B        PROCEDURE HdlInitFailed(error: OSErr; message: LONGINT);
37  254 0- B        BEGIN
37  255 --              Free;
37  256 -0 B        END;
37  257 --
37  258 0- A    BEGIN
37  259 --          fHPrint := NIL;
37  260 --          IPrintHandler(itsView);
37  261 --          fDocument := itsDocument;
37  262 --
37  263 --          CatchFailures(fi, HdlInitFailed);
37  264 --
37  265 --          fStartPage := 1;
37  266 --          fPageDirection := v;                    { Page 2 is below page 1, etc. }
37  267 --          fShowBreaks := FALSE;
37  268 --          fFixedSizePages[h] := itsHFixedSize;
37  269 --          fFixedSizePages[v] := itsVFixedSize;
37  270 --
37  271 --          fPrinterDev := kNeverInitialized;
37  272 --
37  273 --          fFinderSetup := FALSE;
37  274 --          fFinderJobDialog := FALSE;
37  275 --      {$H-}
37  276 --          fSquareDots := itsSquareDots;
37  277 --          SetPt(fLastStrip, MAXINT, MAXINT);
37  278 --          fLastBreak := gZeroVPt;
37  279 --      {$H+}
37  280 --
37  281 --          fPageAreas.theMargins := gStdPageMargins;
37  282 --          fEffectiveDeviceRes.h := 72;
37  283 --          fEffectiveDeviceRes.v := 72;
37  284 --          fMinimalMargins := FALSE;
37  285 --
37  286 --          fLastCheckedPrinter := 0;
37  287 --          IF itsView <> NIL THEN
37  288 1-              BEGIN
37  289 --              { allocates a handle for the print-info, and sets my field fHPrint
37  290 --                  accordingly }
37  291 --              SetDefaultPrintInfo;
37  292 --              fView.AttachPrintHandler(SELF);
37  293 -1              END
37  294 --          ELSE
37  295 --              { We leave fView and fHPrint NIL.  This must be a global print
37  296 --                  handler. }
```

```
37  297 --              itsDocument := NIL;
37  298 --
37  299 --          IF itsDocument <> NIL THEN
37  300 1-              BEGIN
37  301 --              IF itsDocument.fDocPrintHandler = NIL THEN
37  302 --                  itsDocument.fDocPrintHandler := SELF;
37  303 --              IF itsDocument.fPrintInfo = NIL THEN
37  304 --                  IF itsDocument.fSharePrintInfo THEN
37  305 --                      itsDocument.fPrintInfo := fHPrint;
37  306 -1              END;
37  307 --
37  308 --          Success(fi);
37  309 -0 A    END;
37  310 --
37  311 --
37  312 --      {$S PrintClose}
37  313 --      {-----------------------------------+
37  314 --      |   Free                            |
37  315 --      +-----------------------------------}
37  316 -- A    PROCEDURE TStdPrintHandler.Free; OVERRIDE;
37  317 --      VAR
37  318 --          dontDispose:    BOOLEAN;
37  319 --          itsDocument:    TDocument;
37  320 0- A    BEGIN
37  321 --          dontDispose := fView <> NIL;
37  322 --          IF dontDispose THEN
37  323 1-              BEGIN
37  324 --              IF fView.fPrintHandler = SELF THEN
37  325 --                  fView.AttachPrintHandler(gNullPrintHandler);
37  326 --              itsDocument := fDocument;
37  327 --              dontDispose := itsDocument <> NIL;
37  328 --              IF dontDispose THEN
37  329 2-                  BEGIN
37  330 --                  IF itsDocument.fDocPrintHandler = SELF THEN
37  331 --                      itsDocument.fDocPrintHandler := NIL;
37  332 --                  dontDispose := itsDocument.fSharePrintInfo;
37  333 -2                  END;
37  334 --              IF dontDispose THEN
37  335 --                  dontDispose := itsDocument.fPrintInfo = fHPrint;
37  336 -1              END;
37  337 --          IF NOT dontDispose THEN
37  338 --              IF fHPrint <> NIL THEN
37  339 --                  DisposHandle(Handle(fHPrint));
37  340 --          INHERITED Free;
37  341 -0 A    END;
37  342 --
37  343 --
37  344 --      {$S PrintImage}
37  345 --      {-----------------------------------+
37  346 --      |   AdornPage                       |
37  347 --      +-----------------------------------}
37  348 -- A    PROCEDURE TStdPrintHandler.AdornPage;
37  349 --      {$IFC qDebug}
37  350 --      VAR
37  351 --          heading:    Str255;
37  352 --          pgStr:      Str255;
37  353 --          handyRect:  Rect;
37  354 --          xLoc:       INTEGER;
37  355 --          rPlusL:     INTEGER;
37  356 --      {$ENDC}
37  357 0- A    BEGIN
37  358 --      {$IFC qDebug}
37  359 --          IF gDebugPrinting THEN          { Print extra stuff if debugging }
37  360 1-              BEGIN
37  361 --              NumToString(fFocusedPage, pgStr);
37  362 --              TextFont(applFont);
37  363 --              TextFace([]);
37  364 --              TextSize(12);
37  365 --              heading := CONCAT('-', pgStr, '-');  { ??? Make easier for client to change this }
37  366 --
37  367 --              { avoid a $H problem by precomputing the values we need from fPageAreas.thePaper }
37  368 --              WITH fPageAreas.thePaper DO
37  369 --                  rPlusL := right + left;
37  370 --
37  371 --              xLoc := (rPlusL - StringWidth(heading)) DIV 2;
37  372 --              MoveTo(xLoc, fPageAreas.theInk.bottom - 8); { ??? Arbitrary choice }
37  373 --              DrawString(heading);
37  374 --
37  375 --              { Additionally frame the printable area of the page if gDebugPrinting }
37  376 --              handyRect := fPageAreas.theInk;
37  377 --              PenSize(1, 1);
37  378 --              FrameRect(handyRect);
37  379 --
37  380 --              { Frame the 'interior' of the page }
37  381 --              PenSize(2, 2);
37  382 --              handyRect := fPageAreas.theInterior;
37  383 --              FrameRect(handyRect);
37  384 -1              END;
37  385 --      {$ENDC}
37  386 -0 A    END;
37  387 --
37  388 --
37  389 --      {$S PrintActual}
37  390 --      {-----------------------------------+
37  391 --      |   BanishPrintDialog               |
37  392 --      +-----------------------------------}
37  393 -- A    PROCEDURE TStdPrintHandler.BanishPrintDialog;
37  394 0- A    BEGIN
37  395 --          IF gPrintDialog <> NIL THEN
```

```
37  396 1-              BEGIN
37  397 --              DisposDialog(gPrintDialog);
37  398 --              gPrintDialog := NIL;
37  399 -1             END;
37  400 -0 A    END;
37  401 --
37  402 --
37  403 --      {$S PrintRes}
37  404 --      {-----------------------------------+
37  405 --      |   BreakFollowing                  |
37  406 --      +-----------------------------------}
37  407 -- A    FUNCTION  TStdPrintHandler.BreakFollowing(vhs: VHSelect; prevBreak: VCoordinate;
37  408 --                  VAR automatic: BOOLEAN): VCoordinate; OVERRIDE;
37  409 --          { Called from fView.DoBreakFollowing, }
37  410 --
37  411 --      VAR
37  412 --          orthoVHS:   VHSelect;
37  413 --          newLoc:     VCoordinate;
37  414 0- A    BEGIN
37  415 --          orthoVHS := gOrthogonal[vhs];
37  416 --          automatic := TRUE;
37  417 --          newLoc := Min(prevBreak + fViewPerPage.vh[orthoVHS], fPrintExtent.botRight.vh[orthoVHS]);
37  418 --
37  419 --      {$IFC qDebug}
37  420 --          IF newLoc <= prevBreak THEN
37  421 1-              BEGIN
37  422 --              WriteLn('No advance in BreakFollowing: vhs = ', ORD(vhs):1, ' prevBreak = ', prevBreak:1,
37  423 --                  ' newLoc = ', newLoc:1, ' view size: ', fPrintExtent.botRight.vh[orthoVHS]:1);
37  424 --              ProgramBreak('No advance in BreakFollowing');
37  425 -1             END;
37  426 --      {$ENDC}
37  427 --
37  428 --          BreakFollowing := newLoc;
37  429 -0 A    END;
37  430 --
37  431 --
37  432 --      {$S PrintNonRes}
37  433 --      {-----------------------------------+
37  434 --      |   CalcPageStrips                  |
37  435 --      +-----------------------------------}
37  436 -- A    PROCEDURE TStdPrintHandler.CalcPageStrips (VAR pageStrips: Point); OVERRIDE;
37  437 --      VAR
37  438 --          vhs,
37  439 --          ortho:      VHSelect;
37  440 --          nStrips:    INTEGER;
37  441 --
37  442 --          {-------------------------------+
37  443 --          |   FindLimit                   |
37  444 --          +-------------------------------}
37  445 -- B        FUNCTION  FindLimit(loc: VCoordinate; automatic: BOOLEAN): BOOLEAN;
37  446 0- B        BEGIN
37  447 --              nStrips := nStrips + 1;
37  448 --              FindLimit := FALSE;
37  449 -0 B        END;
37  450 --
37  451 0- A    BEGIN
37  452 --
37  453 --          { If pages are of fixed size, then simple divide the total size by the
37  454 --              page size.  Otherwise, count up the page breaks one by one. }
37  455 --
37  456 --          FOR vhs := v TO h DO
37  457 1-              BEGIN
37  458 --              ortho := gOrthogonal[vhs];
37  459 --              IF fFixedSizePages[ortho] THEN
37  460 --                  {$H-}
37  461 --                  WITH fPrintExtent, fViewPerPage DO
37  462 --                      pageStrips.vh[vhs] :=
37  463 --                          (botRight.vh[ortho] - topLeft.vh[ortho] + vh[ortho] - 1) DIV vh[ortho]
37  464 --                  {$H+}
37  465 --              ELSE
37  466 2-                  BEGIN
37  467 --                  nStrips := 0;
37  468 --                  EachBreak(vhs, TRUE, FindLimit);
37  469 --                  pageStrips.vh[vhs] := nStrips;
37  470 -2                  END;
37  471 -1              END;
37  472 -0 A    END;
37  473 --
37  474 --
37  475 --      {$S PrintNonRes}
37  476 --      {-----------------------------------+
37  477 --      |   CalcViewPerPage                 |
37  478 --      +-----------------------------------}
37  479 -- A    PROCEDURE TStdPrintHandler.CalcViewPerPage(VAR amtPerPage: VPoint); OVERRIDE;
37  480 --      VAR
37  481 --          vhs:    VHSelect;
37  482 0- A    BEGIN
37  483 --          WITH fPageAreas DO {$H-}
37  484 --            FOR vhs := v TO h DO
37  485 --              amtPerPage.vh[vhs] := MAX(1,
37  486 --                ((ORD4(thePaper.botRight.vh[vhs] - thePaper.topLeft.vh[vhs]
37  487 --                  - ABS(theMargins.topLeft.vh[vhs])
37  488 --                  - ABS(theMargins.botRight.vh[vhs])))); {$H+}
37  489 -0 A    END;
37  490 --
37  491 --
37  492 --      {$S PrintRes}
37  493 --      {-----------------------------------+
37  494 --      |   CheckPrinter                    |
```

```
37  495 --      +---------------------------------}
37  496 -- A   PROCEDURE TStdPrintHandler.CheckPrinter; OVERRIDE;
37  497 --      VAR
37  498 --          oldPageAreas:       PageAreas;
37  499 --          oldPrinterDev:      INTEGER;
37  500 --          oldDevRes:          Point;
37  501 --          oldEffectiveRes:    Point;
37  502 --          didChange:          BOOLEAN;
37  503 --          msgSent:            BOOLEAN;
37  504 --          driverName:         Str255;
37  505 --
37  506 --          {-------------------------------+
37  507 --          |   ItChanged                   |
37  508 --          +-------------------------------}
37  509 -- B       PROCEDURE ItChanged(aView: TView); { this msg is sent to all views if my associated document is the
37  510 --                                                  kind where one copy of the PrintInfo is shared among all views }
37  511 0- B      BEGIN
37  512 --          aView.DoPrinterChanged;
37  513 -0 B      END;
37  514 0- A   BEGIN
37  515 --          oldPageAreas := fPageAreas;
37  516 --          oldPrinterDev := fPrinterDev;
37  517 --          oldDevRes := fDeviceRes;
37  518 --          oldEffectiveRes := fEffectiveDeviceRes;
37  519 --
37  520 --          { It costs an open and several LoadResources to call PrValidate.  Before
37  521 --              doing so, check to see if the printer has actually changed. }
37  522 --          IF fLastCheckedPrinter < gLastDeskAcc THEN
37  523 --              { If we haven't checked printer since the last time
37  524 --                  the user might have run the Chooser... }
37  525 1-              BEGIN
37  526 --              GetDriverName(driverName);
37  527 --              IF NOT EqualString(fLastPrinterName, driverName, FALSE, TRUE) THEN
37  528 2-                  BEGIN
37  529 --                  { Printer name has changed; remember the new one and re-validate
37  530 --                      the print record. }
37  531 --                  BlockMove(@driverName, @fLastPrinterName, LENGTH(driverName));
37  532 --                  ValidatePrintRecord(didChange);
37  533 -2                  END;
37  534 --              fLastCheckedPrinter := TickCount;
37  535 -1              END;
37  536 --
37  537 --          WITH THPrint(fHPrint)^^  DO
37  538 1-      {$H-}   BEGIN
37  539 --              fPageAreas.thePaper := rPaper;
37  540 --
37  541 --              WITH fPageAreas.thePaper, fEffectiveDeviceRes DO    { Recompute effective device resolution }
37  542 2-                  BEGIN
37  543 --                  h := (IntMultiply(iPrPgFract, right - left)) DIV prStl.iPageH;
37  544 --                  v := (IntMultiply(iPrPgFract, bottom - top)) DIV prStl.iPageV;
37  545 -2                  END;
37  546 --
37  547 --              WITH prInfo DO  { Store latest data from MacPrint prInfo record into my own instance variables }
37  548 2-                  BEGIN
37  549 --                  fPrinterDev := iDev;
37  550 --                  SetPt(fDeviceRes, iHRes, iVRes);
37  551 --                  WITH fPageAreas DO
37  552 3-                      BEGIN
37  553 --                      theInk := rPage;
37  554 --                      IF NOT fMinimalMargins THEN
37  555 --                          SetRect(theMargins, IntMultiply(theMargins.left, fEffectiveDeviceRes.h) DIV oldEffectiveRes.h,
37  556 --                                              IntMultiply(theMargins.top, fEffectiveDeviceRes.v) DIV oldEffectiveRes.v,
37  557 --                                              IntMultiply(theMargins.right, fEffectiveDeviceRes.h) DIV oldEffectiveRes.h,
37  558 --                                              IntMultiply(theMargins.bottom, fEffectiveDeviceRes.v) DIV oldEffectiveRes.v);
37  559 -3                      END;
37  560 -2                  END;
37  561 --      {$H+}
37  562 -1              END;
37  563 --
37  564 --          IF (NOT EqualRect(fPageAreas.thePaper, oldPageAreas.thePaper)) OR
37  565 --                  (NOT EqualRect(fPageAreas.theInk, oldPageAreas.theInk)) OR
37  566 --                  (NOT EqualPt(fDeviceRes, oldDevRes)) OR
37  567 --                  (oldPrinterDev = kNeverInitialized) THEN
37  568 1-                  BEGIN   { something important to our projection model changed... }
37  569 --                  msgSent := FALSE;
37  570 --                  IF fDocument <> NIL THEN
37  571 --                      IF fDocument.fSharePrintInfo THEN
37  572 2-                          BEGIN
37  573 --                          fDocument.ForAllViewsDo(ItChanged);
37  574 --                          msgSent := TRUE;
37  575 -2                          END;
37  576 --
37  577 --                  IF NOT msgSent THEN  { didn't send msg to all views, current company included, so
37  578 --                                              now we need to tell must my local view }
37  579 --                      fView.DoPrinterChanged;
37  580 -1                  END;
37  581 -0 A   END;
37  582 --
37  583 --
37  584 --      {$S PrintImage}
37  585 --      {---------------------------------+
37  586 --      |   ChooseSpoolFile                |
37  587 --      +---------------------------------}
37  588 -- A   PROCEDURE TStdPrintHandler.ChooseSpoolFile(VAR spoolFileName: Str255; VAR spoolVRefNum: INTEGER;
37  589 --          VAR pagesPerSubjob: INTEGER);
37  590 0- A   BEGIN
37  591 --          spoolFileName := '';  { Default choices tell MacPrint to use its standard choice algorithm }
37  592 --          spoolVRefNum := 0;
37  593 --          { Print Shop suggests attempting to print entire document.  If it
```

```
37  594 --              runs out of space, then PerformPrinting will retry with a
37  595 --              smaller number of pages. }
37  596 --          pagesPerSubjob := MAXINT;
37  597 -0 A   END;
37  598 --
37  599 --
37  600 --      {$S PrintRes}
37  601 --      {-----------------------------------+
37  602 --      |   ClosePrintShop                  |
37  603 --      +-----------------------------------}
37  604 -- A   PROCEDURE TStdPrintHandler.ClosePrintShop;
37  605 --
37  606 --      VAR
37  607 --          err:        OSErr;
37  608 --
37  609 0- A   BEGIN
37  610 --          PrClose;
37  611 --      {$IFC qDebug}
37  612 --          err := PrError;     { Now check for fresh error from MacPrint -- only for debugging, since
37  613 --                                  ChkPrintErr checks afresh in the real world }
37  614 --          IF err <> noErr THEN
37  615 --              WriteLn('Error from MacPrint is: ', err:1);
37  616 --      {$ENDC}
37  617 -0 A   END;
37  618 --
37  619 --
37  620 --      {$S PrintRes}
37  621 --      {-----------------------------------+
37  622 --      |   DoInMacPrint                    |
37  623 --      +-----------------------------------}
37  624 -- A   PROCEDURE TStdPrintHandler.DoInMacPrint(PROCEDURE WhatToDo);
37  625 --      VAR
37  626 --          fi:     FailInfo;
37  627 --
37  628 --          {-------------------------------+
37  629 --          |   HdlPrintFailure             |
37  630 --          +-------------------------------}
37  631 -- B       PROCEDURE HdlPrintFailure(error: OSErr; message: LONGINT);
37  632 0- B      BEGIN
37  633 --              ClosePrintShop;
37  634 -0 B       END;
37  635 --
37  636 0- A   BEGIN
37  637 --          IF gCouldPrint THEN
37  638 1-              BEGIN
37  639 --              PrSetError(noErr);    { Clear printer-error flag }
37  640 --              CatchFailures(fi, HdlPrintFailure);
37  641 --              OpenPrintShop;
37  642 --              WhatToDo;           { Do what needs to be done }
37  643 --              Success(fi);
37  644 --              ClosePrintShop;
37  645 -1              END;
37  646 --          { NB: if gCouldPrint is FALSE, DoInMacPrint is a no-op }
37  647 -0 A   END;
37  648 --
37  649 --
37  650 --      {$S PrintSelCommand}
37  651 --      {-----------------------------------+
37  652 --      |   DoMenuCommand                   |
37  653 --      +-----------------------------------}
37  654 -- A   FUNCTION  TStdPrintHandler.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
37  655 --      VAR
37  656 --          proceed:        BOOLEAN;
37  657 --
37  658 0- A   BEGIN
37  659 --          DoMenuCommand := gNoChanges;
37  660 1-          CASE aCmdNumber OF
37  661 --
37  662 --              cPrint:
37  663 2-                  BEGIN
37  664 --                  CheckPrinter;
37  665 --                  IF PoseJobDialog THEN
37  666 --                      DoMenuCommand := Print(aCmdNumber, proceed);
37  667 -2                  END;
37  668 --              cPrintOne:
37  669 2-                  BEGIN
37  670 --                  CheckPrinter;
37  671 --                  IF SetupPrintOne THEN
37  672 --                      DoMenuCommand := Print(aCmdNumber, proceed);
37  673 -2                  END;
37  674 --              cPageSetup:
37  675 --                  DoMenuCommand := PosePageSetupDialog(proceed, TRUE);
37  676 --
37  677 --              cShowBreaks:    { Toggle state of "Show Breaks" }
37  678 2-                  BEGIN
37  679 --                  fShowBreaks := NOT fShowBreaks;
37  680 --                  InvalPageFeedback;  { force redraw of area the breaks did or will occupy }
37  681 -2                  END;
37  682 --
37  683 --              OTHERWISE
37  684 --                  DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
37  685 -1              END { CASE };
37  686 -0 A   END;
37  687 --
37  688 --
37  689 --      {$S PrintRes}
37  690 --      {-----------------------------------+
37  691 --      |   DoSetupMenus                    |
37  692 --      +-----------------------------------}
```

```
37  693 -- A   PROCEDURE TStdPrintHandler.DoSetupMenus; OVERRIDE;
37  694 0- A   BEGIN
37  695 --         INHERITED DoSetupMenus;
37  696 --
37  697 --         IF gCouldPrint AND (fView <> NIL) THEN
37  698 1-             BEGIN
37  699 --             Enable(cPrint, TRUE);
37  700 --             Enable(cPageSetup, TRUE);
37  701 --             Enable(cPrintOne, TRUE);
37  702 -1             END;
37  703 --
37  704 --         EnableCheck(cShowBreaks, TRUE, fShowBreaks);
37  705 -0 A   END;
37  706 --
37  707 --
37  708 --     {$S PrintRes}
37  709 --     {-----------------------------------+
37  710 --     |   DrawPrintFeedback                |
37  711 --     +-----------------------------------}
37  712 -- A   PROCEDURE TStdPrintHandler.DrawPrintFeedback(area: Rect); OVERRIDE;
37  713 --         { Draws page breaks and page numbers }
37  714 --     VAR vhs:            VHSelect;
37  715 --         orthovhs:       VHSelect;
37  716 --         whichBreak:     INTEGER;
37  717 --         viewArea:       VRect;
37  718 --     {$IFC qDebug}
37  719 --         pageNumStyle:   TextStyle;
37  720 --     {$ENDC}
37  721 --
37  722 --         {-------------------------------+
37  723 --         |   DrawABreak                  |
37  724 --         +-------------------------------}
37  725 -- B       FUNCTION  DrawABreak (loc: VCoordinate; automatic: BOOLEAN): BOOLEAN;
37  726 --
37  727 0- B       BEGIN
37  728 --             IF loc > viewArea.botRight.vh[orthovhs] THEN
37  729 --                 DrawABreak := TRUE
37  730 --             ELSE
37  731 1-                 BEGIN
37  732 --                 DrawABreak := FALSE;
37  733 --                 whichBreak := whichBreak + 1;
37  734 --                 IF (loc > viewArea.topLeft.vh[orthovhs] - gBreaksPenState.pnSize.vh[orthovhs]) THEN
37  735 --                     fView.DoDrawPageBreak(vhs, whichBreak, loc, automatic);
37  736 -1                 END;
37  737 -0 B       END;
37  738 --
37  739 0- A   BEGIN
37  740 --         IF fShowBreaks OR gDebugPrinting THEN
37  741 1-             BEGIN
37  742 --             SetPrintExtent;         { Make sure print extent is accurate before starting }
37  743 --
37  744 --     {$IFC qDebug}
37  745 --             IF gDebugPrinting THEN      { Now draw Page numbers in the corners of pages, if desired }
37  746 2-                 BEGIN
37  747 --                 SetTextStyle(pageNumStyle, ApplFont, [bold], 9, gRGBBlack);
37  748 --                 SetPortTextStyle(pageNumStyle);
37  749 -2                 END;
37  750 --     {$ENDC}
37  751 --
37  752 --             SetPenState(gBreaksPenState);
37  753 --             fView.QDToViewRect(area, viewArea);
37  754 --
37  755 --             FOR vhs := v TO h DO
37  756 2-                 BEGIN
37  757 --                 orthovhs := gOrthogonal[vhs];
37  758 --                 whichBreak := 0;
37  759 --                 EachBreak(vhs, FALSE, DrawABreak);
37  760 -2                 END;
37  761 -1             END;
37  762 -0 A   END;
37  763 --
37  764 --
37  765 --     {$S PrintRes}
37  766 --     {-----------------------------------+
37  767 --     |   DrawPageBreak                    |
37  768 --     +-----------------------------------}
37  769 -- A   PROCEDURE TStdPrintHandler.DrawPageBreak (vhs: VHSelect; whichBreak: INTEGER;
37  770 --                                                 loc: VCoordinate; automatic: BOOLEAN); OVERRIDE;
37  771 --
37  772 --     VAR
37  773 --         vPt:            VPoint;
37  774 --         qdStartPt:      Point;
37  775 --         qdEndPt:        Point;
37  776 --     {$IFC qDebug}
37  777 --         i:              INTEGER;
37  778 --         hLoc:           VCoordinate;
37  779 --         aString:        Str255;
37  780 --     {$ENDC}
37  781 --
37  782 0- A   BEGIN
37  783 --         vPt.vh[gOrthogonal[vhs]] := loc;
37  784 --         vPt.vh[vhs] := 0;
37  785 --         qdStartPt := fView.ViewToQDPt(vPt);
37  786 --         vPt.vh[vhs] := fView.fSize.vh[vhs] - gBreaksPenState.pnSize.vh[vhs];
37  787 --         qdEndPt := fView.ViewToQDPt(vPt);
37  788 --
37  789 --         IF fShowBreaks THEN
37  790 1-             BEGIN
37  791 --             MoveTo(qdStartPt.h, qdStartPt.v);
```

```
37  792 --              LineTo(qdEndPt.h, qdEndPt.v);
37  793 -1             END;
37  794 --
37  795 --      {$IFC qDebug}
37  796 --          IF gDebugPrinting THEN
37  797 --              IF vhs = h THEN
37  798 --                  FOR i := 0 TO fPageStrips.v DO
37  799 1-                      BEGIN
37  800 --                      IF i = 0 THEN
37  801 --                          hLoc := 0
37  802 --                      ELSE
37  803 --                          GetBreakCoord(v, i, hLoc);
37  804 --
37  805 --                      NumToString(StripToPage(whichBreak - 1, i), aString);
37  806 --                      MoveTo(hLoc + 3, qdStartPt.v - 3);
37  807 --                      DrawString(aString);
37  808 --
37  809 --                      NumToString(StripToPage(whichBreak, i), aString);
37  810 --                      MoveTo(hLoc + 3, qdEndPt.v + 10);
37  811 --                      DrawString(aString);
37  812 --
37  813 --                      NumToString(StripToPage(whichBreak - 1, i - 1), aString);
37  814 --                      MoveTo(hLoc - StringWidth(aString) - 3, qdStartPt.v - 3);
37  815 --                      DrawString(aString);
37  816 --
37  817 --                      NumToString(StripToPage(whichBreak, i - 1), aString);
37  818 --                      MoveTo(hLoc - StringWidth(aString) - 3, qdEndPt.v + 10);
37  819 --                      DrawString(aString);
37  820 -1                     END;
37  821 --      {$ENDC}
37  822 -0 A   END;
37  823 --
37  824 --
37  825 --      {$S PrintImage}
37  826 --      {-----------------------------------+
37  827 --      |   DrawPageInterior                |
37  828 --      +-----------------------------------}
37  829 -- A    PROCEDURE TStdPrintHandler.DrawPageInterior;
37  830 0- A   BEGIN
37  831 --          fView.DrawContents; { i.e., by default, the same code used for drawing on the screen }
37  832 -0 A   END;
37  833 --
37  834 --
37  835 --      {$S PrintRes}
37  836 --      {-----------------------------------+
37  837 --      |   EachBreak                       |
37  838 --      +-----------------------------------}
37  839 -- A    PROCEDURE TStdPrintHandler.EachBreak(vhs: VHSelect; includeLast: BOOLEAN;
37  840 --                  FUNCTION  DoToBreak(loc: VCoordinate; automatic: BOOLEAN): BOOLEAN);
37  841 --      VAR
37  842 --          startLoc:   VCoordinate;
37  843 --          endLoc:     VCoordinate;
37  844 --          loc:        VCoordinate;
37  845 --          automatic:  BOOLEAN;
37  846 --          done:       BOOLEAN;
37  847 0- A   BEGIN
37  848 --          WITH fPrintExtent DO
37  849 1-              BEGIN
37  850 --              startLoc := topLeft.vh[gOrthogonal[vhs]];
37  851 --              endLoc := botRight.vh[gOrthogonal[vhs]];
37  852 -1              END;
37  853 --
37  854 --          loc := startLoc;
37  855 --          automatic := TRUE;
37  856 --          done := FALSE;
37  857 --          WHILE (loc < endLoc) AND NOT done DO
37  858 1-              BEGIN
37  859 --              IF loc <> startLoc THEN
37  860 --                  done := DoToBreak(loc, automatic);
37  861 --              loc := fView.DoBreakFollowing(vhs, loc, automatic);
37  862 -1              END;
37  863 --
37  864 --          IF includeLast THEN
37  865 --              done := DoToBreak(loc, automatic);
37  866 -0 A   END;
37  867 --
37  868 --
37  869 --      {$IFC qDebug}
37  870 --      {$S PrintFields}
37  871 --      {-----------------------------------+
37  872 --      |   Fields                          |
37  873 --      +-----------------------------------}
37  874 -- A    PROCEDURE TStdPrintHandler.Fields (PROCEDURE DoToField (fieldName: Str255;
37  875 --                                              fieldAddr: Ptr;
37  876 --                                              fieldType: INTEGER));
37  877 0- A   BEGIN
37  878 --          DoToField('TStdPrintHandler', NIL, bClass);
37  879 --          DoToField('fPageAreas', NIL, bTitle);
37  880 --          DoToField('  thePaper', @fPageAreas.thePaper, bRect);
37  881 --          DoToField('  theInk', @fPageAreas.theInk, bRect);
37  882 --          DoToField('  theMargins', @fPageAreas.theMargins, bRect);
37  883 --          DoToField('  theInterior', @fPageAreas.theInterior, bRect);
37  884 --          DoToField('fDocument', @fDocument, bObject);
37  885 --          DoToField('fPrintExtent', @fPrintExtent, bVRect);
37  886 --          DoToField('fFixedSize[h]', @fFixedSizePages[h], bBoolean);
37  887 --          DoToField('fFixedSize[v]', @fFixedSizePages[v], bBoolean);
37  888 --          DoToField('fHPrint', @fHPrint, bHandle);
37  889 --          DoToField('fPageStrips', @fPageStrips, bPoint);
37  890 --          DoToField('fStartPage', @fStartPage, bInteger);
```

```
37  891 --          DoToField('fPrinterDev', @fPrinterDev, bInteger);
37  892 --          DoToField('fLastCheckedPrinter', @fLastCheckedPrinter, bLongInt);
37  893 --          DoToField('fLastPrinterName', @fLastPrinterName, bString);
37  894 --          DoToField('fPageDirection', @fPageDirection, bByte);
37  895 --          DoToField('fShowBreaks', @fShowBreaks, bBoolean);
37  896 --          DoToField('fFinderSetup', @fFinderSetup, bBoolean);
37  897 --          DoToField('fFinderJobDialog', @fFinderJobDialog, bBoolean);
37  898 --          DoToField('fSquareDots', @fSquareDots, bBoolean);
37  899 --          DoToField('fMinimalMargins', @fMinimalMargins, bBoolean);
37  900 --          DoToField('fLastStrip', @fLastStrip, bPoint);
37  901 --          DoToField('fLastBreak', @fLastBreak, bVPoint);
37  902 --          DoToField('fViewedRect', @fViewedRect, bVRect);
37  903 --          INHERITED Fields(DoToField);
37  904 -0 A    END;
37  905 --      {$ENDC}
37  906 --
37  907 --
37  908 --      {$S PrintImage}
37  909 --      {-----------------------------------+
37  910 --      |   FocusOnBorder                   |
37  911 --      +-----------------------------------}
37  912 -- A    PROCEDURE TStdPrintHandler.FocusOnBorder;
37  913 --      VAR
37  914 --          rectToClipTo:   Rect;
37  915 0- A    BEGIN
37  916 --          rectToClipTo := fPageAreas.theInk;
37  917 --          WITH fPageAreas.theInk DO
37  918 --      {$H-}
37  919 --              SetOrigin(left, top);       { Only works for newer LaserWriter drivers }
37  920 --      {$H+}
37  921 --          ClipRect(rectToClipTo);
37  922 -0 A    END;
37  923 --
37  924 --
37  925 --      {$S PrintImage}
37  926 --      {-----------------------------------+
37  927 --      |   FocusOnInterior                 |
37  928 --      +-----------------------------------}
37  929 -- A    PROCEDURE TStdPrintHandler.FocusOnInterior; OVERRIDE;
37  930 --      VAR
37  931 --          rectToClip:         Rect;
37  932 --          aRect:              Rect;
37  933 --          theOrigin:          Point;
37  934 --          vhs:                VHSelect;
37  935 0- A    BEGIN
37  936 --      {$H-}
37  937 --          WITH fPageAreas DO
37  938 1-            BEGIN
37  939 --              aRect := theInk;
37  940 --              theOrigin := theInk.topLeft;
37  941 --              FOR vhs := v TO h DO
37  942 --                  IF fView.fSize.vh[vhs] > kMaxCoord THEN
37  943 --                      gLongOffset.vh[vhs] := gPageOffset.vh[vhs]
37  944 --                  ELSE
37  945 2-                    BEGIN
37  946 --                      gLongOffset.vh[vhs] := 0;
37  947 --                      theOrigin.vh[vhs] := theOrigin.vh[vhs] + gPageOffset.vh[vhs];
37  948 --                      aRect.topLeft.vh[vhs] := aRect.topLeft.vh[vhs] + gPageOffset.vh[vhs];
37  949 --                      aRect.botRight.vh[vhs] := aRect.botRight.vh[vhs] + gPageOffset.vh[vhs];
37  950 -2                    END;
37  951 -1            END;
37  952 --      {$H+}
37  953 --          SetOrigin(theOrigin.h, theOrigin.v);
37  954 --
37  955 --          { Clip the page to the intersection of the visible part of the view and the
37  956 --              printable area of the page.  Note that in some cases (e.g., a WYSIWYG
37  957 --              word processor which showed the complete page margin) parts of
37  958 --              the interior might lie outside theInk.  We must clip to theInk to
37  959 --              avoid slowing down PostScript printers. }
37  960 --          fView.ViewToQDRect(fViewedRect, rectToClip);
37  961 --          IF SectRect(rectToClip, aRect, rectToClip) THEN ;
37  962 --          ClipRect(rectToClip);
37  963 --
37  964 -0 A    END;
37  965 --
37  966 --
37  967 --      {$S PrintRes}
37  968 --      {-----------------------------------+
37  969 --      |   GetBreakCoord                   |
37  970 --      +-----------------------------------}
37  971 -- A    PROCEDURE TStdPrintHandler.GetBreakCoord(vhs: VHSelect; whichBreak: INTEGER; VAR loc: VCoordinate);
37  972 --      VAR
37  973 --          automatic:  BOOLEAN;
37  974 --          i:          INTEGER;
37  975 --          startBreak: INTEGER;
37  976 --          orthoVHS:   VHSelect;
37  977 0- A    BEGIN
37  978 --          orthoVHS := gOrthogonal[vhs];
37  979 --          IF fFixedSizePages[orthoVHS] THEN
37  980 --              loc := Min(fPrintExtent.topLeft.vh[orthoVHS] + fViewPerPage.vh[orthoVHS] * whichBreak,
37  981 --                      fPrintExtent.botRight.vh[orthoVHS])
37  982 --          ELSE IF whichBreak = fLastStrip.vh[vhs] THEN
37  983 --              loc := fLastBreak.vh[vhs]
37  984 --          ELSE
37  985 1-            BEGIN
37  986 --              IF whichBreak > fLastStrip.vh[vhs] THEN
37  987 2-                BEGIN
37  988 --                  startBreak := fLastStrip.vh[vhs] + 1;
37  989 --                  loc := fLastBreak.vh[vhs];
```

```
37  990 -2                  END
37  991 --              ELSE
37  992 2-                  BEGIN
37  993 --                  startBreak := 1;
37  994 --                  loc := fPrintExtent.topLeft.vh[gOrthogonal[vhs]];
37  995 -2                  END;
37  996 --              FOR i := startBreak TO whichBreak DO
37  997 --                  loc := fView.DoBreakFollowing(vhs, loc, automatic);  { ??? error handling??? }
37  998 -1              END;
37  999 --
37 1000 --          fLastStrip.vh[vhs] := whichBreak;
37 1001 --          fLastBreak.vh[vhs] := loc;
37 1002 -0 A   END;
37 1003 --
37 1004 --
37 1005 --      {$S PrintActual}
37 1006 --      {-----------------------------------+
37 1007 --      |   GetDocName                      |
37 1008 --      +-----------------------------------}
37 1009 -- A   PROCEDURE TStdPrintHandler.GetDocName(VAR docName: Str255);
37 1010 --      VAR
37 1011 --          aWindow:    TWindow;
37 1012 0- A   BEGIN
37 1013 --          IF fDocument <> NIL THEN
37 1014 --              docName := fDocument.fTitle
37 1015 --          ELSE
37 1016 --              docName := '';
37 1017 --          IF docName = '' THEN
37 1018 1-              BEGIN
37 1019 --              aWindow := fView.GetWindow;
37 1020 --              IF aWindow <> NIL THEN
37 1021 --                  GetWTitle(aWindow.fWmgrWindow, docName);
37 1022 -1              END;
37 1023 --      {$IFC qDebug}
37 1024 --          IF docName = '' THEN
37 1025 --              ProgramBreak('GetDocName can''t get a document or window name');
37 1026 --      {$ENDC}
37 1027 -0 A   END;
37 1028 --
37 1029 --
37 1030 --      {$IFC qDebug}
37 1031 --      {$S PrintDebug}
37 1032 --      {-----------------------------------+
37 1033 --      |   IdentifySoftware                |
37 1034 --      +-----------------------------------}
37 1035 -- A   PROCEDURE TStdPrintHandler.IdentifySoftware;
37 1036 0- A   BEGIN
37 1037 --          WriteLn('UPrinting of ', COMPDATE, ' @ ', COMPTIME);
37 1038 -0 A   END;
37 1039 --      {$ENDC}
37 1040 --
37 1041 --
37 1042 --      {$S PrintNonRes}
37 1043 --      {-----------------------------------+
37 1044 --      |   InstallMargins                  |
37 1045 --      +-----------------------------------}
37 1046 -- A   PROCEDURE TStdPrintHandler.InstallMargins(newMargins: Rect; areMinimalMargins: BOOLEAN);
37 1047 0- A   BEGIN
37 1048 --          fMinimalMargins := areMinimalMargins;
37 1049 --
37 1050 --          IF fMinimalMargins THEN
37 1051 1-              BEGIN
37 1052 --              WITH fPageAreas DO
37 1053 2-                  BEGIN
37 1054 --                  {$H-}
37 1055 --                  theMargins := theInk;
37 1056 --                  SubPt(thePaper.topLeft, theMargins.topLeft);
37 1057 --                  SubPt(thePaper.botRight, theMargins.botRight);
37 1058 --                  {$H+}
37 1059 -2                  END;
37 1060 --              WITH fPageAreas DO
37 1061 --                  theInterior := theInk;
37 1062 -1              END
37 1063 --          ELSE
37 1064 1-              BEGIN
37 1065 --              fPageAreas.theMargins := newMargins;
37 1066 --              WITH fPageAreas DO
37 1067 2-                  BEGIN
37 1068 --                  theInterior := thePaper;
37 1069 --           {$H-}  AddPt(theMargins.topLeft, theInterior.topLeft);
37 1070 --                  AddPt(theMargins.botRight, theInterior.botRight); {$H+}
37 1071 -2                  END;
37 1072 -1              END;
37 1073 --
37 1074 -0 A   END;
37 1075 --
37 1076 --
37 1077 --      {$S PrintNonRes}
37 1078 --      {-----------------------------------+
37 1079 --      |   InvalPageFeedback               |
37 1080 --      +-----------------------------------}
37 1081 -- A   PROCEDURE TStdPrintHandler.InvalPageFeedback;
37 1082 0- A   BEGIN
37 1083 --          IF ShowsOnScreen THEN
37 1084 --              fView.ForceRedraw;
37 1085 -0 A   END;
37 1086 --
37 1087 --
37 1088 --      {$S PrintNonRes}
```

```
37 1089 --      {-----------------------------------+
37 1090 --      |   LocatePageInterior              |
37 1091 --      +-----------------------------------}
37 1092 -- A    PROCEDURE TStdPrintHandler.LocatePageInterior(pageNumber: INTEGER; VAR loc: Point); OVERRIDE;
37 1093 --
37 1094 0- A    BEGIN
37 1095 --          WITH fPageAreas DO
37 1096 1-              BEGIN
37 1097 --              loc := thePaper.topLeft;
37 1098 --       {$H-}  AddPt(theMargins.topLeft, loc); {$H+}
37 1099 -1              END;
37 1100 -0 A    END;
37 1101 --
37 1102 --
37 1103 --      {$S PrintActual}
37 1104 --      {-----------------------------------+
37 1105 --      |   MaxPageNumber                   |
37 1106 --      +-----------------------------------}
37 1107 -- A    FUNCTION  TStdPrintHandler.MaxPageNumber: INTEGER; OVERRIDE;
37 1108 0- A    BEGIN
37 1109 --          WITH fPageStrips DO
37 1110 --              MaxPageNumber := fStartPage + (v * h) - 1;
37 1111 -0 A    END;
37 1112 --
37 1113 --
37 1114 --      {$S PrintActual}
37 1115 --      {-----------------------------------+
37 1116 --      |   OneSubJob                       |
37 1117 --      +-----------------------------------}
37 1118 -- A    FUNCTION  TStdPrintHandler.OneSubJob(subjobFirstPage, subjobLastPage: INTEGER;
37 1119 --              justSpool: BOOLEAN; partialJob: BOOLEAN; VAR ranOutOfSpace: BOOLEAN;
37 1120 --              VAR lastPageTried: INTEGER; VAR proceed: BOOLEAN): TCommand;
37 1121 --      LABEL   1000;
37 1122 --      VAR
37 1123 --          pass:               INTEGER;
37 1124 --          noOfCopies:         INTEGER;
37 1125 --          aPageNumber:        INTEGER;
37 1126 --          err:                OSErr;
37 1127 --          fi:                 FailInfo;
37 1128 --          {-------------------------------+
37 1129 --          |   FieldClientError            |
37 1130 --          +-------------------------------}
37 1131 -- B        PROCEDURE FieldClientError(error: OSErr; message: LONGINT);
37 1132 0- B        BEGIN
37 1133 --              proceed := FALSE;
37 1134 --              err := error;   { pass along client's error code }  { Why??? }
37 1135 --              GOTO 1000;
37 1136 -0 B        END;
37 1137 0- A    BEGIN
37 1138 --      {$IFC qDebug}
37 1139 --          IF gDebugPrinting THEN
37 1140 --              WriteLn('OneSubJob entered for pages ', subjobFirstPage:1, ' through ', subJobLastPage:1, ', proceed=', ORD(proceed):0);
37 1141 --      {$ENDC}
37 1142 --          OneSubJob := gNoChanges;
37 1143 --          ranOutOfSpace := FALSE;
37 1144 --          lastPageTried := subjobFirstPage - 1;
37 1145 --          WITH THPrint(fHPrint)^^.PrJob DO
37 1146 1-              BEGIN
37 1147 --              iFstPage := 1;
37 1148 --              iLstPage := subjobLastPage - subjobFirstPage + 1;
37 1149 --
37 1150 --              IF bjDocLoop = BSpoolLoop THEN
37 1151 --                  noOfCopies := 1
37 1152 --              ELSE
37 1153 --                  noOfCopies := iCopies;
37 1154 -1              END;
37 1155 --          gPPrPort := PrOpenDoc(THPrint(fHPrint), NIL, NIL);
37 1156 --                      { ??? need to allow/encourage app to supply nonNil args? }
37 1157 --          ChkPrintErr(err, proceed, ranOutOfSpace);
37 1158 --          gCurrPrintHandler := SELF;
37 1159 --
37 1160 --          IF proceed THEN
37 1161 1-              BEGIN
37 1162 --              SetPort(GrafPtr(gPPrPort));
37 1163 --
37 1164 --              gPrinting := TRUE;
37 1165 --              FOR pass := 1 to noOfCopies DO
37 1166 2-                  BEGIN
37 1167 --                  FOR aPageNumber := subjobFirstPage TO subjobLastPage DO
37 1168 --                      IF proceed THEN
37 1169 3-                          BEGIN
37 1170 --                          lastPageTried := aPageNumber;
37 1171 --                          CatchFailures(fi, FieldClientError);
37 1172 --                          PrintPage(aPageNumber);
37 1173 --                          Success(fi);
37 1174 --      1000:
37 1175 --                          ChkPrintErr(err, proceed, ranOutOfSpace);
37 1176 -3                          END;
37 1177 -2                  END;
37 1178 --              gPrinting := FALSE;
37 1179 -1              END;
37 1180 --
37 1181 --          gCurrPrintHandler := NIL;
37 1182 --          PrCloseDoc(gPPrPort);
37 1183 --          ChkPrintErr(err, proceed, ranOutOfSpace);
37 1184 --
37 1185 --          IF ranOutOfSpace THEN
37 1186 --              Exit(OneSubJob)
37 1187 --          ELSE IF proceed THEN
```

```
37 1188 1-              BEGIN
37 1189 --              IF THPrint(fHPrint)^^.PrJob.BJDocLoop = BSpoolLoop THEN
37 1190 --                  IF NOT justSpool THEN
37 1191 --                      PrintSpoolFile(fHPrint, err, proceed);
37 1192 -1              END;
37 1193 --          IF NOT proceed THEN
37 1194 --              IF err <> iprAbort THEN
37 1195 --                  Failure(err, 0);
37 1196 -0 A    END;
37 1197 --
37 1198 --
37 1199 --      {$S PrintRes}
37 1200 --      {-----------------------------------+
37 1201 --      |   OpenPrintShop                   |
37 1202 --      +-----------------------------------}
37 1203 -- A    PROCEDURE TStdPrintHandler.OpenPrintShop;
37 1204 --
37 1205 --      VAR
37 1206 --          err:        OSErr;
37 1207 --
37 1208 0- A    BEGIN
37 1209 --          PrOpen;             { Open the print shop }
37 1210 --          err := PrError;     { Get code }
37 1211 --          IF err <> NoErr THEN
37 1212 1-              BEGIN
37 1213 --              IF (err = fnfErr) OR (err = resFNotFound) THEN
37 1214 --                  err := errNoPrintDrvr;
37 1215 --              Failure(err, 0);
37 1216 -1              END;
37 1217 -0 A    END;
37 1218 --
37 1219 --
37 1220 --      {$S PrintRes}
37 1221 --      {-----------------------------------+
37 1222 --      |   PageToStrip                     |
37 1223 --      +-----------------------------------}
37 1224 -- A    FUNCTION  TStdPrintHandler.PageToStrip (pageNumber: INTEGER): Point;
37 1225 --
37 1226 --      VAR
37 1227 --          normalizedPageNum:      INTEGER;
37 1228 --          ortho:                  VHSelect;
37 1229 --          strip:                  Point;
37 1230 0- A    BEGIN
37 1231 --          normalizedPageNum := pageNumber - fStartPage + 1;
37 1232 --
37 1233 --          ortho := gOrthogonal[fPageDirection];
37 1234 --
37 1235 --          strip.vh[ortho] :=  ((normalizedPageNum - 1) DIV fPageStrips.vh[ortho]);
37 1236 --          strip.vh[fPageDirection] := normalizedPageNum -
37 1237 --                                      ((strip.vh[ortho]) * fPageStrips.vh[ortho]) - 1;
37 1238 --
37 1239 --          PageToStrip := strip;
37 1240 -0 A    END;
37 1241 --
37 1242 --
37 1243 --      {$S PrintRes}
37 1244 --      {-----------------------------------+
37 1245 --      |   PointToPageStrip                |
37 1246 --      +-----------------------------------}
37 1247 -- A    FUNCTION  TStdPrintHandler.PointToPageStrip(pointInView: VPoint): Point;
37 1248 --          { !!! Need to test this! }
37 1249 --      VAR
37 1250 --          currCoord:      INTEGER;
37 1251 --          automatic:      BOOLEAN;
37 1252 --          aStrip:         INTEGER;
37 1253 --          vhs:            VHSelect;
37 1254 --          pageStrip:      Point;
37 1255 0- A    BEGIN
37 1256 --          FOR vhs := v TO h DO
37 1257 --              IF fFixedSizePages[vhs] THEN
37 1258 --                  pageStrip.vh[vhs] := (PointInView.vh[vhs] DIV fViewPerPage.vh[vhs]) + 1
37 1259 --              ELSE
37 1260 1-                  BEGIN
37 1261 --                  pageStrip.vh[vhs] := 1;
37 1262 --                  currCoord := fPrintExtent.topLeft.vh[vhs];  { ??? or orthogonal? }
37 1263 --                  WHILE pointInView.vh[vhs] > currCoord DO    { ??? or orthogonal? }
37 1264 2-                      BEGIN
37 1265 --                      currCoord := fView.DoBreakFollowing(vhs, currCoord, automatic);
37 1266 --                      pageStrip.vh[vhs] := aStrip + 1;
37 1267 -2                      END;
37 1268 -1                  END;
37 1269 --          PointToPageStrip := pageStrip;
37 1270 -0 A    END;
37 1271 --
37 1272 --
37 1273 --      {$S PrintActual}
37 1274 --      {-----------------------------------+
37 1275 --      |   PoseJobDialog                   |
37 1276 --      +-----------------------------------}
37 1277 -- A    FUNCTION  TStdPrintHandler.PoseJobDialog: BOOLEAN;
37 1278 --      VAR
37 1279 --          b:          BOOLEAN;
37 1280 --          proceed:    BOOLEAN;
37 1281 --          temp:       INTEGER;
37 1282 --          err:        OSErr;
37 1283 --
37 1284 --          {-------------------------------+
37 1285 --          |   CallJobDialog               |
37 1286 --          +-------------------------------}
```

```
37 1287 -- B       PROCEDURE CallJobDialog;
37 1288 0- B       BEGIN
37 1289 --             proceed := PrJobDialog(THPrint(fHPrint));
37 1290 -0 B       END;
37 1291 --
37 1292 0- A   BEGIN
37 1293 --         proceed := TRUE;
37 1294 --         (* PrepareForDialog; *)
37 1295 --         DoInMacPrint(CallJobDialog);
37 1296 --         ChkPrintErr(err, proceed, b);
37 1297 --
37 1298 --         WITH THPrint(fHPrint)^^.prJob DO
37 1299 --             IF iFstPage > iLstPage THEN { Rectify the range as a public service if needed }
37 1300 1-                 BEGIN
37 1301 --                 temp := iLstPage;
37 1302 --                 iLstPage := iFstPage;
37 1303 --                 iFstPage := temp;
37 1304 -1                 END;
37 1305 --
37 1306 --         gApplication.UpdateAllWindows;
37 1307 --         PoseJobDialog := proceed;
37 1308 -0 A   END;
37 1309 --
37 1310 --
37 1311 --     {$S PrintActual}    { Needs to be here because it may be called from Print }
37 1312 --     {-----------------------------------+
37 1313 --     |   PosePageSetupDialog             |
37 1314 --     +-----------------------------------}
37 1315 -- A   FUNCTION  TStdPrintHandler.PosePageSetupDialog(
37 1316 --                     VAR proceed: BOOLEAN; isUndoable: BOOLEAN): TCommand;
37 1317 --     VAR
37 1318 --         react:  BOOLEAN;
37 1319 --         aPrintStyleChangeCommand:   TPrintStyleChangeCommand;
37 1320 --
37 1321 --         {-------------------------------+
37 1322 --         |   CallStyleDialog             |
37 1323 --         +-------------------------------}
37 1324 -- B       PROCEDURE CallStyleDialog;
37 1325 0- B       BEGIN
37 1326 --             react :=  PrStlDialog(THPrint(fHPrint));
37 1327 -0 B       END;
37 1328 --
37 1329 0- A   BEGIN
37 1330 --         PosePageSetupDialog := gNoChanges;
37 1331 --         react := FALSE;  { in case MacPrint code not accessible }
37 1332 --         IF NOT isUndoable THEN
37 1333 1-             BEGIN
37 1334 --             (* PrepareForDialog; *)
37 1335 --             DoInMacPrint(CallStyleDialog);
37 1336 --             IF react THEN
37 1337 --                 CheckPrinter;
37 1338 -1             END
37 1339 --         ELSE
37 1340 1-             BEGIN
37 1341 --             { Must setup command before putting up page setup dialog because the
37 1342 --                 command records the current print record to make it undoable. }
37 1343 --             New(aPrintStyleChangeCommand);
37 1344 --             FailNIL(aPrintStyleChangeCommand);
37 1345 --             aPrintStyleChangeCommand.IPrintStyleChangeCommand(SELF);
37 1346 --
37 1347 --             (* PrepareForDialog; *)
37 1348 --             DoInMacPrint(CallStyleDialog);  { Put up the Page Setup Dialog }
37 1349 --
37 1350 --             IF react THEN   { User specified a change }
37 1351 2-                 BEGIN
37 1352 --                 BlockMove(fHPrint^, aPrintStyleChangeCommand.fNewHPrint^,
37 1353 --                             SIZEOF(TPrint));
37 1354 --                 PosePageSetupDialog := aPrintStyleChangeCommand;
37 1355 -2                 END
37 1356 --
37 1357 --             ELSE            { User did not specify a change }
37 1358 --                 aPrintStyleChangeCommand.Free;
37 1359 --
37 1360 -1             END;
37 1361 --         proceed := react;
37 1362 -0 A   END;
37 1363 --
37 1364 --
37 1365 --     {$S PrintActual}
37 1366 --     {-----------------------------------+
37 1367 --     |   PosePrintDialog                 |
37 1368 --     +-----------------------------------}
37 1369 -- A   PROCEDURE TStdPrintHandler.PosePrintDialog;
37 1370 --     VAR
37 1371 --         dlgNumber:      INTEGER;
37 1372 --         docName:        Str255;
37 1373 --         itemNo:         INTEGER;
37 1374 --         theItem:        Handle;
37 1375 --         itemType:       INTEGER;
37 1376 --         box:            Rect;
37 1377 --         itemText:       Str255;
37 1378 --         preDocName,
37 1379 --         constTitle:     INTEGER;
37 1380 --
37 1381 0- A   BEGIN
37 1382 --         IF gFinderPrinting THEN
37 1383 1-             BEGIN
37 1384 --             dlgNumber := phFinderPrintDialog;
37 1385 --             itemNo := 3;
```

```
37 1386 -1              END
37 1387 --          ELSE
37 1388 1-              BEGIN
37 1389 --              dlgNumber := phSpoolPrintDialog;
37 1390 --              itemNo := 2;
37 1391 -1              END;
37 1392 --          gPrintDialog := GetNewDialog(dlgNumber, NIL, POINTER(-1));
37 1393 --
37 1394 --          { Substitute the document name for '<<<>>>' in 'Document "<<<>>>" is being printed'.
37 1395 --              ParamText would be a lot simpler, but the Print Mgr. also uses ParamText, and
37 1396 --              the substitution doesn't happen until draw time.}
37 1397 --          GetDocName(docName);
37 1398 --          GetDItem(gPrintDialog, ItemNo, itemType, theItem, box);
37 1399 --          IF theItem <> NIL THEN
37 1400 1-              BEGIN
37 1401 --              GetIText(theItem, itemText);
37 1402 --              IF ParseTitleTemplate(itemText, preDocName, constTitle) &
37 1403 --                 SubstituteInTitle(itemText, docName, preDocName, constTitle) THEN
37 1404 --                  SetIText(theItem, itemText);
37 1405 -1              END;
37 1406 --
37 1407 --      {$IFC qDebug}
37 1408 --          IF gPrintDialog = NIL THEN
37 1409 1-              BEGIN
37 1410 --              ProgramBreak('TStdPrintHandler.PosePrintDialog: Could not load print dialog.');
37 1411 --              EXIT(PosePrintDialog);
37 1412 -1              END;
37 1413 --      {$ENDC}
37 1414 --
37 1415 --          THPrint(fHPrint)^^.prJob.pIdleProc := @CheckButton;
37 1416 --          SetWTitle(gPrintDialog, docName);   { In case Print Mgr. needs it. }
37 1417 --          DrawDialog(gPrintDialog);
37 1418 -0 A    END;
37 1419 --
37 1420 --
37 1421 --      {$S PrintRes}
37 1422 --      { ???Was in PrintActual.  Doesn't really belong in this segment, but
37 1423 --          can get unloaded during Finder printing otherwise. }
37 1424 --      {-----------------------------------+
37 1425 --      |   Print                           |
37 1426 --      +-----------------------------------}
37 1427 -- A    FUNCTION  TStdPrintHandler.Print(itsCmdNumber: CmdNumber; VAR proceed: BOOLEAN): TCommand; OVERRIDE;
37 1428 --      VAR
37 1429 --          fi:             FailInfo;
37 1430 --
37 1431 --          {-------------------------------+
37 1432 --          |   DoPrint                     |
37 1433 --          +-------------------------------}
37 1434 -- B        PROCEDURE DoPrint;
37 1435 --          VAR
37 1436 --              firstPage:          INTEGER;
37 1437 --              lastPage:           INTEGER;
37 1438 --              err:                OSErr;
37 1439 --              spoolFileName:      Str255;
37 1440 --              spoolVRefNum:       INTEGER;
37 1441 --              aPrJob:             TPrJob;
37 1442 --              totalPages:         INTEGER;
37 1443 --              lastPrinted:        INTEGER;
37 1444 --              lastAttempted:      INTEGER;
37 1445 --              pagesPerSubjob:     INTEGER;
37 1446 --              firstSubjobPage:    INTEGER;
37 1447 --              proceed:            BOOLEAN;
37 1448 --              ranOutOfSpace:      BOOLEAN;
37 1449 --              spoolMethod:        BOOLEAN;
37 1450 --              justSpool:          BOOLEAN;
37 1451 --              dontCare:           BOOLEAN;
37 1452 --              pageStrips:         Point;
37 1453 --              fi:                 FailInfo;
37 1454 --
37 1455 --          {-------------------------------+
37 1456 --          |   HdlSubjobFailure            |
37 1457 --          +-------------------------------}
37 1458 -- C        PROCEDURE HdlSubjobFailure(error: OSErr; message: LONGINT);
37 1459 0- C        BEGIN
37 1460 --              BanishPrintDialog;
37 1461 -0 C        END;
37 1462 --
37 1463 0- B        BEGIN
37 1464 --              justSpool := (itsCmdNumber = cPrintToFile);
37 1465 --              proceed := TRUE;
37 1466 --              ranOutOfSpace := FALSE;
37 1467 --              aPrJob := THPrint(fHPrint)^^.prJob;
37 1468 --              fView.DoCalcPageStrips(pageStrips);
37 1469 --              fPageStrips := pageStrips;
37 1470 --
37 1471 --              firstPage := Max(aPrJob.iFstPage, fStartPage);
37 1472 --              lastPage :=  Min(aPrJob.iLstPage, MaxPageNumber);
37 1473 --
37 1474 --              IF lastPage < firstPage THEN
37 1475 --                  err := Alert(phNoPages, NIL)
37 1476 --              ELSE
37 1477 1-                  BEGIN
37 1478 --                  totalPages := lastPage - firstPage + 1;
37 1479 --                  spoolMethod := (aPrJob.BJDocLoop = BSpoolLoop);
37 1480 --                  IF spoolMethod THEN
37 1481 2-                      BEGIN
37 1482 --                      ChooseSpoolFile(spoolFileName, spoolVRefNum, pagesPerSubjob);
37 1483 --                      IF NOT justSpool THEN  { if justSpool is true, then the spool filename & vRefNum will
37 1484 --                                               already have been stuffed into the prJob record before
```

```
37 1485 --                                                  this method is called }
37 1486 3-                          BEGIN
37 1487 --                          IF spoolFileName <> '' THEN
37 1488 --                              WITH THPrint(fHPrint)^^.prJob DO
37 1489 4-                                  BEGIN
37 1490 --                                  pFileName := @spoolFileName;
37 1491 --                                  iFileVol := spoolVRefNum;
37 1492 -4                                  END;
37 1493 -3                          END;
37 1494 -2                      END
37 1495 --                  ELSE
37 1496 --                      pagesPerSubjob := MAXINT;
37 1497 --
37 1498 --                  lastPrinted := firstPage - 1;
37 1499 --
37 1500 --                  pagesPerSubjob := Min(pagesPerSubjob, totalPages);
37 1501 --                  PosePrintDialog;
37 1502 --
37 1503 --                  CatchFailures(fi, HdlSubjobFailure);
37 1504 2-                  REPEAT
37 1505 --                      firstSubjobPage := lastPrinted + 1;
37 1506 --                      Print := OneSubJob(firstSubjobPage,
37 1507 --                          firstSubjobPage + pagesPerSubjob - 1,
37 1508 --                          justSpool, (pagesPerSubjob < totalPages), ranOutOfSpace,
37 1509 --                          lastAttempted, proceed);
37 1510 --                      IF proceed THEN
37 1511 --                          lastPrinted := lastAttempted;
37 1512 --                      IF ranOutOfSpace THEN
37 1513 3-                          BEGIN
37 1514 --                          pagesPerSubjob := lastAttempted - 1 - firstSubjobPage;
37 1515 --                          proceed := TRUE;
37 1516 -3                          END;
37 1517 -2                  UNTIL
37 1518 --                      (lastPrinted = lastPage) OR (pagesPerSubjob < 1) OR NOT proceed;
37 1519 --
37 1520 --                  IF pagesPerSubjob < 1 THEN
37 1521 --                      Failure(errSpooling, 0);
37 1522 --                  Success(fi);
37 1523 --                  BanishPrintDialog; { having removed, put back here, unsure if exactly right }
37 1524 -1                  END;  { if there are pages within requested range }
37 1525 --
37 1526 -0 B       END;
37 1527 --
37 1528 --          {-------------------------------+
37 1529 --          |   HdlPrintFailure             |
37 1530 --          +-------------------------------}
37 1531 -- B       PROCEDURE HdlPrintFailure(error: OSErr; message: LONGINT);
37 1532 0- B       BEGIN
37 1533 --              { Certain Print Manager errors should not result in any alert,
37 1534 --                  since the Print Manager will have already put one up. }
37 1535 --              IF (error >= -8160) AND (error <= -8150) THEN
37 1536 --                  Failure(0, msgPrintFailed);
37 1537 --              IF message = 0 THEN
37 1538 --                  GetDocName(gErrorParm3);
37 1539 --              FailNewMessage(error, message, msgPrintFailed);
37 1540 -0 B       END;
37 1541 --
37 1542 0- A   BEGIN { TStdPrintHandler.Print }
37 1543 --          Print := gNoChanges;
37 1544 --          gCancelAllPrinting := FALSE;
37 1545 --
37 1546 --          SetPrintExtent;     { Make sure we've got the right area. }
37 1547 --
37 1548 --          CatchFailures(fi, HdlPrintFailure);
37 1549 --          DoInMacPrint(DoPrint);
37 1550 --          Success(fi);
37 1551 --          proceed := NOT gCancelAllPrinting;
37 1552 -0 A   END;
37 1553 --
37 1554 --
37 1555 --      {$S PrintNonRes}
37 1556 --      {-----------------------------------+
37 1557 --      |   PrinterChanged                  |
37 1558 --      +-----------------------------------}
37 1559 -- A   PROCEDURE TStdPrintHandler.PrinterChanged; OVERRIDE;
37 1560 0- A   BEGIN
37 1561 --          fView.DoPagination;
37 1562 -0 A   END;
37 1563 --
37 1564 --
37 1565 --      {$S PrintImage}
37 1566 --      {-----------------------------------+
37 1567 --      |   PrintPage                       |
37 1568 --      +-----------------------------------}
37 1569 -- A   PROCEDURE TStdPrintHandler.PrintPage(aPageNumber: INTEGER);  { Print a single page }
37 1570 --      VAR
37 1571 --      {$IFC qDebug}
37 1572 --          aPort:  GrafPtr;
37 1573 --      {$ENDC}
37 1574 --          fi:     FailInfo;
37 1575 --
37 1576 --          {-------------------------------+
37 1577 --          |   CheckPrintFailure           |
37 1578 --          +-------------------------------}
37 1579 -- B       PROCEDURE CheckPrintFailure;
37 1580 0- B       BEGIN
37 1581 --              FailOSErr(PrError);
37 1582 -0 B       END;
37 1583 --
```

```
37 1584 --          {-------------------------------+
37 1585 --          |   HdlPrintFailure             |
37 1586 --          +-------------------------------}
37 1587 -- B        PROCEDURE HdlPrintFailure(error: OSErr; message: LONGINT);
37 1588 0- B        BEGIN
37 1589 --              PrClosePage(gPPrPort);
37 1590 -0 B        END;
37 1591 --
37 1592 0- A    BEGIN
37 1593 --          SetPage(aPageNumber); { gets gPage set up correctly for coordinate transformations }
37 1594 --          CatchFailures(fi, HdlPrintFailure);
37 1595 --          PrOpenPage(gPPrPort, NIL);
37 1596 --          CheckPrintFailure;
37 1597 --          FocusOnInterior;
37 1598 --          DrawPageInterior;
37 1599 --      {$IFC qDebug}
37 1600 --          GetPort(aPort);
37 1601 --          IF aPort <> GrafPtr(gPPrPort) THEN
37 1602 --              ProgramBreak('The view''s DrawPageInterior method changed the grafPort');
37 1603 --      {$ENDC}
37 1604 --          CheckPrintFailure;
37 1605 --          FocusOnBorder;
37 1606 --          AdornPage;
37 1607 --          CheckPrintFailure;
37 1608 --          Success(fi);
37 1609 --          PrClosePage(gPPrPort);
37 1610 -0 A    END;
37 1611 --
37 1612 --
37 1613 --      {$S PrintNonRes}
37 1614 --      {-----------------------------------+
37 1615 --      |   RedoPageBreaks                  |
37 1616 --      +-----------------------------------}
37 1617 -- A    PROCEDURE TStdPrintHandler.RedoPageBreaks; OVERRIDE;
37 1618 --          { Called from fView.DoPagination. }
37 1619 --
37 1620 --      VAR
37 1621 --          worryAboutBreaks:   BOOLEAN;
37 1622 --          oldViewPerPage,
37 1623 --          viewPerPage:        VPoint;
37 1624 --          pageStrips:         Point;
37 1625 --          newInterior,
37 1626 --          oldInterior:        Rect;
37 1627 0- A    BEGIN
37 1628 --          worryAboutBreaks := (fView.GetGrafPort <> NIL) AND gInitialized AND (fShowBreaks OR gDebugPrinting);
37 1629 --
37 1630 --          IF worryAboutBreaks THEN
37 1631 --              InvalPageFeedback; { invalidate old page breaks, if relevant }
37 1632 --
37 1633 --          SetPrintExtent;
37 1634 --          oldInterior := fPageAreas.theInterior;
37 1635 --          oldViewPerPage := fViewPerPage;
37 1636 --          SetMargins;
37 1637 --
37 1638 --              { computes view per page from papersize, printer resolution, margins desired, view resolution,
37 1639 --                  and, if desired, other factors such as printable rectangle of page and font metrics in the
37 1640 --                  printer space }
37 1641 --          fView.DoCalcViewPerPage(viewPerPage);
37 1642 --          fViewPerPage := viewPerPage;
37 1643 --          SetPageInterior(kUsualPages);
37 1644 --          newInterior := fPageAreas.theInterior;
37 1645 --
37 1646 --      {$IFC qDebug}
37 1647 --          IF gDebugPrinting THEN
37 1648 --              IF NOT EqualRect(oldInterior, newInterior) THEN
37 1649 --                  ProgramBreak('Setting new interior');
37 1650 --      {$ENDC}
37 1651 --
37 1652 --          IF NOT EqualRect(oldInterior, newInterior) THEN
37 1653 --              fView.PageInteriorChanged(newInterior);
37 1654 --
37 1655 --      {$IFC qDebug}
37 1656 --          IF gDebugPrinting THEN
37 1657 --              IF NOT EqualVPt(oldViewPerPage, viewPerPage) THEN
37 1658 --                  ProgramBreak('Setting new view per page');
37 1659 --      {$ENDC}
37 1660 --
37 1661 --          IF NOT EqualRect(oldInterior, newInterior) |
37 1662 --             NOT EqualVPt(oldViewPerPage, viewPerPage) THEN
37 1663 --              fView.AdjustSize;
37 1664 --
37 1665 --          fView.DoCalcPageStrips(pageStrips);
37 1666 --          fPageStrips := pageStrips;
37 1667 --
37 1668 --          IF worryAboutBreaks THEN
37 1669 --              InvalPageFeedback; { force redraw of new page breaks, if relevant }
37 1670 --
37 1671 -0 A    END;
37 1672 --
37 1673 --
37 1674 --      {$S PrintNonRes}
37 1675 --      {-----------------------------------+
37 1676 --      |   Reset                           |
37 1677 --      +-----------------------------------}
37 1678 -- A    PROCEDURE TStdPrintHandler.Reset;
37 1679 --          { subclass may stuff some special values by redefining this }
37 1680 --
37 1681 --      LABEL
37 1682 --          1000;
```

```
37 1683 --
37 1684 --      TYPE
37 1685 --          TXWord = PACKED RECORD
37 1686 --              CASE INTEGER OF
37 1687 --                  0:  (c1, c0: CHAR);
37 1688 --                  1:  (b1, b0: SignedByte);
37 1689 --                  2:  (f15, f14, f13, f12, f11, f10, f9, f8, f7, f6, f5, f4, f3, f2, f1, f0: BOOLEAN);
37 1690 --                  3:  (i0: INTEGER);
37 1691 --              END;
37 1692 --
37 1693 --      VAR
37 1694 --          anHPrint:       THPrint;
37 1695 --          didChange:      BOOLEAN;
37 1696 --          inited:         BOOLEAN;
37 1697 --          fi:             FailInfo;
37 1698 --
37 1699 --          {-------------------------------+
37 1700 --          |   CallPrintDefault            |
37 1701 --          +-------------------------------}
37 1702 -- B       PROCEDURE CallPrintDefault;
37 1703 0- B           BEGIN
37 1704 --              PrintDefault(anHPrint);
37 1705 -0 B           END;
37 1706 --
37 1707 --          {-------------------------------+
37 1708 --          |   HdlDefaultError             |
37 1709 --          +-------------------------------}
37 1710 -- B       PROCEDURE HdlDefaultError(error: OSErr; message: LONGINT);
37 1711 0- B       BEGIN
37 1712 --              GOTO 1000;
37 1713 -0 B       END;
37 1714 --
37 1715 0- A    BEGIN
37 1716 --          anHPrint := THPrint(fHPrint);
37 1717 --          inited := FALSE;
37 1718 --          IF anHPrint <> NIL THEN
37 1719 1-              BEGIN
37 1720 --              IF gCouldPrint THEN
37 1721 2-                  BEGIN
37 1722 --                  CatchFailures(fi, HdlDefaultError);
37 1723 --                  DoInMacPrint(CallPrintDefault);
37 1724 --                  IF fSquareDots THEN
37 1725 --                      WITH TXWord(THPrint(anHPrint)^^.prStl.wDev) DO
37 1726 --                        IF bl = 1 THEN { 1 = bDevCItoh => ImageWriter }
37 1727 3-                          BEGIN
37 1728 --                          { Set the square-pixel flag in the print record, then
37 1729 --                              ensure we didn't corrupt it. }
37 1730 --                          f2 := TRUE;
37 1731 --                          ValidatePrintRecord(didChange);
37 1732 -3                          END;
37 1733 --                  inited := TRUE;
37 1734 --                  Success(fi);
37 1735 -2                  END;
37 1736 --      1000:
37 1737 --              IF NOT (gCouldPrint AND inited) THEN
37 1738 --                  { MacPrint code unavailable, but we want some plausible things
37 1739 --                      anyway - set up for Portrait Tall Adj. }
37 1740 2-                  BEGIN
37 1741 --                  { ??? Replace the following expensive stuffing code with StuffHex
37 1742 --                      calls or a GetResource or some such, later }
37 1743 --                  WITH anHPrint^^ DO  {$H-}
37 1744 3-                      BEGIN
37 1745 --                      iPrVersion := 0; { something invalid }
37 1746 --
37 1747 --                      WITH prInfo DO
37 1748 4-                          BEGIN
37 1749 --                          iHRes := 72;
37 1750 --                          iVRes := 72;
37 1751 --                          SetRect(rPage, 0, 0, 576, 752); { must have its top left @ (0,0) }
37 1752 -4                          END;
37 1753 --
37 1754 --                      SetRect(rPaper, -18, -36, 594, 756);
37 1755 --
37 1756 --                      WITH prStl DO
37 1757 4-                          BEGIN
37 1758 --                          iPageV := 1320; { 11 inches in 120th of an inch }
37 1759 --                          iPageH := 1020; { 8.5 inches in 120th of an inch }
37 1760 -4                          END;
37 1761 -3                      END;  {$H+}
37 1762 -2                  END;    { IF NOT (gCouldPrint AND inited) }
37 1763 --
37 1764 -1              END; { IF anHPrint <> NIL }
37 1765 -0 A    END;
37 1766 --
37 1767 --
37 1768 --      {$S PrintNonRes}
37 1769 --      {-----------------------------------+
37 1770 --      |   SetPrintExtent                  |
37 1771 --      +-----------------------------------}
37 1772 -- A    PROCEDURE TStdPrintHandler.SetPrintExtent;
37 1773 --      VAR
37 1774 --          printExtent:    VRect;
37 1775 0- A    BEGIN
37 1776 --          fView.GetPrintExtent(printExtent);
37 1777 --          { Make sure the bottom or right is not smaller than the top or left.
37 1778 --              This can happen when views are being created and before the window
37 1779 --              has been resized to its actual size. }
37 1780 --          printExtent.bottom := Max(printExtent.bottom, printExtent.top);
37 1781 --          printExtent.right :=  Max(printExtent.right, printExtent.left);
```

```
37 1782 --          fPrintExtent := printExtent;
37 1783 -0 A    END;
37 1784 --
37 1785 --
37 1786 --      {$S PrintOpen}
37 1787 --      {-----------------------------------+
37 1788 --      |   SetDefaultPrintInfo             |
37 1789 --      +-----------------------------------}
37 1790 -- A    PROCEDURE TStdPrintHandler.SetDefaultPrintInfo;
37 1791 --      VAR
37 1792 --          didChange:      BOOLEAN;
37 1793 --          wantValidate:   BOOLEAN;
37 1794 --          anHPrint:       THPrint;
37 1795 --          haveHPrint:     BOOLEAN;
37 1796 0- A    BEGIN
37 1797 --          anHPrint := THPrint(fHPrint);
37 1798 --          IF anHPrint <> NIL THEN
37 1799 --              DisposHandle(Handle(anHPrint));
37 1800 --          wantValidate := FALSE;
37 1801 --
37 1802 --          haveHPrint := FALSE;
37 1803 --          IF fView <> NIL THEN
37 1804 1-              BEGIN
37 1805 --              IF fDocument <> NIL THEN
37 1806 --                  IF fDocument.fSharePrintInfo AND (fDocument.fPrintInfo <> NIL) THEN
37 1807 2-                      BEGIN
37 1808 --                      fHPrint := fDocument.fPrintInfo;
37 1809 --                      haveHPrint := TRUE;
37 1810 -2                      END;
37 1811 -1              END;
37 1812 --
37 1813 --          IF haveHPrint THEN
37 1814 --              wantValidate := TRUE
37 1815 --          ELSE
37 1816 1-              BEGIN
37 1817 --              fHPrint := NewPermHandle(SIZEOF(TPrint));
37 1818 --              FailNIL(fHPrint);
37 1819 --              Reset;
37 1820 -1              END;
37 1821 --
37 1822 --          IF wantValidate THEN
37 1823 --              ValidatePrintRecord(didChange);
37 1824 -0 A    END;
37 1825 --
37 1826 --
37 1827 --      {$S PrintNonRes}
37 1828 --      {-----------------------------------+
37 1829 --      |   SetMargins                      |
37 1830 --      +-----------------------------------}
37 1831 -- A    PROCEDURE TStdPrintHandler.SetMargins;
37 1832 --      VAR
37 1833 --          someMargins:    Rect;
37 1834 0- A    BEGIN
37 1835 --          someMargins := fPageAreas.theMargins;           { Prevent heap scramble }
37 1836 --          InstallMargins(someMargins, fMinimalMargins);
37 1837 -0 A    END;
37 1838 --
37 1839 --
37 1840 --      {$S PrintImage}
37 1841 --      {-----------------------------------+
37 1842 --      |   SetPage                         |
37 1843 --      +-----------------------------------}
37 1844 -- A    PROCEDURE TStdPrintHandler.SetPage(aPageNumber: INTEGER);
37 1845 --      VAR
37 1846 --          viewedRect: VRect;
37 1847 --          vhs:        VHSelect;
37 1848 --          strip:      Point;
37 1849 0- A    BEGIN
37 1850 --          fFocusedPage := aPageNumber;
37 1851 --
37 1852 --          strip := PageToStrip(aPageNumber);
37 1853 --          FOR vhs := v TO h DO
37 1854 1-              BEGIN
37 1855 --              GetBreakCoord(gOrthogonal[vhs], strip.vh[vhs], viewedRect.topLeft.vh[vhs]);
37 1856 --              GetBreakCoord(gOrthogonal[vhs], strip.vh[vhs] + 1, viewedRect.botRight.vh[vhs]);
37 1857 -1              END;
37 1858 --
37 1859 --          SetPageInterior(aPageNumber);
37 1860 --          fView.DoSetPageOffset(viewedRect.topLeft);
37 1861 --          fViewedRect := viewedRect;
37 1862 --
37 1863 --      {$IFC qDebug}
37 1864 --          IF gDebugPrinting THEN
37 1865 1-              BEGIN
37 1866 --              Write('pg #: ', aPageNumber:1);
37 1867 --              Write(':  coord = ');
37 1868 --              WriteVPt(viewedRect.topLeft);
37 1869 --              Write('Page Interior: ');
37 1870 --              WriteRect(fPageAreas.theInterior);{ ???HS }
37 1871 --              WriteLn;
37 1872 -1              END;
37 1873 --      {$ENDC}
37 1874 -0 A    END;
37 1875 --
37 1876 --
37 1877 --      {$S PrintNonRes}
37 1878 --      {-----------------------------------+
37 1879 --      |   SetPageInterior                 |
37 1880 --      +-----------------------------------}
```

```
37 1881 -- A   PROCEDURE TStdPrintHandler.SetPageInterior(pageNumber: INTEGER); OVERRIDE;
37 1882 --     VAR
37 1883 --         vhs:        VHSelect;
37 1884 --         pegPoint:   Point;
37 1885 0- A   BEGIN
37 1886 --         WITH fPageAreas, theInterior DO
37 1887 1-             BEGIN
37 1888 --             FOR vhs := v TO h DO
37 1889 2-                 BEGIN
37 1890 --                 topLeft.vh[vhs] := thePaper.topLeft.vh[vhs] + theMargins.topLeft.vh[vhs];
37 1891 --                 botRight.vh[vhs] := topLeft.vh[vhs] + fViewPerPage.vh[vhs];
37 1892 -2                 END;
37 1893 -1             END;
37 1894 --
37 1895 --         LocatePageInterior(pageNumber, pegPoint);
37 1896 --         WITH fPageAreas.theInterior DO
37 1897 1-             BEGIN
37 1898 --             topLeft := pegPoint;
37 1899 --             FOR vhs := v TO h DO
37 1900 --                 botRight.vh[vhs] := topLeft.vh[vhs] + fViewPerPage.vh[vhs];
37 1901 -1             END;
37 1902 -0 A   END;
37 1903 --
37 1904 --
37 1905 --     {$S PrintImage}
37 1906 --     {-----------------------------------+
37 1907 --     |   SetPageOffset                   |
37 1908 --     +-----------------------------------}
37 1909 -- A   PROCEDURE TStdPrintHandler.SetPageOffset(coord: VPoint); OVERRIDE;
37 1910 --     VAR
37 1911 --         vhs:    VHSelect;
37 1912 0- A   BEGIN
37 1913 --         FOR vhs := v to h DO
37 1914 --             gPageOffset.vh[vhs] := coord.vh[vhs] -
37 1915 --                                    fPageAreas.theInterior.topLeft.vh[vhs];
37 1916 -0 A   END;
37 1917 --
37 1918 --
37 1919 --     {$S PrintFinder}
37 1920 --     {-----------------------------------+
37 1921 --     |   SetupForFinder                  |
37 1922 --     +-----------------------------------}
37 1923 -- A   FUNCTION  TStdPrintHandler.SetupForFinder: BOOLEAN; OVERRIDE;
37 1924 --     VAR
37 1925 --         proceed:        BOOLEAN;
37 1926 --         didChange:      BOOLEAN;
37 1927 0- A   BEGIN
37 1928 --         proceed := TRUE;
37 1929 --         CheckPrinter;       { Uncertain if necessary??? }
37 1930 --
37 1931 --         IF fFinderSetup THEN
37 1932 --             IF PosePageSetupDialog(proceed, FALSE) <> gNoChanges THEN
37 1933 --     {$IFC qDebug}
37 1934 --                 ProgramBreak('PosePageSetupDialog returned real command.')
37 1935 --     {$ENDC}     ;
37 1936 --
37 1937 --         IF proceed THEN
37 1938 1-             BEGIN
37 1939 --             ShowDocBeingPrinted(TRUE);
37 1940 --
37 1941 --             IF fFinderJobDialog OR (gFinderHPrint = NIL) THEN
37 1942 2-                 BEGIN
37 1943 --                 proceed := PoseJobDialog;
37 1944 --                 { Merge into gPrintHandler, in case next document needs it }
37 1945 --                 IF gFinderHPrint = NIL THEN
37 1946 3-                     BEGIN
37 1947 --                     gFinderHPrint := NewPermHandle(SIZEOF(TPrint));
37 1948 --                     FailNIL(gFinderHPrint);
37 1949 -3                     END;
37 1950 --                 BlockMove(fHPrint^, gFinderHPrint^, SIZEOF(TPrint));
37 1951 -2                 END
37 1952 --             ELSE
37 1953 2-                 BEGIN
37 1954 --                 PrJobMerge(THPrint(gFinderHPrint), THPrint(fHPrint));
37 1955 --                 ValidatePrintRecord(didChange);
37 1956 -2                 END;
37 1957 --
37 1958 --             ShowDocBeingPrinted(FALSE);
37 1959 -1             END;
37 1960 --         SetupForFinder := proceed;
37 1961 -0 A   END;
37 1962 --
37 1963 --
37 1964 --     {$S PrintRes}
37 1965 --     {-----------------------------------+
37 1966 --     |   SetupPrintOne                   |
37 1967 --     +-----------------------------------}
37 1968 -- A   FUNCTION  TStdPrintHandler.SetupPrintOne: BOOLEAN;
37 1969 --     VAR
37 1970 --         didChange:      BOOLEAN;
37 1971 0- A   BEGIN
37 1972 --         { Call PrValidate to 1) make sure the record is OK, and
37 1973 --             2) Get the LaserWriter driver to get the name of the
37 1974 --             front window. }
37 1975 --         ValidatePrintRecord(didChange);
37 1976 --
37 1977 --         PrSetError(noErr);
37 1978 --         WITH THPrint(fHPrint)^^.prJob DO
37 1979 1-             BEGIN
```

```
37 1980 --              iFstPage := 0;
37 1981 --              iLstPage := 9999;       { This is what the Print Mgr. uses }
37 1982 -1              END;
37 1983 --          SetupPrintOne := TRUE;      { Should always be able to continue from here }
37 1984 -0 A    END;
37 1985 --
37 1986 --
37 1987 --      {$S PrintFinder}
37 1988 --      {-----------------------------------+
37 1989 --      |   ShowDocBeingPrinted             |
37 1990 --      +-----------------------------------}
37 1991 -- A    PROCEDURE TStdPrintHandler.ShowDocBeingPrinted (entering: BOOLEAN);
37 1992 --      VAR
37 1993 --          aTitle:     Str255;
37 1994 0- A    BEGIN
37 1995 --          IF entering THEN
37 1996 1-              BEGIN
37 1997 --              gPrintDialog := GetNewDialog(phWhichDoc, NIL, POINTER(-1));
37 1998 --              IF gPrintDialog <> NIL THEN
37 1999 2-                  BEGIN
37 2000 --                  GetDocName(aTitle);
37 2001 --                  SetWTitle(gPrintDialog, aTitle);
37 2002 --                  DrawDialog(gPrintDialog);
37 2003 -2                  END
37 2004 --      {$IFC qDebug}
37 2005 --              ELSE
37 2006 --                  ProgramBreak('TStdPrintHandler.ShowDocBeingPrinted: Unable to load dialog.')
37 2007 --      {$ENDC}                     { semicolon };
37 2008 -1              END
37 2009 --          ELSE
37 2010 --              BanishPrintDialog;
37 2011 -0 A    END;
37 2012 --
37 2013 --
37 2014 --      {$S MARes}
37 2015 --      {-----------------------------------+
37 2016 --      |   ShowsOnScreen                   |
37 2017 --      +-----------------------------------}
37 2018 -- A    FUNCTION  TStdPrintHandler.ShowsOnScreen: BOOLEAN;
37 2019 0- A    BEGIN
37 2020 --          IF fView <> NIL THEN
37 2021 --              ShowsOnScreen := fView.GetWindow <> NIL
37 2022 --          ELSE
37 2023 --              ShowsOnScreen := FALSE;
37 2024 -0 A    END;
37 2025 --
37 2026 --
37 2027 --      {$S PrintRes}
37 2028 --      {-----------------------------------+
37 2029 --      |   StripToPage                     |
37 2030 --      +-----------------------------------}
37 2031 -- A    FUNCTION  TStdPrintHandler.StripToPage (hStrip, vStrip: INTEGER): INTEGER;
37 2032 --
37 2033 --      VAR
37 2034 --          strip:      Point;
37 2035 --          ortho:      VHSelect;
37 2036 0- A    BEGIN
37 2037 --          SetPt(strip, hStrip, vStrip);
37 2038 --          ortho := gOrthogonal[fPageDirection];
37 2039 --          StripToPage := strip.vh[fPageDirection] * fPageStrips.vh[ortho] +
37 2040 --                         strip.vh[ortho] + fStartPage;
37 2041 -0 A    END;
37 2042 --
37 2043 --
37 2044 --      {$S PrintTerminate}
37 2045 --      {-----------------------------------+
37 2046 --      |   Terminate                       |
37 2047 --      +-----------------------------------}
37 2048 -- A    PROCEDURE TStdPrintHandler.Terminate; OVERRIDE;
37 2049 0- A    BEGIN
37 2050 --          EmpowerChooser(TRUE); { reenable Chooser, in case it was disabled }
37 2051 -0 A    END;
37 2052 --
37 2053 --
37 2054 --      {$S PrintRes}
37 2055 --      {-----------------------------------+
37 2056 --      |   ValidatePrintRecord             |
37 2057 --      +-----------------------------------}
37 2058 -- A    PROCEDURE TStdPrintHandler.ValidatePrintRecord(VAR didChange: BOOLEAN);
37 2059 --      VAR
37 2060 --          fi:     FailInfo;
37 2061 --
37 2062 --          {--------------------------------+
37 2063 --          |   CallPrValidate               |
37 2064 --          +--------------------------------}
37 2065 -- B        PROCEDURE CallPrValidate;
37 2066 0- B        BEGIN
37 2067 --              didChange := PrValidate(THPrint(fHPrint));
37 2068 -0 B        END;
37 2069 --
37 2070 --          {--------------------------------+
37 2071 --          |   HdlValidateFailed            |
37 2072 --          +--------------------------------}
37 2073 -- B        PROCEDURE HdlValidateFailed(error: OSErr; message: LONGINT);
37 2074 0- B        BEGIN
37 2075 --              Reset;
37 2076 --              EXIT(ValidatePrintRecord);
37 2077 -0 B        END;
37 2078 --
```

```
37 2079 0- A   BEGIN
37 2080 --         CatchFailures(fi, HdlValidateFailed);
37 2081 --         DoInMacPrint(CallPrValidate);
37 2082 --         Success(fi);
37 2083 -0 A   END;
37 2084 --
37 2085 --     (*****************************************************************************************)
37 2086 --     (*      T P r i n t S t y l e C h a n g e C o m m a n d                                  *)
37 2087 --     (*****************************************************************************************)
37 2088 --     {$S PrintDoCommand}
37 2089 --     {-----------------------------------+
37 2090 --     |   IPrintStyleChangeCommand        |
37 2091 --     +-----------------------------------}
37 2092 -- A   PROCEDURE TPrintStyleChangeCommand.IPrintStyleChangeCommand(itsPrintHandler: TStdPrintHandler);
37 2093 --     VAR
37 2094 --         fi:     FailInfo;
37 2095 --
37 2096 --         {-------------------------------+
37 2097 --         |   HdlInitFailed               |
37 2098 --         +-------------------------------}
37 2099 -- B       PROCEDURE HdlInitFailed(error: OSErr; message: LONGINT);
37 2100 0- B       BEGIN
37 2101 --             Free;
37 2102 -0 B       END;
37 2103 --
37 2104 0- A   BEGIN
37 2105 --         fStdPrintHandler := itsPrintHandler;
37 2106 --         fOldHPrint := NIL;
37 2107 --         fNewHPrint := NIL;
37 2108 --         ICommand(cChangePrinterStyle, itsPrintHandler.fDocument, NIL, NIL);
37 2109 --         CatchFailures(fi, HdlInitFailed);
37 2110 --         fOldHPrint := NewPermHandle(SIZEOF(TPrint));
37 2111 --         FailNIL(fOldHPrint);
37 2112 --         { Make a copy of the old version of the PrintInfo record }
37 2113 --         BlockMove(itsPrintHandler.fHPrint^, fOldHPrint^, SIZEOF(TPrint));
37 2114 --         fNewHPrint := NewPermHandle(SIZEOF(TPrint));
37 2115 --         FailNIL(fNewHPrint);
37 2116 --         Success(fi);
37 2117 -0 A   END;
37 2118 --
37 2119 --
37 2120 --     {$S PrintDoCommand}
37 2121 --     {-----------------------------------+
37 2122 --     |   Free                            |
37 2123 --     +-----------------------------------}
37 2124 -- A   PROCEDURE TPrintStyleChangeCommand.Free; OVERRIDE;
37 2125 0- A   BEGIN
37 2126 --         IF fOldHPrint <> NIL THEN
37 2127 --             DisposHandle(fOldHPrint);
37 2128 --         IF fNewHPrint <> NIL THEN
37 2129 --             DisposHandle(fNewHPrint);
37 2130 --         INHERITED Free;
37 2131 -0 A   END;
37 2132 --
37 2133 --
37 2134 --     {$S PrintDoCommand}
37 2135 --     {-----------------------------------+
37 2136 --     |   DoIt                            |
37 2137 --     +-----------------------------------}
37 2138 -- A   PROCEDURE TPrintStyleChangeCommand.DoIt; OVERRIDE;
37 2139 0- A   BEGIN
37 2140 --         fStdPrintHandler.CheckPrinter;  { Will find it changed, and hence dispatch to view's DoPrinterChanged }
37 2141 -0 A   END;
37 2142 --
37 2143 --
37 2144 --     {$IFC qDebug}
37 2145 --     {$S PrintInspect}
37 2146 --     {-----------------------------------+
37 2147 --     |   Fields                          |
37 2148 --     +-----------------------------------}
37 2149 -- A   PROCEDURE TPrintStyleChangeCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
37 2150 --                                                 fieldAddr: Ptr;
37 2151 --                                                 fieldType: INTEGER));
37 2152 0- A   BEGIN
37 2153 --         DoToField('fStdPrintHandler', @fStdPrintHandler, bObject);
37 2154 --         DoToField('fOldHPrint', @fOldHPrint, bHandle);
37 2155 --         DoToField('fNewHPrint', @fNewHPrint, bHandle);
37 2156 --         INHERITED Fields(DoToField);
37 2157 -0 A   END;
37 2158 --     {$ENDC}
37 2159 --
37 2160 --
37 2161 --     {$S PrintDoCommand}
37 2162 --     {-----------------------------------+
37 2163 --     |   UndoIt                          |
37 2164 --     +-----------------------------------}
37 2165 -- A   PROCEDURE TPrintStyleChangeCommand.UndoIt; OVERRIDE;
37 2166 0- A   BEGIN
37 2167 --         BlockMove(fOldHPrint^, fStdPrintHandler.fHPrint^, SIZEOF(TPrint));
37 2168 --         fStdPrintHandler.CheckPrinter; { Will find it changed, and hence dispatch to
37 2169 --                                          view's DoPrinterChanged }
37 2170 -0 A   END;
37 2171 --
37 2172 --
37 2173 --     {$S PrintDoCommand}
37 2174 --     {-----------------------------------+
37 2175 --     |   RedoIt                          |
37 2176 --     +-----------------------------------}
37 2177 -- A   PROCEDURE TPrintStyleChangeCommand.RedoIt; OVERRIDE;
```

```
37 2178 0- A   BEGIN
37 2179 --         BlockMove(fNewHPrint^, fStdPrintHandler.fHPrint^, SIZEOF(TPrint));
37 2180 --         fStdPrintHandler.CheckPrinter; { Will find it changed, and hence dispatch to
37 2181 --                                                 view's DoPrinterChanged }
37 2182 -0 A   END;
37 2183 --
36  549 --
36  550 --     END.
```

```
38    1 --    {$P}
38    2 --    { UTEView.p }
38    3 --    { Copyright © 1986-1988 Apple Computer, Inc.  All rights reserved. }
38    4 --
38    5 --    UNIT UTEView;
38    6 --
38    7 --    (*****************************************************************************************)
38    8 --    INTERFACE
38    9 --    (*****************************************************************************************)
38   10 --
38   11 --    {-----------------------------------+
38   12 --    |   Flags                           |
38   13 --    +-----------------------------------}
38   14 --    {$SC+}  { Use short circuit ops!        }
38   15 --
38   16 --    {-----------------------------------+
38   17 --    |   Uses                            |
38   18 --    +-----------------------------------}
38   19 --    USES
38   20 --        {$LOAD MacIntf.LOAD}
38   21 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
38   22 --        {$LOAD UMacApp.LOAD}
38   23 --            UMAUtil, UViewCoords, UFailure,
38   24 --            UMemory, UMenuSetup, UObject, UList, UAssociation, UMacApp,
38   25 --        {$LOAD}
38   26 --        {$IFC qDebug} UWriteLnWindow, {$ENDC}
38   27 --            Script;
38   28 --
38   29 --    {$IFC qMPW2}    { TextEdit interface that isn't included in MPW 2.x. }
38   30 --    CONST
38   31 --        doToggle    = 32;
38   32 --
38   33 --    TYPE
38   34 --        TEIntHook = (intEOLHook, intDrawHook, intWidthHook, intHitTestHook);
38   35 --
38   36 --    FUNCTION  TEContinuousStyle(VAR mode: INTEGER; VAR aStyle: TextStyle;
38   37 --                                hTE: TEHandle): BOOLEAN;
38   38 --                                INLINE $3F3C, $000A, $A83D;
38   39 --
38   40 --    PROCEDURE SetStylScrap(rangeStart, rangeEnd: LONGINT; newStyles: StScrpHandle;
38   41 --                           redraw: BOOLEAN; hTE: TEHandle);
38   42 --                           INLINE $3F3C, $000B, $A83D;
38   43 --
38   44 --    PROCEDURE TECustomHook(which: TEIntHook; VAR addr: ProcPtr; hTE: TEHandle);
38   45 --                           INLINE $3F3C, $000C, $A83D;
38   46 --
38   47 --    FUNCTION  TENumStyles(rangeStart, rangeEnd: LONGINT; hTE: TEHandle): LONGINT;
38   48 --                           INLINE $3F3C, $000D, $A83D;
38   49 --    {$ENDC qMPW2}
38   50 --
38   51 --    {-----------------------------------+
38   52 --    |   global constants                |
38   53 --    +-----------------------------------}
38   54 --    CONST
38   55 --        kUnlimited      =   MAXINT; { The maximum number of characters in the fText
38   56 --                                      of a TTEView object }
38   57 --
38   58 --        kWithStyle      =   TRUE;   { Parameters to TTEView.ITEView }
38   59 --        kWithoutStyle   =   FALSE;
38   60 --        kSaveCurrentChars = TRUE;   { Parameter to ITECommand }
38   61 --
38   62 --    {-----------------------------------+
38   63 --    |   global types                    |
38   64 --    +-----------------------------------}
38   65 --    TYPE
38   66 --        {-----------------------------------+
38   67 --        |   TTEView                         |
38   68 --        +-----------------------------------}
38   69 --        TEViewTemplatePtr = ^TEViewTemplate;
38   70 --        TEViewTemplate =    PACKED RECORD
38   71 --                        itsStyleType:       BOOLEAN;
38   72 --                        itsAutoWrap:        BOOLEAN;
38   73 --                        itsAcceptsChanges:  BOOLEAN;
38   74 --                        itsFreesText:       BOOLEAN;
38   75 --                        filler1:            0..4095;
38   76 --                        itsKeyCmdNumber:    INTEGER;
38   77 --                        itsMaxChars:        INTEGER;
38   78 --                        itsInset:           Rect;
38   79 --                        itsJustification:   INTEGER;
38   80 --                        itsTextFace:        Style;
38   81 --                        itsTextSize:        INTEGER;
38   82 --                        itsTextColor:       RGBColor;
38   83 --                        itsFontName:        Str255;     { Actually a variable length P-String   }
38   84 --                        END;
38   85 --
38   86 --        TTEView = OBJECT(TView)
38   87 --
38   88 --        { TTEView is a view subclass representing a TextEdit record.  TextEdit is
38   89 --            the simple text-editing facility built into the Macintosh ROM.  The
38   90 --            purpose of a TEView is to make TextEdit function properly in a MacApp
38   91 --            environment, for things like scrolling, printing, page breaks, and
38   92 --            command handling }
38   93 --
38   94 --        { Fields }
38   95 --
38   96 --            fHTE:               TEHandle;   { Handle to the actual TextEdit object }
38   97 --
38   98 --            fText:              Handle;     { The text in the TEHandle }
```

```
38   99 --              fSavedTEHandle:     Handle;     { Saved handle from TENew }
38  100 --
38  101 --              fInset:             Rect;       { Amount to inset text from edges of view }
38  102 --
38  103 --              fKeyCmdNumber:      CmdNumber;  { Will be used as the string number for
38  104 --                                                  "Undo Typing" }
38  105 --
38  106 --              fMaxChars:          INTEGER;    { Maximum number of chars to accept into
38  107 --                                                  fText.  Default is MAXINT; stuff to
38  108 --                                                  different value if you want to
38  109 --                                                  limit max chars to fewer than
38  110 --                                                  MAXINT }
38  111 --
38  112 --              fLastHeight:        INTEGER;    { Last checked height of record.  If the record is
38  113 --                                                  stylish, this represents the height in pixels.
38  114 --                                                  For old-style records, equals number of lines. }
38  115 --
38  116 --              fTypingCommand:     TTETypingCommand;
38  117 --                                              { The current TE Typing command relating
38  118 --                                                  to me, if any }
38  119 --
38  120 --              fTextStyle:         TextStyle;  { Current style of text }
38  121 --
38  122 --              fJustification:     INTEGER;    { Current justification of text record }
38  123 --
38  124 --              fAcceptsChanges:    BOOLEAN;    { Set to FALSE to have text which will
38  125 --                                                  not accept any change, such as text
38  126 --                                                  on the Clipboard, or perhaps
38  127 --                                                  received mail }
38  128 --              fStyleType:         BOOLEAN;    { Set to kWithStyle if record is styled }
38  129 --              fAutoWrap:          BOOLEAN;    { Set to FALSE for line wrapping at CR's only }
38  130 --              fFreeText:          BOOLEAN;    { Determines if fText should be freed on
38  131 --                                                  Free. }
38  132 --              fSpecsChanged:      BOOLEAN;    { Something recently happened which could effect
38  133 --                                                  font/style/size/color menu item updating.
38  134 --                                                  Should be reset to FALSE when application
38  135 --                                                  has taken the appropriate action. }
38  136 --              fScroller:          TScroller;  { The scroller in which this view is contained }
38  137 --              fLastPageBreak:     INTEGER;    { Caches last page break computed. }
38  138 --              fLastLine:          INTEGER;    { Last line of text of the last page break }
38  139 --          { Initialize and Free }
38  140 --
38  141 --              PROCEDURE TTEView.ITEView(
38  142 --                  itsDocument:        TDocument;      { OK to be NIL, if view will
38  143 --                                                        belong to a documentless
38  144 --                                                        window }
38  145 --                  itsSuperView:       TView;          { The view in which this view is contained }
38  146 --                  itsLocation:        VPoint;         { Location in its superview }
38  147 --                  itsSize:            VPoint;         { Size of view }
38  148 --                  itsHDeterminer:     SizeDeterminer; { How to width of the view is to
38  149 --                                                        be determined }
38  150 --                  itsVDeterminer:     SizeDeterminer; { How the height of the view is
38  151 --                                                        to be determined }
38  152 --                  itsInset:           Rect;           { Amount to inset text }
38  153 --                  itsTextStyle:       TextStyle;      { Initial text style }
38  154 --                  itsJustification:   INTEGER;        { Its justification }
38  155 --                  itsStyleType:       BOOLEAN;        { Whether or not record is styled }
38  156 --                  itsAutoWrap:        BOOLEAN         { FALSE if newline occurs at CR only }
38  157 --                  );
38  158 --
38  159 --          {$IFC qTemplateViews}
38  160 --              PROCEDURE TTEView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
38  161 --                  { Initialize a TTEView via a 'view' resource. }
38  162 --          {$ENDC}
38  163 --
38  164 --          {$IFC qWriteTemplates}
38  165 --              PROCEDURE TTEView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
38  166 --
38  167 --              PROCEDURE TTEView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
38  168 --          {$ENDC}
38  169 --
38  170 --              PROCEDURE TTEView.Free; OVERRIDE;
38  171 --                  { Frees the TERecord and optionally fText, then frees SELF }
38  172 --
38  173 --              PROCEDURE TTEView.MakeTERecord;
38  174 --                  { Called with grafport properly set, to create the actual TE Record }
38  175 --
38  176 --          { Commands and Menus }
38  177 --
38  178 --              FUNCTION  TTEView.DoKeyCommand(ch: Char; aKeyCode: INTEGER;
38  179 --                      VAR info: EventInfo): TCommand; OVERRIDE;
38  180 --                  { Handles keystrokes }
38  181 --
38  182 --              FUNCTION  TTEView.DoMakeEditCommand(aCmdNumber: CmdNumber): TTECommand;
38  183 --                  { Make a command for handling edit menu stuff }
38  184 --
38  185 --              FUNCTION  TTEView.DoMakeStyleCommand(aStyle: TextStyle; itsCmdNumber: CmdNumber;
38  186 --                                  itsMode: INTEGER): TTEStyleCommand;
38  187 --                  { Make a style change command }
38  188 --
38  189 --              FUNCTION  TTEView.DoMakeTypingCommand(ch: Char): TTETypingCommand;
38  190 --                  { Make a typing command for handling keystrokes }
38  191 --
38  192 --              FUNCTION  TTEView.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
38  193 --                  { Handles menu commands }
38  194 --
38  195 --              FUNCTION  TTEView.DoMouseCommand(VAR theMouse: Point;
38  196 --                      VAR info: EventInfo; VAR hysteresis: Point): TCommand; OVERRIDE;
38  197 --                  { Handles mousepresses }
```

```
38  198 --
38  199 --              PROCEDURE TTEView.DoneTyping;
38  200 --                  { No further typing can occur for the current typing command }
38  201 --
38  202 --              PROCEDURE TTEView.DoSetupMenus; OVERRIDE;
38  203 --                  { Sets up menus }
38  204 --
38  205 --              PROCEDURE TTEView.InstallSelection(wasActive, beActive: BOOLEAN); OVERRIDE;
38  206 --                  { Called at activate/deactivate time to get insertion-point-blinking
38  207 --                      right (or to highlight the selection) }
38  208 --
38  209 --          { Screen Display }
38  210 --
38  211 --              PROCEDURE TTEView.DoIdle(phase: IdlePhase); OVERRIDE;
38  212 --                  { Blinks the insertion point }
38  213 --
38  214 --              FUNCTION  TTEView.DoSetCursor(localPoint: Point;
38  215 --                              cursorRgn: RgnHandle): BOOLEAN; OVERRIDE;
38  216 --                  { Sets the cursor to the I-beam }
38  217 --
38  218 --              PROCEDURE TTEView.Draw(area: Rect); OVERRIDE;
38  219 --                  { Draw the text in a frame or on the printed page }
38  220 --
38  221 --              PROCEDURE TTEView.ShowReverted; OVERRIDE;
38  222 --                  { Make sure line starts are correct before redisplay }
38  223 --
38  224 --
38  225 --          { Size Changes }
38  226 --
38  227 --              PROCEDURE TTEView.BeInPort (itsPort: GrafPtr); OVERRIDE;
38  228 --                  { Now that a grafport is established for the view, tells TextEdit about it }
38  229 --
38  230 --              PROCEDURE TTEView.BeInScroller (itsScroller: TScroller); OVERRIDE;
38  231 --                  { Gives us a chance to set the scroll parameters }
38  232 --
38  233 --              PROCEDURE TTEView.CalcMinSize(VAR minSize: VPoint); OVERRIDE;
38  234 --                  { Compute the minimum size of the view }
38  235 --
38  236 --              FUNCTION  TTEView.CalcRealHeight: INTEGER;
38  237 --                  { Calculate true height of record, including last character if it is a
38  238 --                      carriage return }
38  239 --
38  240 --              PROCEDURE TTEView.ComputeSize(VAR newSize: VPoint); OVERRIDE;
38  241 --                  { Compute the actual size of the view }
38  242 --
38  243 --              PROCEDURE TTEView.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
38  244 --                  { Installs the new size for both MacApp and TextEdit }
38  245 --
38  246 --              PROCEDURE TTEView.SynchView (redraw: BOOLEAN);
38  247 --                  { Keeps view metrics in synch after TextEdit operation.  If redraw is
38  248 --                      true then ScrollSelectionIntoView is called to make sure the
38  249 --                      selection is visible.}
38  250 --
38  251 --          { Clipboard }
38  252 --
38  253 --              { NOTE -- These methods apply only to a TEView installed as the view in
38  254 --                          the Clipboard }
38  255 --
38  256 --              FUNCTION  TTEView.ContainsClipType(aType: ResType): BOOLEAN; OVERRIDE;
38  257 --                  { Determines whether the indicated clipboard type can be produced by the view }
38  258 --
38  259 --              FUNCTION  TTEView.GivePasteData(aDataHandle: Handle; dataType: ResType): LONGINT; OVERRIDE;
38  260 --                  { Supplies data to be PASTED by some requesting PASTE command }
38  261 --
38  262 --              PROCEDURE TTEView.WriteToDeskScrap;  OVERRIDE;
38  263 --                  { Produces TEXT data for the Desk Scrap when called upon to do so }
38  264 --
38  265 --          { Miscellaneous }
38  266 --
38  267 --              PROCEDURE TTEView.AutoScrolling (doScrolling: BOOLEAN);
38  268 --                  { Set TERecord auto scrolling to given value via TEAutoScroll }
38  269 --
38  270 --              PROCEDURE TTEView.CalcSelLoc(VAR selectionRect: Rect);
38  271 --                  { Calculates the location of the current Selection }
38  272 --
38  273 --              PROCEDURE TTEView.ChangeWrap (newAutoWrap, redraw: BOOLEAN);
38  274 --                  { Changes auto-wrapping behavior, redrawing if requested }
38  275 --
38  276 --              FUNCTION  TTEView.ContinuousStyle(firstChar, lastChar: INTEGER;
38  277 --                          VAR mode: INTEGER; VAR aStyle: TextStyle): BOOLEAN;
38  278 --                  { Returns TRUE if the style of the characters within the given range is
38  279 --                      wholly continuous.  The style attributes to check are specified by mode,
38  280 --                      which has the same meaning as in the TextEdit call TESetStyle.  Attributes
38  281 --                      which are continuous are returned in aStyle, with the appropriate bits set
38  282 --                      in mode.  Font style attribute bits are individually checked across
38  283 --                      the selection range, and returned set to 1 if that attribute is continuous. }
38  284 --
38  285 --              PROCEDURE TTEView.ExtractStyles(VAR theStyles: TEStyleHandle; VAR theElements: STHandle);
38  286 --                  { Extract handles to style information for a TextEdit record, presumably
38  287 --                      for the purpose of storing the data to disk.  Note that line heights
38  288 --                      can be recalculated, and as such don't need to be stored. }
38  289 --
38  290 --              FUNCTION  TTEView.ExtractText: Handle;
38  291 --                  { For sake of completeness.  Extract handle to text, presumably for the
38  292 --                      purpose of storing the text to disk. }
38  293 --
38  294 --              PROCEDURE TTEView.RecalcText;
38  295 --                  { Tells TextEdit to recompute linestarts }
38  296 --
```

```
38  297 --              PROCEDURE TTEView.ScrollSelectionIntoView;
38  298 --                  { Scrolls the selection into view -- called after Commands are done or undone
38  299 --                    If auto scrolling is on, calls TESelView to do the right thing }
38  300 --
38  301 --              PROCEDURE TTEView.SetJustification(newJust: INTEGER; redraw: BOOLEAN);
38  302 --                  { Sets the TERecord to the given justification.  If redraw is true then
38  303 --                    the view is redrawn by calling ForceRedraw.  Otherwise it is assumed
38  304 --                    the called will get the view redrawn. }
38  305 --
38  306 --              PROCEDURE TTEView.SetOneStyle(theStart, theEnd, theMode: INTEGER;
38  307 --                                            theStyle: TextStyle; redraw: BOOLEAN);
38  308 --                  { Sets a text style continuously over a range.  In unstylish TERecords,
38  309 --                    the style is set across the entire record. If redraw is true then
38  310 --                    the text is redrawn. }
38  311 --
38  312 --              PROCEDURE TTEView.SetText(theText: Str255);
38  313 --                  { Convenience routine, to save from having to create a handle just to pass
38  314 --                    to StuffText.  Note that if fSavedTEHandle already holds a saved handle,
38  315 --                    it will be disposed before StuffText is called. }
38  316 --
38  317 --              FUNCTION  TTEView.SpaceForStyles(rangeStart, rangeEnd: LONGINT): BOOLEAN;
38  318 --                  { Returns TRUE if there is enough memory to hold the styles over the
38  319 --                    given range in the TERecord }
38  320 --
38  321 --              PROCEDURE TTEView.StuffStyles(theStyles: TEStyleHandle; theElements: STHandle);
38  322 --                  { Installs style information, presumably fetched from disk, into the
38  323 --                    TextEdit record. }
38  324 --
38  325 --              PROCEDURE TTEView.StuffText(theText: Handle);
38  326 --                  { Installs text into the TextEdit record.  Note that this destroys any
38  327 --                    existing style information -- StuffStyles should be called if
38  328 --                    styles are to be attached to this text information }
38  329 --
38  330 --              PROCEDURE TTEView.StuffTERects(newTERect: Rect);
38  331 --                  { installs "newTERect" as the dest & view rect of the TextEdit record }
38  332 --
38  333 --          { Printing }
38  334 --
38  335 --              FUNCTION  TTEView.DoBreakFollowing(vhs: VHSelect; prevBreak: VCoordinate;
38  336 --                      VAR automatic: BOOLEAN): VCoordinate; OVERRIDE;
38  337 --                  { Computes the next page-break following 'prevBreak' in direction
38  338 --                    given by 'vhs' }
38  339 --
38  340 --              PROCEDURE TTEView.DoCalcViewPerPage (VAR viewPerPage: VPoint); OVERRIDE;
38  341 --                  { Computes how much of the view is to be allocated to each printed page }
38  342 --
38  343 --              PROCEDURE TTEView.DoSetPageOffset(coord: VPoint); OVERRIDE;
38  344 --                  { At printing time, determine the location of the 'interior' (body)
38  345 --                    of the page }
38  346 --
38  347 --              PROCEDURE TTEView.GetPrintExtent (VAR printExtent: VRect); OVERRIDE;
38  348 --                  { Print extent is the view's extent minus the insets. }
38  349 --
38  350 --          { Debugging }
38  351 --
38  352 --      {$IFC qDebug}
38  353 --              PROCEDURE TTEView.IdentifySoftware; OVERRIDE;
38  354 --                  { Tells compile date of this unit }
38  355 --
38  356 --              PROCEDURE TTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
38  357 --                                                             fieldAddr: Ptr;
38  358 --                  { Tell interesting stuff }                 fieldType: INTEGER)); OVERRIDE;
38  359 --      {$ENDC}
38  360 --
38  361 --          END;
38  362 --
38  363 --
38  364 --          { A command that adds characters to, or delete characters from, a TEView }
38  365 --          {-----------------------------------+
38  366 --          |   TTECommand                       |
38  367 --          +-----------------------------------}
38  368 --          TTECommand = OBJECT(TCommand)
38  369 --
38  370 --          { Fields }
38  371 --
38  372 --              fTEView:    TTEView;            { The TEView operated on }
38  373 --
38  374 --              fHTE:       TEHandle;           { same as fTEView's fHTE; duplicated for code
38  375 --                                                  efficiency }
38  376 --
38  377 --              fOldStart:  INTEGER;            { The beginning and ending positions of the }
38  378 --              fOldEnd:    INTEGER;            { selection at the moment just before the
38  379 --                                                  command was done }
38  380 --              fOldText:   Handle;             { If fOldStart = fOldEnd, i.e., if old selection
38  381 --                                                  had been an insertion point, this will be
38  382 --                                                  NIL.  Otherwise, provides a temporary home
38  383 --                                                  for the characters comprising the old
38  384 --                                                  selection }
38  385 --              fOldStyles: StScrpHandle;
38  386 --
38  387 --              fNewStart:  INTEGER;            { The beginning and ending locations in the Text… }
38  388 --              fNewEnd:    INTEGER;            { …of the new text that is added by the
38  389 --                                                  command, if any }
38  390 --              fNewText:   Handle;             { A Handle to the characters added by the command }
38  391 --              fNewStyles: StScrpHandle;
38  392 --
38  393 --              fPadding:   Handle;             { Handle to fill size between new and old.  This
38  394 --                                                  insures that we can always undo and redo. }
38  395 --              fTextPad:   INTEGER;            { Size difference between New and Old text }
```

```
38  396 --              fStylePad:  LONGINT;            { Size difference between New and Old styles }
38  397 --
38  398 --          { Initialize and Free }
38  399 --
38  400 --              PROCEDURE TTECommand.ITECommand(itsTEView: TTEView; itsCmdNumber: CmdNumber; itsSaveText: BOOLEAN);
38  401 --                  { Initialize the command; if unsuccessful, exit is via Failure mechanism. }
38  402 --
38  403 --              PROCEDURE TTECommand.Free; OVERRIDE;
38  404 --                  { Free the text handles holding information needed for Undo/Redo, then Frees SELF }
38  405 --
38  406 --          { Command execution phase overrides }
38  407 --
38  408 --              PROCEDURE TTECommand.DoIt; OVERRIDE;
38  409 --                  { Focuses, then calls DoMainFunction }
38  410 --
38  411 --              PROCEDURE TTECommand.RedoIt; OVERRIDE;
38  412 --                  { Focuses, calls RestoreSelection to get selection right, then calls DoMainFunction
38  413 --                      to reinstate the changes done in DoIt which were undone by a preceding UndoIt }
38  414 --
38  415 --              PROCEDURE TTECommand.UndoIt; OVERRIDE;
38  416 --                  { Focuses, then dispatches to RemoveAdditions, ReviveDeletions,
38  417 --                      and RestoreSelection to accomplish the Undo }
38  418 --
38  419 --          { Command execution - Restoration of Selection -- called in both UNDO and REDO phases }
38  420 --
38  421 --              PROCEDURE TTECommand.RestoreSelection;
38  422 --                  { Set the Selection to be what it was just before the DO phase of
38  423 --                      the command was performed }
38  424 --
38  425 --          { Command execution - steps in DO and REDO phases }
38  426 --
38  427 --              PROCEDURE TTECommand.BanishOldText;
38  428 --                  { Remove text that was selected at the outset of the Do phase }
38  429 --
38  430 --              PROCEDURE TTECommand.InstallNewText;
38  431 --                  { Install the new text }
38  432 --
38  433 --              PROCEDURE TTECommand.DoMainFunction;
38  434 --                  { Dispatches to the relevant methods for Do and Redo phases }
38  435 --
38  436 --          { Command execution - steps in UNDO phase }
38  437 --
38  438 --              PROCEDURE TTECommand.RemoveAdditions;
38  439 --                  { Remove any characters which were added by the DO phase of the
38  440 --                      command }
38  441 --
38  442 --              PROCEDURE TTECommand.ReviveDeletions;
38  443 --                  { Bring back the characters which were removed during the DO phase }
38  444 --
38  445 --      {$IFC qDebug}
38  446 --          { Debugging }
38  447 --
38  448 --              PROCEDURE TTECommand.Fields (PROCEDURE DoToField (fieldName: Str255;
38  449 --                                                  fieldAddr: Ptr;
38  450 --                                                  fieldType: INTEGER)); OVERRIDE;
38  451 --      {$ENDC}
38  452 --
38  453 --              END;
38  454 --
38  455 --
38  456 --          {-----------------------------------+
38  457 --          |   TTECutCopyCommand                |
38  458 --          +-----------------------------------}
38  459 --          TTECutCopyCommand = OBJECT(TTECommand)
38  460 --
38  461 --              fClipCreated:   BOOLEAN;        { Clipboard view created OK }
38  462 --
38  463 --          { Creation and Destruction }
38  464 --
38  465 --              PROCEDURE TTECutCopyCommand.ITECutCopyCommand(itsTEView: TTEView;
38  466 --                      itsCmdNumber: CmdNumber);
38  467 --                  { Initializes the command }
38  468 --
38  469 --              PROCEDURE TTECutCopyCommand.Free; OVERRIDE;
38  470 --                  { Free the command }
38  471 --
38  472 --            { Command execution }
38  473 --
38  474 --              PROCEDURE TTECutCopyCommand.DoIt; OVERRIDE;
38  475 --                  { Launches a TEView for installation in the Clipboard, then calls
38  476 --                      DoMainFunction }
38  477 --
38  478 --              PROCEDURE TTECutCopyCommand.ReviveDeletions; OVERRIDE;
38  479 --                  { Bring back the characters which were removed during the DO phase;
38  480 --                      called during UNDO phase }
38  481 --
38  482 --          END;
38  483 --
38  484 --
38  485 --          {-----------------------------------+
38  486 --          |   TTEPasteCommand                  |
38  487 --          +-----------------------------------}
38  488 --          TTEPasteCommand = OBJECT(TTECommand)
38  489 --
38  490 --            { Initialization }
38  491 --
38  492 --              PROCEDURE TTEPasteCommand.ITEPasteCommand(itsTEView: TTEView);
38  493 --                  { Initialize the command; if unsuccessful, signalled by Failure mechanism }
38  494 --
```

```
38  495 --              END;
38  496 --
38  497 --
38  498 --          {-----------------------------------+
38  499 --          |   TTEStyleCommand                 |
38  500 --          +-----------------------------------}
38  501 --          TTEStyleCommand = OBJECT(TTECommand)
38  502 --
38  503 --            { Fields }
38  504 --            { These two fields are only used in non-styled TextEdit records }
38  505 --              fMode:              INTEGER;          { Mode for style change                    }
38  506 --              fOldTextStyle:      TextStyle;        { The original text style                  }
38  507 --              fNewTextStyle:      TextStyle;        { What we're replacing it with             }
38  508 --
38  509 --            { Initialization }
38  510 --
38  511 --              PROCEDURE TTEStyleCommand.ITEStyleCommand(itsTEView: TTEView; itsNewStyle: TextStyle;
38  512 --                              itsCmdNumber: CmdNumber; itsMode: INTEGER);
38  513 --                  { Initialize the command; if unsuccessful, signalled by Failure mechanism }
38  514 --
38  515 --            { Command execution }
38  516 --
38  517 --              PROCEDURE TTEStyleCommand.InstallOneStyle(newStyl: TextStyle);
38  518 --                  { Installs one style over entire record - for use with old TextEdit }
38  519 --
38  520 --              PROCEDURE TTEStyleCommand.InstallManyStyles(newStyls: StScrpHandle);
38  521 --                  { Installs styles over selection range - for use with styled TextEdit }
38  522 --
38  523 --              PROCEDURE TTEStyleCommand.DoIt; OVERRIDE;
38  524 --                  { Applies style to selection range }
38  525 --
38  526 --              PROCEDURE TTEStyleCommand.UndoIt; OVERRIDE;
38  527 --
38  528 --              PROCEDURE TTEStyleCommand.RedoIt; OVERRIDE;
38  529 --
38  530 --      {$IFC qDebug}
38  531 --          { Debugging }
38  532 --              PROCEDURE TTEStyleCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
38  533 --                                              fieldAddr: Ptr;
38  534 --                                              fieldType: INTEGER)); OVERRIDE;
38  535 --      {$ENDC}
38  536 --
38  537 --              END;
38  538 --
38  539 --          {-----------------------------------+
38  540 --          |   TTETypingCommand                |
38  541 --          +-----------------------------------}
38  542 --          TTETypingCommand = OBJECT(TTECommand)
38  543 --
38  544 --              fCompleted: BOOLEAN;    { Indicates whether further keystrokes will be
38  545 --                                          extensions to this command (FALSE) or
38  546 --                                          whether the command has already been
38  547 --                                          completed (TRUE) }
38  548 --              fFirstChar: CHAR;       { First character typed }
38  549 --
38  550 --          { Creation and destruction }
38  551 --
38  552 --              PROCEDURE TTETypingCommand.ITETypingCommand(itsTEView: TTEView;
38  553 --                      itsFirstChar: Char);
38  554 --                  { Initialize the command; if not successful, exit is via Failure
38  555 --                    mechanism }
38  556 --
38  557 --              PROCEDURE TTETypingCommand.Free; OVERRIDE;
38  558 --                  { Deallocate the command and dependent structures }
38  559 --
38  560 --          { Command processing }
38  561 --
38  562 --              PROCEDURE TTETypingCommand.DoNormalChar(aChar: CHAR);
38  563 --                  { Handle any typed character except a backspace }
38  564 --
38  565 --              PROCEDURE TTETypingCommand.BkSpcLeft(theText: Handle; curStart: INTEGER);
38  566 --                  { Handle backspace to the left of the original selection }
38  567 --
38  568 --              PROCEDURE TTETypingCommand.BkSpcRight(theText: Handle; curStart: INTEGER);
38  569 --                  { Handle backspace to the right of the original selection }
38  570 --
38  571 --              PROCEDURE TTETypingCommand.AddCharacter(aChar: Char);
38  572 --                  { Add another character to an already-launched TETypingCommand }
38  573 --
38  574 --              PROCEDURE TTETypingCommand.DoIt; OVERRIDE;
38  575 --                  { First processing for command; adds first character }
38  576 --
38  577 --              PROCEDURE TTETypingCommand.UndoIt; OVERRIDE;
38  578 --                  { Process undo typing }
38  579 --
38  580 --              PROCEDURE TTETypingCommand.CompleteTyping;
38  581 --                  { Mark as no more typing allowed and fix up style scrap, if any }
38  582 --
38  583 --      {$IFC qDebug}
38  584 --          { Debugging }
38  585 --
38  586 --              PROCEDURE TTETypingCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
38  587 --                                                  fieldAddr: Ptr;
38  588 --                                                  fieldType: INTEGER)); OVERRIDE;
38  589 --      {$ENDC}
38  590 --
38  591 --              END;
38  592 --
38  593 --      {-----------------------------------+
```

```
38  594 --      |   Global variables                |
38  595 --      +-----------------------------------}
38  596 --      VAR
38  597 --          defClikLoopProc:    ProcPtr;
38  598 --
38  599 --      {-----------------------------------+
38  600 --      |   Global routines                 |
38  601 --      +-----------------------------------}
38  602 --          PROCEDURE InitUTEView;
38  603 --              { Initialize TEView unit. }
38  604 --
38  605 --          PROCEDURE SetSelect(theStart, theEnd: INTEGER; hTE: TEHandle);
38  606 --              { For those times when we don't want the selection range hilited when changed }
38  607 --
38  608 --          FUNCTION  AutoscrollTEView: BOOLEAN;
38  609 --              { Called from TextEdit when the mouse is down, so that autoscrolling
38  610 --                  may be done. }
38  611 --
38  612 --          {$IFC qDebug}
38  613 --          PROCEDURE DumpTERecord(aTEH: TEHandle);
38  614 --              { writes salient information about the TERecord out to the Debug window }
38  615 --          PROCEDURE DumpTTECommand(theTTECommand: TTECommand);
38  616 --              { writes TTECommand info out to Debug window }
38  617 --          {$ENDC}
38  618 --
38  619 --      (*****************************************************************************************)
38  620 --      IMPLEMENTATION
38  621 --      (*****************************************************************************************)
38  622 --
38  623 --      {$I UTEView.Globals.p}
```

```
39    1 --      {$P}
39    2 --      {UTEView.Globals.p}
39    3 --      { Copyright © 1985 - 1988 Apple Computer, Inc.  All rights reserved. }
39    4 --
39    5 --      {-----------------------------------+
39    6 --      |   local variables                 |
39    7 --      +-----------------------------------}
39    8 --      VAR
39    9 --          gCurrTEView:    TTEView;                    { For the benefit of AutoScroll only        }
39   10 --      {$IFC qDebug}
39   11 --          gTEIntenseDebugging:    BOOLEAN;
39   12 --      {$ENDC}
39   13 --
39   14 --      {*****************************************************************************************}
39   15 --      {*      U t i l i t y   r o u t i n e s                                                  *}
39   16 --      {*****************************************************************************************}
39   17 --      {$S TEInit}
39   18 --      {-----------------------------------+
39   19 --      |   InitUTEView                     |
39   20 --      +-----------------------------------}
39   21 -- A    PROCEDURE InitUTEView;
39   22 --      VAR
39   23 --          aTEView:        TTEView;
39   24 --
39   25 0- A    BEGIN
39   26 --      {$IFC qTemplateViews}
39   27 --          NEW(aTEView);       FailNIL(aTEView);       RegisterStdType('TTEView', 'tevw', aTEView);
39   28 --      {$ENDC}
39   29 -0 A    END;
39   30 --
39   31 --      {$IFC qDebug}
39   32 --          {$S TEDebug}
39   33 --          {-----------------------------------+   For debugging - prints out actual character at
39   34 --          |   WriteChar                       |       position "index" in text < 2047
39   35 --          +-----------------------------------}
39   36 -- A        PROCEDURE WriteChar(index: INTEGER; hText: Handle);
39   37 --          CONST
39   38 --              kMaxIndex = 2047;
39   39 --          TYPE
39   40 --              x   =   PACKED ARRAY[0..kMaxIndex] OF Char;
39   41 --              PX  =   ^x;
39   42 --              HX  =   ^Px;
39   43 0- A        BEGIN
39   44 --              IF index <= kMaxIndex THEN
39   45 --                  Write('[', index:1, ':', HX(hText)^^[index]:1, '] ');
39   46 -0 A        END;
39   47 --
39   48 --          {$S TEDebug}
39   49 --          {-----------------------------------+   Reset kMaxCharsToPrint upwards as needed.  Set
39   50 --          |   DumpTERecord                    |       to 100 to avoid runaway debug window output.
39   51 --          +-----------------------------------}
39   52 -- A        PROCEDURE DumpTERecord(aTEH: TEHandle);
39   53 --          CONST
39   54 --              kMaxCharsToPrint = 100;
39   55 --          VAR
39   56 --              size:   INTEGER;
39   57 --              hText:  Handle;
39   58 --              i:      INTEGER;
39   59 --
39   60 0- A        BEGIN
39   61 --              Write('TE -- dest rect: '); WriteRect(aTEH^^.destRect);
39   62 --              WriteLn;
39   63 --
39   64 --              size :=  aTEH^^.teLength;
39   65 --              Write('Line ht: ', aTEH^^.lineHeight:1);
39   66 --              Write('; teLength: ', size:1);
39   67 --              Write('; hText length: ', GetHandleSize(aTEH^^.hText):1);
39   68 --              Write('; lines: ', aTEH^^.nLines:1);
39   69 --              WriteLn;
39   70 --
39   71 --              IF gIntenseDebugging THEN
39   72 1-                  BEGIN
39   73 --                  hText := aTEH^^.hText;
39   74 --                  Write('Chars: ');
39   75 --                  FOR i := 0 TO Min(kMaxCharsToPrint, Min(2047, size - 1)) DO
39   76 2-                      BEGIN
39   77 --                      WriteChar(i, hText);
39   78 --                      IF i MOD 10 = 9 THEN
39   79 --                          WriteLn;
39   80 -2                      END;
39   81 --                  WriteLn;
39   82 --
39   83 --                  Write('InPort vis bbox: '); WriteRect(aTEH^^.inPort^.visRgn^^.rgnBBox);
39   84 --                  Write('; clip bbox: '); WriteRect(aTEH^^.inPort^.clipRgn^^.rgnBBox); WriteLn;
39   85 --
39   86 --                  IF aTEH^^.inPort <> thePort THEN
39   87 2-                      BEGIN
39   88 --                      Write('thePort vis bbox: '); WriteRect(thePort^.visRgn^^.rgnBBox);
39   89 --                      Write('; clip bbox: '); WriteRect(thePort^.clipRgn^^.rgnBBox); WriteLn;
39   90 -2                      END;
39   91 -1                  END;
39   92 --
39   93 -0 A        END;
39   94 --
39   95 --
39   96 --          {$S TEFields}
39   97 --          {-----------------------------------+
39   98 --          |   DumpTTECommand                  |   Debug routine to print out involved TTECommands
```

```
39   99 --          +-----------------------------------}
39  100 -- A       PROCEDURE DumpTTECommand(theTTECommand: TTECommand);
39  101 --          TYPE
39  102 --              TSPtr = ^TextStyle;
39  103 --          VAR
39  104 --              i:              LONGINT;
39  105 --              oldStyleSize:   LONGINT;
39  106 --              newStyleSize:   LONGINT;
39  107 --              last:           LONGINT;
39  108 0- A       BEGIN
39  109 --              WITH theTTECommand DO
39  110 --                  WRITELN('old start/end, new start/end, diff :',
39  111 --                    fOldStart:0, '/', fOldEnd:0, ',  ',
39  112 --                    fNewStart:0, '/', fNewEnd:0, ',  ',
39  113 --                    MAX(MAX(fOldEnd-fOldStart, 0), MAX(fNewEnd-fNewStart, 0)):0);
39  114 --              WRITELN('fTextPad :', theTTECommand.fTextPad:0);
39  115 --              WRITELN('fPadding size :', GetHandleSize(theTTECommand.fPadding):0);
39  116 --
39  117 --              oldStyleSize := 0;
39  118 --              newStyleSize := 0;
39  119 --              WRITELN('** Old styles:');
39  120 --              IF theTTECommand.fOldStyles = NIL THEN
39  121 --                  WRITELN('NONE!')
39  122 --              ELSE
39  123 1-                  BEGIN
39  124 --                  last := theTTECommand.fOldStyles^^.scrpNStyles;
39  125 --                  WRITELN('Number of table entries: ', last:0);
39  126 --                  FOR i := 0 TO last-1 DO
39  127 --                      WITH theTTECommand.fOldStyles^^.scrpStyleTab[i] DO
39  128 2-                          BEGIN
39  129 --                          WRITE(i:0, ', ofs:', scrpStartChar:0, ' ');
39  130 --                          WrLblField('', @scrpFont, bStyle);
39  131 --                          WRITELN;
39  132 -2                          END;
39  133 --                  oldStyleSize := GetHandleSize(Handle(theTTECommand.fOldStyles));
39  134 -1                  END;
39  135 --
39  136 --              WRITELN('** New styles:');
39  137 --              IF theTTECommand.fNewStyles = NIL THEN
39  138 --                  WRITELN('NONE!')
39  139 --              ELSE
39  140 1-                  BEGIN
39  141 --                  last := theTTECommand.fNewStyles^^.scrpNStyles;
39  142 --                  WRITELN('Number of table entries: ', last:0);
39  143 --                  FOR i := 0 TO last-1 DO
39  144 --                      WITH theTTECommand.fNewStyles^^.scrpStyleTab[i] DO
39  145 2-                          BEGIN
39  146 --                          WRITE(i:0, ', ofs:', scrpStartChar:0, ' ');
39  147 --                          WrLblField ('', @scrpFont, bStyle);
39  148 --                          WRITELN;
39  149 -2                          END;
39  150 --                  newStyleSize := GetHandleSize(Handle(theTTECommand.fNewStyles));
39  151 -1                  END;
39  152 --
39  153 --              WRITELN('Styles size diff: ', MAX(newStyleSize, oldStyleSize):0);
39  154 --              WRITELN('fStylePad :', theTTECommand.fStylePad:0);
39  155 -0 A      END;
39  156 --
39  157 --      {$ENDC qDebug}
39  158 --
39  159 --      {$S TERes}
39  160 --      {-----------------------------------+   TESetSelect does highlighting too, which is ugly
39  161 --      |   SetSelect                       |       in most places where we need to alter record
39  162 --      +-----------------------------------}
39  163 -- A   PROCEDURE SetSelect(theStart, theEnd: INTEGER; hTE: TEHandle);
39  164 0- A   BEGIN
39  165 --          WITH hTE^^ DO
39  166 1-              BEGIN
39  167 --              selStart := theStart;
39  168 --              selEnd := theEnd;
39  169 -1              END;
39  170 -0 A   END;
39  171 --
39  172 --      {$S TENonRes}
39  173 --      {-----------------------------------+
39  174 --      |   AutoscrollTEView                |   Callback routine from TextEdit, for auto-scrolling
39  175 --      +-----------------------------------}
39  176 -- A   FUNCTION  AutoscrollTEView:  BOOLEAN;
39  177 --      VAR
39  178 --          msePt:      Point;
39  179 --          viewPt:     VPoint;
39  180 --          visRect:    Rect;
39  181 --          delta:      VPoint;
39  182 --          vhs:        VHSelect;
39  183 --          lead:       INTEGER;
39  184 --          trail:      INTEGER;
39  185 --          itsDown:    BOOLEAN;
39  186 --          scroller:   TScroller;
39  187 --
39  188 0- A   BEGIN
39  189 --          GetMouse(msePt);
39  190 --          itsDown := StillDown;
39  191 --          IF itsDown AND (gCurrTEView <> NIL) THEN
39  192 1-              BEGIN
39  193 --              scroller := gCurrTEView.fScroller;
39  194 --              IF scroller <> NIL THEN
39  195 --                  IF (scroller.fScrollBars[h] <> NIL) OR (scroller.fScrollBars[v] <> NIL) THEN
39  196 2-                      BEGIN
39  197 --                      gCurrTEView.QDToViewPt(msePt, viewPt);
```

```
39  198 --                      scroller.AutoScroll(viewPt, delta);
39  199 --                      gCurrTEView.GetVisibleRect(visRect);
39  200 --
39  201 --                      FOR vhs := v TO h DO
39  202 3-                          BEGIN
39  203 --                          lead := gCurrTEView.fLocation.vh[vhs] - visRect.topLeft.vh[vhs];
39  204 --                          trail := gCurrTEView.fLocation.vh[vhs] + gCurrTEView.fSize.vh[vhs] -
39  205 --                                   visRect.botRight.vh[vhs];
39  206 --
39  207 --                          IF delta.vh[vhs] < 0 THEN
39  208 --                              delta.vh[vhs] := Min(Max(delta.vh[vhs], lead), 0)
39  209 --                          ELSE
39  210 --                              delta.vh[vhs] := Max(Min(delta.vh[vhs], trail), 0);
39  211 -3                          END;
39  212 --                      { The intent of the above is not to do autoscrolling that would scroll
39  213 --                        beyond the subview boundary in any direction }
39  214 --
39  215 --                      IF (delta.v <> 0) OR (delta.h <> 0) THEN
39  216 3-                          BEGIN
39  217 --                          scroller.ScrollBy(delta.h, delta.v, TRUE);
39  218 --
39  219 --                          { includes a frame Update, which could change lots of things, thus
39  220 --                            requiring us, tiresomely, to take some or all of the following
39  221 --                            restorative precautions: }
39  222 --
39  223 --                          IF gCurrTEView.Focus THEN
39  224 --                              gCurrTEView.ClipFurtherTo(gCurrTEView.fHTE^^.destRect, 0, 0);
39  225 -3                          END;
39  226 -2                      END;
39  227 -1              END;
39  228 --          AutoscrollTEView := TRUE;
39  229 -0 A    END;
38  624 --      {$I UTEView.TTEView.p}
```

```
40    1 --      {$P}
40    2 --      { UTEView.TTEView.p }
40    3 --      { Copyright © 1984-1988 Apple Computer Inc.  All rights reserved. }
40    4 --
40    5 --      (*****************************************************************************************)
40    6 --      (*      T T E V i e w                                                                    *)
40    7 --      (*****************************************************************************************)
40    8 --
40    9 --      {$S TEOpen}
40   10 --      {-----------------------------------+
40   11 --      |   ITEView                         |
40   12 --      +-----------------------------------}
40   13 -- A    PROCEDURE TTEView.ITEView (itsDocument: TDocument; itsSuperView: TView;
40   14 --                                 itsLocation, itsSize: VPoint;
40   15 --                                 itsHDeterminer, itsVDeterminer: SizeDeterminer;
40   16 --                                 itsInset: Rect;
40   17 --                                 itsTextStyle: TextStyle;
40   18 --                                 itsJustification: INTEGER;
40   19 --                                 itsStyleType, itsAutoWrap: BOOLEAN);
40   20 --
40   21 0- A    BEGIN { TTEView.ITEView }
40   22 --
40   23 --      {$IFC qDebug}
40   24 --          gTEIntenseDebugging := FALSE;
40   25 --      {$ENDC}
40   26 --
40   27 --          fHTE := NIL;
40   28 --          fScroller := NIL;
40   29 --          fText := NIL;
40   30 --          fSavedTEHandle := NIL;
40   31 --          fInset := itsInset;
40   32 --          fKeyCmdNumber := cTyping;
40   33 --          fMaxChars := kUnlimited;
40   34 --          fLastHeight := 0;
40   35 --          fTypingCommand := NIL;
40   36 --          fTextStyle := itsTextStyle;
40   37 --          fJustification := itsJustification;
40   38 --          fAcceptsChanges := TRUE;                    { Stuff to FALSE if you don't want to allow }
40   39 --                                                      { …Cut, Paste, or Typing                    }
40   40 --          fStyleType := itsStyleType AND              { Only if asked, and we _can_               }
40   41 --            gConfiguration.hasStyleTextedit;
40   42 --          fAutoWrap := itsAutoWrap;
40   43 --          fFreeText := FALSE;
40   44 --          fSpecsChanged := FALSE;
40   45 --
40   46 --          fLastPageBreak := 0;
40   47 --          fLastLine := 0;
40   48 --
40   49 --          IView(itsDocument, itsSuperView, itsLocation, itsSize, itsHDeterminer, itsVDeterminer);
40   50 --
40   51 --          MakeTERecord;
40   52 --
40   53 --          defClikLoopProc := fHTE^^.clikLoop;         { Just in case we want to restore it…       }
40   54 --          SetClikLoop(@AutoscrollTEView, fHTE);       { fHTE^^.clikLoop := @TEViewClikLoop;       }
40   55 --          { ??? Things don't work well if fText is non-NIL.  Is this the way to solve it? }
40   56 --          fText := fHTE^^.hText;
40   57 --
40   58 --          fIdleFreq := MAX(GetCaretTime DIV 2, 1);    { Makes sure I get DoIdle messages at idle  }
40   59 --                                                      { …time, so I can blink the caret           }
40   60 --          IF fScroller <> NIL THEN                    { Set scroll parameters if in a scroller    }
40   61 --              BeInScroller(fScroller);
40   62 -0 A    END;
40   63 --
40   64 --
40   65 --      {$IFC qTemplateViews}
40   66 --      {$S TEOpen}
40   67 --      {-----------------------------------+
40   68 --      |   IRes                            |
40   69 --      +-----------------------------------}
40   70 -- A    PROCEDURE TTEView.IRes (itsDocument: TDocument; itsSuperView: TView; VAR itsParams: Ptr); OVERRIDE;
40   71 --
40   72 --      VAR
40   73 --          aTextStyle:     TextStyle;
40   74 --
40   75 0- A    BEGIN
40   76 --      {$IFC qDebug}
40   77 --          gTEIntenseDebugging := FALSE;
40   78 --      {$ENDC}
40   79 --
40   80 --          fHTE := NIL;                                { In case of emergency.                     }
40   81 --          fScroller := NIL;
40   82 --          fText := NIL;
40   83 --          fSavedTEHandle := NIL;
40   84 --          fLastPageBreak := 0;
40   85 --          fLastLine := 0;
40   86 --
40   87 --          INHERITED IRes(itsDocument, itsSuperView, itsParams);
40   88 --
40   89 --          WITH TEViewTemplatePtr(itsParams)^ DO
40   90 1-             BEGIN
40   91 --              fTypingCommand := NIL;
40   92 --              fLastHeight := 0;
40   93 --              fSpecsChanged := FALSE;
40   94 --
40   95 --              fInset := itsInset;
40   96 --              fKeyCmdNumber := itsKeyCmdNumber;
40   97 --              fMaxChars := itsMaxChars;
40   98 --              SetTextStyle(aTextStyle,
```

```
40   99 --              GetFontNum(itsFontName), itsTextFace, itsTextSize, itsTextColor);
40  100 --            fTextStyle := aTextStyle;
40  101 --            fJustification := itsJustification;
40  102 --            fAcceptsChanges := itsAcceptsChanges;
40  103 --            fStyleType := itsStyleType AND gConfiguration.hasStyleTextedit;
40  104 --            fAutoWrap := itsAutoWrap;
40  105 --            fFreeText := itsFreesText;
40  106 --
40  107 --            MakeTERecord;
40  108 --
40  109 --            defClikLoopProc := fHTE^^.clikLoop;     { Just in case we want to restore it_       }
40  110 --            SetClikLoop(@AutoscrollTEView, fHTE);   { fHTE^^.clikLoop := @TEViewClikLoop;      }
40  111 --            fText := fHTE^^.hText;
40  112 --
40  113 --            fIdleFreq := MAX(GetCaretTime DIV 2,1); { Makes sure I get DoIdle messages at idle  }
40  114 --                                                    { _time, so I can blink the caret           }
40  115 -1            END;
40  116 --
40  117 --        OffsetPtrWStr(itsParams, SIZEOF(TEViewTemplate));
40  118 -0 A   END;
40  119 --     {$ENDC}
40  120 --
40  121 --
40  122 --     {$IFC qWriteTemplates}
40  123 --     {$S MAWriteRes}
40  124 --     {-----------------------------------+
40  125 --     |   WRes                            |
40  126 --     +-----------------------------------}
40  127 -- A   PROCEDURE TTEView.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
40  128 --     VAR
40  129 --         theFont:    Str255;
40  130 --         ttPtr:      TEViewTemplatePtr;
40  131 0- A   BEGIN
40  132 --         INHERITED WRes(theResource, itsParams);
40  133 --
40  134 --         GetFontName(fTextStyle.tsFont, theFont);
40  135 --
40  136 --         ttPtr := TEViewTemplatePtr(ExpandPtrWStr(theResource, itsparams,
40  137 --           SIZEOF(TEViewTemplate), LENGTH(theFont)));
40  138 --
40  139 --         WITH ttPtr^ DO
40  140 1-             BEGIN
40  141 --             itsStyleType := fStyleType;
40  142 --             itsAutoWrap := fAutoWrap;
40  143 --             itsAcceptsChanges := fAcceptsChanges;
40  144 --             itsFreesText := fFreeText;
40  145 --             itsKeyCmdNumber := fKeyCmdNumber;
40  146 --             itsMaxChars := fMaxChars;
40  147 --             itsInset := fInset;
40  148 --             itsJustification := fJustification;
40  149 --             WITH fTextStyle DO
40  150 2-                 BEGIN
40  151 --                 itsTextFace := tsFace;
40  152 --                 itsTextSize := tsSize;
40  153 --                 itsTextColor := tsColor;
40  154 -2                 END;
40  155 --             (* itsFontName := theFont; *)
40  156 --             CopyStr255(theFont, PRStr(itsFontName));
40  157 -1             END;
40  158 -0 A   END;
40  159 --
40  160 --     {$S MAWriteRes}
40  161 --     {-----------------------------------+
40  162 --     |   WriteRes                        |
40  163 --     +-----------------------------------}
40  164 -- A   PROCEDURE TTEView.WriteRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
40  165 0- A   BEGIN
40  166 --         gWResSignature := 'tevw';   gWResType := 'TTEView';
40  167 --         WRes(theResource, itsParams);
40  168 -0 A   END;
40  169 --     {$ENDC}
40  170 --
40  171 --
40  172 --     {$S TEClose}
40  173 --     {-----------------------------------+
40  174 --     |   Free                            |
40  175 --     +-----------------------------------}
40  176 -- A   PROCEDURE TTEView.Free; OVERRIDE;
40  177 0- A   BEGIN
40  178 --         IF fHTE <> NIL THEN
40  179 1-             BEGIN
40  180 --             IF fSavedTEHandle <> NIL THEN
40  181 --                 WITH fHTE^^ DO
40  182 2-                     BEGIN                       { Worry about fText separately.  Put back   }
40  183 --                     hText := fSavedTEHandle;    { _the handle which TE allocated            }
40  184 --                 {$H-}
40  185 --                     teLength :=                 { This is here because it only makes        }
40  186 --                       GetHandleSize(hText);
40  187 --                     IF fFreeText THEN           { _sense if fSavedTEHandle is not NIL.      }
40  188 --                         DisposIfHandle(fText);
40  189 --                 {$H+}
40  190 -2                     END;
40  191 --             TEDispose(fHTE);
40  192 --             fSavedTEHandle := NIL;
40  193 -1             END;
40  194 --         INHERITED Free;
40  195 -0 A   END;
40  196 --
40  197 --     {$S TENonRes}
```

```
40  198 --      {-----------------------------------+
40  199 --      |   AutoScrolling                   |
40  200 --      +-----------------------------------}
40  201 -- A    PROCEDURE TTEView.AutoScrolling (doScrolling: BOOLEAN);
40  202 0- A    BEGIN
40  203 --      {$IFC NOT qNeedsROM128K}
40  204 --          IF gConfiguration.hasROM128K THEN
40  205 --      {$ENDC}
40  206 --              IF fHTE <> NIL THEN
40  207 --                  TEAutoView(doScrolling, fHTE);
40  208 -0 A    END;
40  209 --
40  210 --      {$S TENonRes}
40  211 --      {-----------------------------------+
40  212 --      |   BeInPort                        |
40  213 --      +-----------------------------------}
40  214 -- A    PROCEDURE TTEView.BeInPort(itsPort: GrafPtr); OVERRIDE;
40  215 0- A    BEGIN
40  216 --
40  217 --          IF fHTE <> NIL THEN
40  218 1-              BEGIN
40  219 --              WITH fHTE^^ DO
40  220 --                  IF itsPort = NIL THEN
40  221 --                      inPort := gWorkPort
40  222 --                  ELSE
40  223 --                      inPort := itsPort;
40  224 --
40  225 --              IF itsPort = NIL THEN
40  226 2-                  BEGIN
40  227 --                  DoneTyping;
40  228 --                  fSpecsChanged := TRUE;
40  229 -2                  END;
40  230 -1              END;
40  231 --
40  232 --          INHERITED BeInPort(itsPort);
40  233 -0 A    END;
40  234 --
40  235 --
40  236 --      {$S TENonRes}
40  237 --      {-----------------------------------+
40  238 --      |   BeInScroller                    |
40  239 --      +-----------------------------------}
40  240 -- A    PROCEDURE TTEView.BeInScroller(itsScroller: TScroller); OVERRIDE;
40  241 --      VAR
40  242 --          vertScrollUnit:     INTEGER;
40  243 0- A    BEGIN
40  244 --          fScroller := itsScroller;
40  245 --          IF (fHTE <> NIL) & (itsScroller <> NIL) THEN
40  246 1-              BEGIN
40  247 --              IF fHTE^^.lineHeight > 0 THEN           { This works for both old & new TextEdit    }
40  248 --                  vertScrollUnit := fHTE^^.lineHeight
40  249 --              ELSE IF gConfiguration.hasScriptManager THEN { Ask for system default size          }
40  250 --                  vertScrollUnit := GetDefFontSize
40  251 --              ELSE
40  252 --                  vertScrollUnit := kStdScrollUnit;
40  253 --
40  254 --              itsScroller.SetScrollParameters(kStdScrollUnit, vertScrollUnit, FALSE, TRUE);
40  255 -1              END;
40  256 --
40  257 --          INHERITED BeInScroller(itsScroller);
40  258 -0 A    END;
40  259 --
40  260 --
40  261 --      {$S TENonRes}
40  262 --      {-----------------------------------+
40  263 --      |   CalcMinSize                     |
40  264 --      +-----------------------------------}
40  265 -- A    PROCEDURE TTEView.CalcMinSize(VAR minSize: VPoint); OVERRIDE;
40  266 --      VAR
40  267 --          selLoc:         Point;
40  268 --          theHeight:      INTEGER;
40  269 --          theAscent:      INTEGER;
40  270 --          vHite:          LONGINT;
40  271 --          vhs:            VHSelect;
40  272 --          theLines:       INTEGER;
40  273 --          theLength:      INTEGER;
40  274 --          theStyle:       TextStyle;
40  275 --
40  276 0- A    BEGIN
40  277 --          vHite := CalcRealHeight;
40  278 --
40  279 --          { Note that we omit the margins here, so that if TView.ComputeExtent rounds up to a
40  280 --            page multiple, the margins will get tacked on after.  Also, insure we don't run off
40  281 --            the end of the coordinate system if there are many lines of text. }
40  282 --
40  283 --          SetVPt(minSize, fSize.h - fInset.left - fInset.right, MIN(kMaxCoord, vHite));
40  284 --
40  285 --          FOR vhs := v TO h DO
40  286 --              IF fSizeDeterminer[vhs] = sizeFixed THEN
40  287 --                  minSize.vh[vhs] := fSize.vh[vhs];
40  288 -0 A    END;
40  289 --
40  290 --
40  291 --      {$S TENonRes}
40  292 --      {-----------------------------------+
40  293 --      |   CalcRealHeight                  |
40  294 --      +-----------------------------------}
40  295 -- A    FUNCTION  TTEView.CalcRealHeight: INTEGER;
40  296 --      VAR
```

```
40  297 --          lastIsCR:       BOOLEAN;
40  298 --          theMode:        INTEGER;
40  299 --          theHeight:      INTEGER;
40  300 --          theStyle:       TextStyle;
40  301 --          theFontInfo:    FontInfo;
40  302 --          savedPort:      GrafPtr;
40  303 0- A    BEGIN
40  304 --      {$IFC qRangeCheck}{$R-}{$ENDC}
40  305 --          WITH fHTE^^ DO
40  306 --              lastIsCR := (teLength <= 0) |
40  307 --                          (CharsHandle(hText)^^[PRED(teLength)] = chReturn);
40  308 --      {$IFC qRangeCheck}{$R+}{$ENDC}
40  309 --
40  310 --          IF fStyleType = kWithStyle THEN
40  311 1-              BEGIN
40  312 --              theHeight := 0;
40  313 --              IF fHTE^^.nLines > 0 THEN
40  314 --                  theHeight := TEGetHeight(0, MAXINT, fHTE);
40  315 --
40  316 --              IF lastIsCR THEN                        { then can't use TEGetHeight so we  }
40  317 2-                  BEGIN                               { have to figure it out ourselves. }
40  318 --                  theMode := doAll;
40  319 --                  lastIsCR := ContinuousStyle(MAXINT, MAXINT, theMode, theStyle);
40  320 --                  GetPort(savedPort);
40  321 --                  SetPort(gWorkPort);
40  322 --                  SetPortTextStyle(theStyle);
40  323 --                  GetFontInfo(theFontInfo);           { What a pain…                      }
40  324 --                  SetPort(savedPort);
40  325 --                  WITH theFontInfo DO
40  326 --                      theHeight := theHeight + ascent + descent + leading;
40  327 -2                  END;
40  328 -1              END
40  329 --          ELSE
40  330 1-              BEGIN
40  331 --              theHeight := fHTE^^.nLines + ORD(lastIsCR);
40  332 --              theHeight := theHeight * fHTE^^.lineHeight;
40  333 -1              END;
40  334 --
40  335 --          CalcRealHeight := theHeight;
40  336 --      {$IFC qDebug}
40  337 --          IF gIntenseDebugging THEN
40  338 --              WRITELN('CalcRealHeight=', theHeight:0);
40  339 --      {$ENDC}
40  340 -0 A    END;
40  341 --
40  342 --
40  343 --      {$S TENonRes}
40  344 --      {-----------------------------------+
40  345 --      |   ChangeWrap                      |
40  346 --      +-----------------------------------}
40  347 -- A    PROCEDURE TTEView.ChangeWrap (newAutoWrap, redraw: BOOLEAN);
40  348 0- A    BEGIN
40  349 --          fAutoWrap := newAutoWrap;
40  350 --          WITH fHTE^^ DO
40  351 --              IF newAutoWrap THEN
40  352 --                  crOnly := 0
40  353 --              ELSE
40  354 --                  crOnly := -1;
40  355 --          IF redraw THEN
40  356 1-              BEGIN
40  357 --              RecalcText;
40  358 --              SynchView(kRedraw);
40  359 -1              END;
40  360 -0 A    END;
40  361 --
40  362 --      {$S TENonRes}
40  363 --      {-----------------------------------+
40  364 --      |   ComputeSize                     |
40  365 --      +-----------------------------------}
40  366 -- A    PROCEDURE TTEView.ComputeSize(VAR newSize: VPoint); OVERRIDE;
40  367 --
40  368 --          {-------------------------------+
40  369 --          |   NeedAdjust                  |
40  370 --          +-------------------------------]
40  371 -- B        FUNCTION NeedAdjust(vhs:   VHSelect): BOOLEAN;
40  372 0- B        BEGIN
40  373 --              NeedAdjust := NOT (fSizeDeterminer[vhs] IN
40  374 --                                 [sizeFixed, sizeSuperView, sizeRelSuperView]);
40  375 -0 B        END;
40  376 --
40  377 0- A    BEGIN { TTEView.ComputeSize }
40  378 --          INHERITED ComputeSize(newSize);
40  379 --
40  380 --          IF NeedAdjust(h) THEN                   { If necessary, tack on the margins         }
40  381 --              newSize.h := MIN(kMaxCoord, ORD4(newSize.h) + fInset.left + fInset.right);
40  382 --          IF NeedAdjust(v) THEN
40  383 --              newSize.v := MIN(kMaxCoord, ORD4(newSize.v) + fInset.top + fInset.bottom);
40  384 -0 A    END;
40  385 --
40  386 --
40  387 --      {$S TERes}
40  388 --      {-----------------------------------+   Assumes grafPort set appropriately upon entry, so
40  389 --      |   CalcSelLoc                      |       that TextWidth gives width in screen resolution
40  390 --      +-----------------------------------}
40  391 -- A    PROCEDURE TTEView.CalcSelLoc(VAR selectionRect: Rect);
40  392 --
40  393 --      CONST
40  394 --          teSysJust =         $BAC;               { System justification direction            }
40  395 --          kSlopToAllow =      36;
```

```
40  396 --
40  397 --      VAR
40  398 --          handleToText:       CharsHandle;
40  399 --          startOfSelection:   INTEGER;
40  400 --          endOfSelection:     INTEGER;
40  401 --          startOfCurrentLine: INTEGER;
40  402 --          startOfNextLine:    INTEGER;
40  403 --          charCount:          INTEGER;
40  404 --          selLine:            INTEGER;
40  405 --          justification:      INTEGER;
40  406 --          upToSelWidth:       INTEGER;
40  407 --          restWidth:          INTEGER;
40  408 --          lineHeight:         INTEGER;
40  409 --          fontAscent:         INTEGER;
40  410 --          theStyle:           TextStyle;
40  411 --          keyDirection:       SignedByte;
40  412 --          lastIsReturn:       BOOLEAN;
40  413 --          theMode:            INTEGER;
40  414 --          dontCare:           BOOLEAN;
40  415 --          savedPort:          GrafPtr;
40  416 --          theFontInfo:        FontInfo;
40  417 --
40  418 0- A    BEGIN
40  419 --          WITH fHTE^^ DO
40  420 1-              BEGIN
40  421 --              charCount := teLength;
40  422 --              startOfSelection := selStart;
40  423 --              endOfSelection := selEnd;
40  424 --              handleToText := CharsHandle(hText);
40  425 --              justification := just;
40  426 -1              END;
40  427 --
40  428 --      {$IFC qRangeCheck}{$R-}{$ENDC}
40  429 --          lastIsReturn := (startOfSelection = charCount) & (charCount > 0) &
40  430 --                          (handleToText^^[charCount - 1] = chReturn);
40  431 --      {$IFC qRangeCheck}{$R+}{$ENDC}
40  432 --
40  433 --          IF gConfiguration.hasStyleTextEdit THEN
40  434 1-              BEGIN
40  435 --              selectionRect.topLeft := TEGetPoint(startOfSelection, fHTE);
40  436 --              TEGetStyle(startOfSelection, theStyle, lineHeight, fontAscent, fHTE);
40  437 --              selectionRect.top := selectionRect.top - lineHeight;
40  438 --              selectionRect.botRight := TEGetPoint(endOfSelection, fHTE);
40  439 --
40  440 --              IF lastIsReturn THEN
40  441 2-                  BEGIN
40  442 --                  theMode := doAll;
40  443 --                  dontCare := ContinuousStyle(MAXINT, MAXINT, theMode, theStyle);
40  444 --                  GetPort(savedPort);
40  445 --                  SetPort(gWorkPort);
40  446 --                  SetPortTextStyle(theStyle);
40  447 --                  GetFontInfo(theFontInfo);           { What a pain…                    }
40  448 --                  SetPort(savedPort);
40  449 --                  selectionRect.top := selectionRect.bottom;
40  450 --                  WITH theFontInfo DO
40  451 --                      selectionRect.bottom := selectionRect.top + ascent + descent + leading;
40  452 -2                  END;
40  453 -1              END
40  454 --          ELSE
40  455 1-              BEGIN
40  456 --
40  457 --              WITH fHTE^^ DO          { for lineStarts and nLines }
40  458 2-                  BEGIN
40  459 --                  selLine := 1;
40  460 --                  WHILE (selLine < nLines) &
40  461 --                        ( startOfSelection >= lineStarts[selLine]) DO
40  462 --                      selLine := selLine + 1;
40  463 --                  selLine := selLine - 1;
40  464 --
40  465 --                  IF lastIsReturn |
40  466 --                     (justification <> teJustLeft) AND (startOfSelection = lineStarts[SUCC(selLine)]) THEN
40  467 --                      selLine := SUCC(selLine);
40  468 --
40  469 --                  selectionRect.top := (lineHeight * selLine) + fInset.top;
40  470 --                  startOfCurrentLine := lineStarts[selLine];
40  471 --                  IF selLine < nLines THEN
40  472 --                      startOfNextLine := lineStarts[SUCC(selLine)]
40  473 --                  ELSE
40  474 --                      startOfNextLine := charCount;
40  475 -2                  END;
40  476 --
40  477 --              { We could move the handle high first, but this routine is called frequently and
40  478 --                fragmentation shouldn't be a problem for the TextWidth call--the worst that will
40  479 --                happen is the Font Manager will have to substitute a font & we'll be off slightly.}
40  480 --              { ??? The following Text calls may be unnecessary, since this may only get done
40  481 --                after TextEdit has run and set the font already.  This appears not to be the case
40  482 --                for the Select All command, however, and is not true in general. }
40  483 --              TextFont(fHTE^^.txFont);
40  484 --              TextFace(fHTE^^.txFace);
40  485 --              TextSize(fHTE^^.txSize);
40  486 --              HLock(Handle(handleToText));
40  487 --
40  488 --              { Figure out where on the line the selection is.  If the Script Manager is
40  489 --                installed, use it.  Otherwise, try and figure it out ourselves. }
40  490 --              IF gConfiguration.hasScriptManager THEN
40  491 2-                  BEGIN
40  492 --                  { Unfortunately, we must get this every time, as the user can change script
40  493 --                    directions at a moment's notice. }
40  494 --                  IF startOfSelection = endOfSelection THEN
```

```
40  495 --                  keyDirection := GetScript(GetEnvirons(smKeyScript), smScriptRight)
40  496 --              ELSE
40  497 --                  keyDirection := smHilite;
40  498 --
40  499 --              upToSelWidth := Char2Pixel(Ptr(ORD(handleToText^)+startOfCurrentLine),
40  500 --                                  startOfNextLine-startOfCurrentLine, 0,
40  501 --                                  startOfSelection-startOfCurrentLine,
40  502 --                                  keyDirection);
40  503 --              restWidth := TextWidth(QDPtr(handleToText^), startOfCurrentLine,
40  504 --                                 startOfNextLine - startOfCurrentLine) - upToSelWidth;
40  505 -2              END
40  506 --          ELSE
40  507 2-              BEGIN
40  508 --              upToSelWidth := TextWidth(QDPtr(handleToText^), startOfCurrentLine,
40  509 --                                  startOfSelection - startOfCurrentLine);
40  510 --              restWidth := TextWidth(QDPtr(handleToText^), startOfSelection,
40  511 --                                  startOfNextLine - startOfSelection);
40  512 -2              END;
40  513 --          HUnlock(Handle(handleToText));
40  514 2-          CASE justification OF
40  515 --              teJustLeft:
40  516 --                  selectionRect.left := fInset.left + upToSelWidth;
40  517 --              teJustRight:
40  518 --                  selectionRect.left := fSize.h - fInset.right - restWidth;
40  519 --              { Following looks somewhat baroque, but avoids integer overflow }
40  520 --              teJustCenter:
40  521 --                  selectionRect.left := fSize.h DIV 2 - restWidth DIV 2 + upToSelWidth DIV 2;
40  522 -2              END;
40  523 --
40  524 --          WITH selectionRect DO
40  525 2-              BEGIN
40  526 --              left := MAX(left - 2, 0);
40  527 --              IF left <= kSlopToAllow THEN
40  528 --                  left := 0;                      { show start of line if close; gets around  }
40  529 --                                                  { …potential annoyance assoc w/typeahead    }
40  530 --  {$IFC FALSE}
40  531 --              top := MAX(top - 2, 0);
40  532 --  {$ENDC}
40  533 --              right := MIN(left + 8, fSize.h);
40  534 --              bottom := MIN(top + fHTE^^.lineHeight, fSize.v);
40  535 -2              END;
40  536 -1          END;
40  537 -0 A   END;
40  538 --
40  539 --      {$S TENonRes}
40  540 --      {-----------------------------------+
40  541 --      |   ContinuousStyle                 |
40  542 --      +-----------------------------------}
40  543 -- A   FUNCTION  TTEView.ContinuousStyle(firstChar, lastChar: INTEGER; VAR mode: INTEGER;
40  544 --                                      VAR aStyle: TextStyle): BOOLEAN;
40  545 --      VAR
40  546 --          oldSelStart,
40  547 --          oldSelEnd:      INTEGER;
40  548 0- A   BEGIN
40  549 --          WITH fHTE^^ DO
40  550 1-              BEGIN
40  551 --              oldSelStart := selStart;
40  552 --              oldSelEnd := selEnd;
40  553 -1              END;
40  554 --          SetSelect(firstChar, lastChar, fHTE);       { Use SetSelect so this is invisible }
40  555 --          ContinuousStyle := TEContinuousStyle(mode, aStyle, fHTE);
40  556 --          SetSelect(oldSelStart, oldSelEnd, fHTE);
40  557 -0 A   END;
40  558 --
40  559 --      {$S TERes}
40  560 --      {-----------------------------------+
40  561 --      |   ContainsClipType                |
40  562 --      +-----------------------------------}
40  563 -- A   FUNCTION  TTEView.ContainsClipType(aType: ResType): BOOLEAN; OVERRIDE;
40  564 0- A   BEGIN
40  565 --          ContainsClipType := (aType = 'TEXT');
40  566 -0 A   END;
40  567 --
40  568 --      { Put in resident segment since this gets called while typing }
40  569 --      {$S TERes}
40  570 --      {-----------------------------------+
40  571 --      |   DoBreakFollowing                |
40  572 --      +-----------------------------------}
40  573 -- A   FUNCTION  TTEView.DoBreakFollowing(vhs: VHSelect; prevBreak: VCoordinate;
40  574 --              VAR automatic: BOOLEAN): VCoordinate; OVERRIDE;
40  575 --      VAR
40  576 --          orthoVhs:       VHSelect;
40  577 --          possibleLoc:    INTEGER;
40  578 --          theStyles:      TEStyleHandle;
40  579 --          lhTab:          LHHandle;
40  580 --          height,
40  581 --          lineHeight:     INTEGER;
40  582 --          lineNo:         INTEGER;
40  583 --
40  584 0- A   BEGIN
40  585 --          orthoVhs := gOrthogonal[vhs];
40  586 --          automatic := TRUE;
40  587 --
40  588 --          possibleLoc := MIN(kMaxCoord, ORD4(prevBreak) + fPrintHandler.fViewPerPage.vh[orthoVhs]);
40  589 --
40  590 --          { We want to get rid of the on-screen margin represented by fMargin when printing, so
40  591 --            adjust things so that the portion of the view occupied by the screen margin doesn't
40  592 --            get printed. }
40  593 --          IF prevBreak = 0 THEN
```

```
40  594 --              possibleLoc := possibleLoc + fInset.topLeft.vh[orthoVhs];
40  595 --
40  596 --          IF (fStyleType = kWithStyle) AND (vhs = h) THEN
40  597 1-              BEGIN
40  598 --              IF (fLastPageBreak = prevBreak) THEN
40  599 2-                  BEGIN
40  600 --                  height := fLastPageBreak;
40  601 --                  lineNo := fLastLine;
40  602 -2                  END
40  603 --              ELSE
40  604 2-                  BEGIN
40  605 --                  height := fInset.topLeft.vh[orthoVhs];
40  606 --                  lineNo := 0;
40  607 -2                  END;
40  608 --
40  609 --              theStyles := GetStylHandle(fHTE);
40  610 --              lhTab := theStyles^^.lhTab;
40  611 --              WHILE lineNo < fHTE^^.nLines DO
40  612 2-                  BEGIN
40  613 --                  lineHeight := lhTab^^[lineNo].lhHeight;
40  614 --                  IF height + lineHeight <= possibleLoc THEN
40  615 --                      height := height + lineHeight
40  616 --                  ELSE
40  617 --                      LEAVE;
40  618 --                  lineNo := lineNo + 1;
40  619 -2                  END;
40  620 --              IF lineNo >= fHTE^^.nLines THEN
40  621 --                  possibleLoc := Max(possibleLoc, height)
40  622 --              ELSE
40  623 --                  possibleLoc := height;
40  624 --              fLastPageBreak := possibleLoc;
40  625 --              fLastLine := lineNo;
40  626 -1              END;
40  627 --
40  628 --
40  629 --          IF ORD4(possibleLoc + fInset.topLeft.vh[orthoVhs])
40  630 --                  >= fSize.vh[orthoVhs] THEN
40  631 --              DoBreakFollowing := fSize.vh[orthoVhs]
40  632 --          ELSE
40  633 --              DoBreakFollowing := possibleLoc;
40  634 -0 A    END;
40  635 --
40  636 --
40  637 --      {$S TENonRes}
40  638 --      {-----------------------------------+
40  639 --      |   DoCalcViewPerPage               |
40  640 --      +-----------------------------------}
40  641 -- A    PROCEDURE TTEView.DoCalcViewPerPage (VAR viewPerPage: VPoint); OVERRIDE;
40  642 --      VAR
40  643 --          vhs:            VHSelect;
40  644 --
40  645 0- A    BEGIN
40  646 --          INHERITED DoCalcViewPerPage(viewPerPage);   { Get max amount allowed given margins      }
40  647 --
40  648 --          {$IFC qDebug}
40  649 --          IF gDebugPrinting THEN
40  650 1-              BEGIN
40  651 --              WRITE('TTEView: incoming generic viewPerPage:');
40  652 --              WriteVPt(viewPerPage);
40  653 --              WRITELN;
40  654 -1              END;
40  655 --          {$ENDC}
40  656 --
40  657 --          IF (fStyleType <> kWithStyle) AND (fHTE <> NIL) THEN    { Adjust for integral # of lines per page   }
40  658 --              WITH fHTE^^ DO
40  659 --                  viewPerPage.v := lineHeight * (viewPerPage.v DIV lineHeight);
40  660 --
40  661 --          {$IFC qDebug}
40  662 --          IF gDebugPrinting THEN
40  663 1-              BEGIN
40  664 --              WRITE('TTEView: computed viewPerPage:');
40  665 --              WriteVPt(viewPerPage);
40  666 --              WRITELN;
40  667 -1              END;
40  668 --          {$ENDC}
40  669 -0 A    END;
40  670 --
40  671 --      {$S TERes}
40  672 --      {-----------------------------------+
40  673 --      |   DoIdle                          |
40  674 --      +-----------------------------------}
40  675 -- A    PROCEDURE TTEView.DoIdle(phase: IdlePhase); OVERRIDE;
40  676 --      VAR
40  677 --          aRect:  Rect;
40  678 --
40  679 0- A    BEGIN
40  680 --          IF (fHTE <> NIL) & fAcceptsChanges & Focus THEN
40  681 1-              BEGIN
40  682 --              TEIdle(fHTE);
40  683 --              fIdleFreq := MAX(GetCaretTime DIV 2,1); { Reset idle frequency in case user         }
40  684 -1              END;                                    { _changed cursor blink time               }
40  685 -0 A    END;
40  686 --
40  687 --
40  688 --      {$S TERes}
40  689 --      { The Tab character has no width in some fonts, and so can cause confusing screen feedback.
40  690 --          If you want to filter it out, OVERRIDE DoKeyCommand and either translate it to something
40  691 --          else, or throw it away (remembering to return gNoChanges). }
40  692 --      {-----------------------------------+
```

```
40  693 --      |   DoKeyCommand                    |
40  694 --      +-----------------------------------+
40  695 -- A   FUNCTION  TTEView.DoKeyCommand(ch: Char; aKeyCode: INTEGER; VAR info: EventInfo): TCommand; OVERRIDE;
40  696 --      VAR
40  697 --          aTypingCommand: TTETypingCommand;
40  698 --          needNewCommand: BOOLEAN;
40  699 --
40  700 0- A   BEGIN
40  701 --          DoKeyCommand := gNoChanges;
40  702 --          IF fAcceptsChanges THEN
40  703 1-              BEGIN
40  704 --              IF Focus THEN ;                         { Try to focus before accepting the key.    }
40  705 --              IF (ch >= chLeft) AND                   { If the user typed a cursor key, handle it }
40  706 --                (ch <= chDown) THEN                   { ...like a mouse down.                     }
40  707 2-                BEGIN
40  708 --                DoneTyping;                           { Like mousedown, further typing - new cmd  }
40  709 --                fSpecsChanged := TRUE;
40  710 --                TEKey(ch, fHTE);
40  711 --                ScrollSelectionIntoView;
40  712 --                EXIT(DoKeyCommand);
40  713 -2                END;
40  714 --
40  715 --              IF ch = chClear THEN                    { Hitting the Clear key is equivalent to    }
40  716 2-                BEGIN                                 { ...choosing Clear from the Edit menu.     }
40  717 --                {$H-}
40  718 --                WITH fHTE^^ DO                        { If nothing is selected, does nothing.     }
40  719 --                    IF selStart < selEnd THEN
40  720 --                        DoKeyCommand := DoMakeEditCommand(cClear);
40  721 --                EXIT(DoKeyCommand);
40  722 --                {$H+}
40  723 -2                END;
40  724 --
40  725 --              IF ch = chEnter THEN                    { If we get an Enter, throw it away         }
40  726 --                  EXIT(DoKeyCommand);
40  727 --
40  728 --              IF (ch <> chBackspace) AND                  { Check max size for text, and that we're   }
40  729 --                (fHTE^^.selStart = fHTE^^.selEnd) THEN    { ...not running out of memory              }
40  730 --                  IF ((fMaxChars - GetHandleSize(fText)) < 1) OR MemSpaceIsLow THEN
40  731 2-                    BEGIN
40  732 --                    SysBeep(0);
40  733 --                    EXIT(DoKeyCommand);               { Flush further keystrokes                  }
40  734 -2                    END;
40  735 --
40  736 --              needNewCommand := (fTypingCommand = NIL);
40  737 --              IF NOT needNewCommand THEN
40  738 --                  needNewCommand := fTypingCommand.fCompleted;
40  739 --
40  740 --              IF needNewCommand THEN
40  741 2-                  BEGIN
40  742 --                  aTypingCommand := DoMakeTypingCommand(ch);
40  743 --                  fTypingCommand := aTypingCommand;
40  744 --                  DoKeyCommand := aTypingCommand;
40  745 -2                  END
40  746 --              ELSE
40  747 --                  fTypingCommand.AddCharacter(ch);
40  748 --
40  749 -1              END;
40  750 -0 A   END;
40  751 --
40  752 --
40  753 --      {$S TESelCommand}
40  754 --      {-----------------------------------+
40  755 --      |   DoMakeEditCommand               |
40  756 --      +-----------------------------------}
40  757 -- A   FUNCTION  TTEView.DoMakeEditCommand(aCmdNumber: CmdNumber): TTECommand;
40  758 --      VAR
40  759 --          aCutCopyCommand:    TTECutCopyCommand;
40  760 --          aPasteCommand:      TTEPasteCommand;
40  761 --          aClearCommand:      TTECommand;
40  762 --
40  763 0- A   BEGIN
40  764 1-          CASE aCmdNumber OF
40  765 --              cCut,
40  766 2-              cCopy:  BEGIN
40  767 --                      New(aCutCopyCommand);
40  768 --                      FailNIL(aCutCopyCommand);
40  769 --                      aCutCopyCommand.ITECutCopyCommand(SELF, aCmdNumber);
40  770 --                      DoMakeEditCommand := aCutCopyCommand;
40  771 -2                      END;
40  772 --
40  773 2-              cPaste: BEGIN
40  774 --                      New(aPasteCommand);
40  775 --                      FailNIL(aPasteCommand);
40  776 --                      aPasteCommand.ITEPasteCommand(SELF);
40  777 --                      DoMakeEditCommand := aPasteCommand;
40  778 -2                      END;
40  779 --
40  780 2-              cClear: BEGIN
40  781 --                      New(aClearCommand);
40  782 --                      FailNIL(aClearCommand);
40  783 --                      aClearCommand.ITECommand(SELF, aCmdNumber, TRUE);
40  784 --                      DoMakeEditCommand := aClearCommand;
40  785 -2                      END;
40  786 --
40  787 -1              END;
40  788 -0 A   END;
40  789 --
40  790 --
40  791 --      {$S TERes}
```

```
40  792 --      {-----------------------------------+
40  793 --      |   DoMakeStyleCommand              |
40  794 --      +-----------------------------------}
40  795 -- A    FUNCTION  TTEView.DoMakeStyleCommand(aStyle: TextStyle; itsCmdNumber: CmdNumber;
40  796 --                            itsMode: INTEGER): TTEStyleCommand;
40  797 --      VAR
40  798 --          aTEStyleCommand:    TTEStyleCommand;
40  799 --
40  800 0- A    BEGIN
40  801 --          New(aTEStyleCommand);
40  802 --          FailNil(aTEStyleCommand);
40  803 --          aTEStyleCommand.ITEStyleCommand(SELF, aStyle, itsCmdNumber, itsMode);
40  804 --          DoMakeStyleCommand := aTEStyleCommand;
40  805 -0 A    END;
40  806 --
40  807 --
40  808 --      {$S TERes}
40  809 --      {-----------------------------------+
40  810 --      |   DoMakeTypingCommand             |
40  811 --      +-----------------------------------}
40  812 -- A    FUNCTION  TTEView.DoMakeTypingCommand(ch: Char): TTETypingCommand;
40  813 --      VAR
40  814 --          aTypingCommand:     TTETypingCommand;
40  815 --
40  816 0- A    BEGIN
40  817 --          New(aTypingCommand);
40  818 --          FailNil(aTypingCommand);
40  819 --          aTypingCommand.ITETypingCommand(SELF, ch);
40  820 --          DoMakeTypingCommand := aTypingCommand;
40  821 -0 A    END;
40  822 --
40  823 --
40  824 --      {$S TESelCommand}
40  825 --      {-----------------------------------+
40  826 --      |   DoMenuCommand                   |
40  827 --      +-----------------------------------}
40  828 -- A    FUNCTION  TTEView.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
40  829 --      VAR
40  830 --          command:            TCommand;
40  831 --          aCutCopyCommand:    TTECutCopyCommand;
40  832 --          aPasteCommand:      TTEPasteCommand;
40  833 --          nChars:             LONGINT;
40  834 --          dataType:           ResType;
40  835 --
40  836 0- A    BEGIN
40  837 --          DoMenuCommand := gNoChanges;
40  838 1-          CASE aCmdNumber OF
40  839 --              cCut,
40  840 --              cCopy,
40  841 --              cClear:
40  842 --                  DoMenuCommand := DoMakeEditCommand(aCmdNumber);
40  843 --
40  844 --              cPaste:
40  845 2-                  BEGIN
40  846 --                      nChars := gApplication.GetDataToPaste(NIL, dataType);
40  847 --                      IF nChars < 0 THEN
40  848 --          {$IFC qDebug}
40  849 --                          ProgramBreak('Couldn''t get data to paste') { ??? }
40  850 --          {$ENDC}
40  851 --                      ELSE
40  852 3-                          BEGIN
40  853 --                          IF nChars - (fHTE^^.selEnd - fHTE^^.selStart) >
40  854 --                            fMaxChars - GetHandleSize(fText) THEN
40  855 --                              StdAlert(phTooManyChars)
40  856 --                          ELSE
40  857 --                              DoMenuCommand := DoMakeEditCommand(aCmdNumber);
40  858 -3                          END;
40  859 -2                  END;
40  860 --
40  861 --              cSelectAll:
40  862 2-                  BEGIN
40  863 --                      IF Focus THEN
40  864 3-                          BEGIN
40  865 --                          TESetSelect(0, fHTE^^.teLength, fHTE);
40  866 --                          DoneTyping;
40  867 --                          fSpecsChanged := TRUE;
40  868 --                          ScrollSelectionIntoView;
40  869 -3                          END;
40  870 -2                  END;
40  871 --
40  872 --              OTHERWISE
40  873 --                  DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
40  874 --
40  875 -1              END;
40  876 -0 A    END;
40  877 --
40  878 --
40  879 --      {$S TERes}
40  880 --      {-----------------------------------+
40  881 --      |   DoMouseCommand                  |
40  882 --      +-----------------------------------}
40  883 -- A    FUNCTION  TTEView.DoMouseCommand(VAR theMouse: Point; VAR info: EventInfo;
40  884 --                            VAR hysteresis: Point): TCommand; OVERRIDE;
40  885 0- A    BEGIN
40  886 --          IF Focus THEN
40  887 1-              BEGIN
40  888 --              gCurrTEView := SELF;
40  889 --              DoneTyping;                         { Mousedown terminates the Typing command   }
40  890 --              fSpecsChanged := TRUE;
```

```
40  891 --              TEClick(theMouse, info.theShiftKey, fHTE);
40  892 -1             END;
40  893 --          DoMouseCommand := gNoChanges;
40  894 -0 A    END;
40  895 --
40  896 --
40  897 --      {$S TENonRes}
40  898 --      {-----------------------------------+   Some operation has taken place which precludes
40  899 --      |   DoneTyping                      |       further typing
40  900 --      +-----------------------------------}
40  901 -- A    PROCEDURE TTEView.DoneTyping;
40  902 0- A    BEGIN
40  903 --          IF fTypingCommand <> NIL THEN
40  904 --              fTypingCommand.CompleteTyping;
40  905 -0 A    END;
40  906 --
40  907 --      {$S TERes}
40  908 --      {-----------------------------------+
40  909 --      |   DoSetCursor                     |
40  910 --      +-----------------------------------}
40  911 -- A    FUNCTION  TTEView.DoSetCursor(localPoint: Point;
40  912 --                      cursorRgn: RgnHandle): BOOLEAN; OVERRIDE;
40  913 --      VAR
40  914 --          qdExtent:   Rect;
40  915 --
40  916 0- A    BEGIN
40  917 --          IF SELF <> gClipView THEN
40  918 1-              BEGIN
40  919 --              UseROMMap(TRUE);
40  920 --              SetCursor(GetCursor(iBeamCursor)^^);
40  921 --              GetQDExtent(qdExtent);
40  922 --              RectRgn(cursorRgn, qdExtent);
40  923 --              DoSetCursor := TRUE;
40  924 -1              END
40  925 --          ELSE
40  926 --              DoSetCursor := FALSE;
40  927 -0 A    END;
40  928 --
40  929 --
40  930 --      {$S TEPrint}
40  931 --      {-----------------------------------+   Get rid of the on-screen margin (fMargin) when
40  932 --      |   DoSetPageOffset                 |       printing by adjusting view
40  933 --      +-----------------------------------}
40  934 -- A    PROCEDURE TTEView.DoSetPageOffset(coord: VPoint); OVERRIDE;
40  935 --      VAR
40  936 --          vhs:    VHSelect;
40  937 --
40  938 0- A    BEGIN
40  939 --          INHERITED DoSetPageOffset(coord);
40  940 --          FOR vhs := v to h DO
40  941 --              IF coord.vh[vhs] = 0 THEN
40  942 --                  gPageOffset.vh[vhs] := gPageOffset.vh[vhs] + fInset.topLeft.vh[vhs];
40  943 -0 A    END;
40  944 --
40  945 --
40  946 --      {$S TERes}
40  947 --      {-----------------------------------+
40  948 --      |   DoSetupMenus                    |
40  949 --      +-----------------------------------}
40  950 -- A    PROCEDURE TTEView.DoSetupMenus; OVERRIDE;
40  951 --      VAR
40  952 --          manyChars: BOOLEAN;
40  953 --
40  954 0- A    BEGIN
40  955 --          INHERITED DoSetupMenus;
40  956 --
40  957 --          WITH fHTE^^ DO
40  958 --              manyChars := selStart < selEnd;
40  959 --          IF NOT MemSpaceIsLow THEN
40  960 1-              BEGIN
40  961 --              IF fAcceptsChanges THEN                 { One way or another, we can paste text     }
40  962 --                  CanPaste('TEXT');                   { If styles exist, all the better           }
40  963 --
40  964 --              Enable(cCopy, manyChars);
40  965 -1              END;
40  966 --          Enable(cSelectAll, (fHTE^^.teLength > 0));
40  967 --
40  968 --          { We enable Cut even if space is low, since it's nice to be able to rescue some of
40  969 --            the stuff you have to delete even if space is low.  Note that it is possible to
40  970 --            get into the "can't do any commands" situation as a result.  You should be able
40  971 --            to close and save the big document, then save the rescued text elsewhere, however. }
40  972 --
40  973 --          Enable(cCut, manyChars AND fAcceptsChanges);
40  974 --          Enable(cClear, manyChars AND fAcceptsChanges);
40  975 -0 A    END;
40  976 --
40  977 --
40  978 --      {$S TERes}
40  979 --      {-----------------------------------+
40  980 --      |   Draw                            |
40  981 --      +-----------------------------------}
40  982 -- A    PROCEDURE TTEView.Draw(area: Rect); OVERRIDE;
40  983 --      VAR
40  984 --          savedPort:      GrafPtr;
40  985 --          aPort:          GrafPtr;
40  986 --          isActive:       BOOLEAN;
40  987 0- A    BEGIN
40  988 --          IF gPrinting OR gDrawingPictScrap THEN
40  989 1-              BEGIN
```

```
40  990 --              savedPort := fHTE^^.inPort;
40  991 --              GetPort(aPort);
40  992 --              fHTE^^.inPort := aPort;
40  993 --
40  994 --              isActive := BOOLEAN(fHTE^^.active);
40  995 --              IF isActive THEN                            { If active…                                }
40  996 2-                  BEGIN                                   { …prevent selection from being drawn.      }
40  997 --      {$IFC qDebug}
40  998 --                  UseTempRgn('TTEView.Draw');
40  999 --      {$ENDC}
40 1000 --                  GetClip(gTempRgn);
40 1001 --                  ClipRect(gZeroRect);
40 1002 --                  TEDeactivate(fHTE);
40 1003 --                  SetClip(gTempRgn);
40 1004 --      {$IFC qDebug}
40 1005 --                  DoneWithTempRgn;
40 1006 --      {$ENDC}
40 1007 -2                  END;
40 1008 --
40 1009 --              TEUpdate(area, fHTE);
40 1010 --
40 1011 --              IF isActive THEN
40 1012 2-                  BEGIN
40 1013 --      {$IFC qDebug}
40 1014 --                  UseTempRgn('TTEView.Draw');
40 1015 --      {$ENDC}
40 1016 --                  GetClip(gTempRgn);
40 1017 --                  ClipRect(gZeroRect);
40 1018 --                  TEActivate(fHTE);
40 1019 --                  SetClip(gTempRgn);
40 1020 --      {$IFC qDebug}
40 1021 --                  DoneWithTempRgn;
40 1022 --      {$ENDC}
40 1023 -2                  END;
40 1024 --
40 1025 --              fHTE^^.inPort := savedPort;
40 1026 -1              END
40 1027 --
40 1028 --          ELSE { normal screen update handled by TextEdit directly }
40 1029 --              TEUpdate(area, fHTE);
40 1030 -0 A    END;
40 1031 --
40 1032 --
40 1033 --      {$S TENonRes}
40 1034 --      {-----------------------------------+
40 1035 --      |   ExtractStyles                   |
40 1036 --      +-----------------------------------}
40 1037 -- A    PROCEDURE TTEView.ExtractStyles(VAR theStyles: TEStyleHandle; VAR theElements: STHandle);
40 1038 --
40 1039 0- A    BEGIN
40 1040 --          theStyles := GetStylHandle(fHTE);
40 1041 --          theElements := theStyles^^.styleTab;
40 1042 -0 A    END;
40 1043 --
40 1044 --
40 1045 --      {$S TENonRes}
40 1046 --      {-----------------------------------+
40 1047 --      |   ExtractText                     |   For consistency
40 1048 --      +-----------------------------------}
40 1049 -- A    FUNCTION  TTEView.ExtractText: Handle;
40 1050 --
40 1051 0- A    BEGIN
40 1052 --          ExtractText := fText;
40 1053 -0 A    END;
40 1054 --
40 1055 --      {$S TENonRes}
40 1056 --      {-----------------------------------+
40 1057 --      |   GetPrintExtent                  |
40 1058 --      +-----------------------------------}
40 1059 -- A    PROCEDURE TTEView.GetPrintExtent (VAR printExtent: VRect); OVERRIDE;
40 1060 --
40 1061 0- A    BEGIN
40 1062 --          INHERITED GetPrintExtent(printExtent);
40 1063 --          WITH fInset DO
40 1064 1-              BEGIN
40 1065 --              printExtent.top := printExtent.top + top;
40 1066 --              printExtent.left := printExtent.left + left;
40 1067 --              printExtent.bottom := printExtent.bottom - bottom;
40 1068 --              printExtent.right := printExtent.right - right;
40 1069 -1              END;
40 1070 -0 A    END;
40 1071 --
40 1072 --      {$S TEClipboard}
40 1073 --      {-----------------------------------+
40 1074 --      |   GivePasteData                   |
40 1075 --      +-----------------------------------}
40 1076 -- A    FUNCTION  TTEView.GivePasteData(aDataHandle: Handle; dataType: ResType): LONGINT; OVERRIDE;
40 1077 --      VAR
40 1078 --          oldStart:   INTEGER;
40 1079 --          oldEnd:     INTEGER;
40 1080 --          aSize:      LONGINT;
40 1081 --          aHandle:    Handle;
40 1082 --          err:        OSErr;
40 1083 --          savedPerm:  BOOLEAN;
40 1084 --          fi:         FailInfo;
40 1085 --
40 1086 --          {-----------------------------------+
40 1087 --          |   HdlGivePasteFailed              |
40 1088 --          +-----------------------------------}
```

```
40 1089 -- B       PROCEDURE HdlGivePasteFailed(error: OSErr; message: LONGINT);
40 1090 0- B       BEGIN
40 1091 --             IF aHandle <> NIL THEN
40 1092 1-                 BEGIN
40 1093 --                 HUnlock(aHandle);
40 1094 --                 DisposHandle(aHandle);
40 1095 -1                 END;
40 1096 --             IF error = phStylesTooBig THEN
40 1097 1-                 BEGIN
40 1098 --                 StdAlert(phStylesTooBig);
40 1099 --                 GivePasteData := noErr;
40 1100 -1                 END
40 1101 --             ELSE
40 1102 --                 GivePasteData := error;
40 1103 -0 B       END;
40 1104 --
40 1105 0- A   BEGIN
40 1106 --         savedPerm := FALSE;
40 1107 --         aSize := 0;                                 { Assume the worst                          }
40 1108 --         aHandle := NIL;
40 1109 --         CatchFailures(fi, HdlGivePasteFailed);
40 1110 --
40 1111 --         IF dataType = 'TEXT' THEN
40 1112 1-             BEGIN
40 1113 --             aSize := GetHandleSize(fText);
40 1114 --             IF aDataHandle <> NIL THEN
40 1115 2-                 BEGIN
40 1116 --                 savedPerm := PermAllocation(TRUE);      { Passed in handle remains a permanent one  }
40 1117 --                 SetHandleSize(aDataHandle, aSize);
40 1118 --                 err := MemError;                        { Save MemError & test it later because     }
40 1119 --                 savedPerm := PermAllocation(savedPerm); { _PermAllocation may change it.            }
40 1120 --                 IF err <> noErr THEN
40 1121 --                     Failure(err, 0);
40 1122 --                 BlockMove(fText^, aDataHandle^, aSize);
40 1123 -2                 END;
40 1124 -1             END
40 1125 --         ELSE IF dataType = 'styl' THEN
40 1126 1-             BEGIN
40 1127 --             IF fStyleType = kWithStyle THEN
40 1128 --                 IF NOT SpaceForStyles(0, MAXINT) THEN
40 1129 --                     Failure(noErr, 0)                       { We'll accept this error in worst case     }
40 1130 --                 ELSE
40 1131 2-                     BEGIN
40 1132 --                     WITH fHTE^^ DO
40 1133 3-                         BEGIN
40 1134 --                         oldStart := selStart;
40 1135 --                         oldEnd := selEnd;
40 1136 -3                         END;
40 1137 --                     SetSelect(0, MAXINT, fHTE);
40 1138 --                     aHandle := Handle(GetStylScrap(fHTE));
40 1139 --                     SetSelect(oldStart, oldEnd, fHTE);
40 1140 --
40 1141 --                     IF aHandle <> NIL THEN
40 1142 3-                         BEGIN
40 1143 --                         aSize := GetHandleSize(aHandle);
40 1144 --                         IF aDataHandle <> NIL THEN
40 1145 4-                             BEGIN
40 1146 --                             savedPerm := PermAllocation(TRUE);
40 1147 --                             MoveHHi(aHandle);           { Try to prevent fragmentation, in case     }
40 1148 --                             HLock(aHandle);             { Can't move while we're copying it!        }
40 1149 --
40 1150 --                             err := PtrToXHand(aHandle^, { Copy styles into user-supplied handle     }
40 1151 --                               aDataHandle, aSize);
40 1152 --                             HUnlock(aHandle);           { Okay for it to move again                 }
40 1153 --                             savedPerm := PermAllocation(savedPerm);
40 1154 --                             IF err <> noErr THEN        { Maybe enough for one copy, but not two!   }
40 1155 --                                 Failure(phStylesTooBig, 0);
40 1156 -4                             END;
40 1157 --                         DisposHandle(aHandle);
40 1158 --                         aHandle := NIL;
40 1159 -3                         END
40 1160 --                     ELSE IF aDataHandle <> NIL THEN     { Hmm.  There _was_ enough memory, but the  }
40 1161 --                         Failure(phStylesTooBig, 0);     { _heap is probably pretty fragmented       }
40 1162 --
40 1163 -2                     END;
40 1164 -1             END
40 1165 --         ELSE
40 1166 --             Failure(noTypeErr, 0);
40 1167 --
40 1168 --         FailSpaceIsLow;
40 1169 --         Success(fi);
40 1170 --         GivePasteData := aSize;
40 1171 -0 A   END;
40 1172 --
40 1173 --
40 1174 --     {$S TENonRes}
40 1175 --     {-----------------------------------+
40 1176 --     |   InstallSelection                |
40 1177 --     +-----------------------------------}
40 1178 -- A   PROCEDURE TTEView.InstallSelection(wasActive, beActive: BOOLEAN); OVERRIDE;
40 1179 --
40 1180 0- A   BEGIN
40 1181 --         IF Focus THEN ;                             { Try to focus because TEActivate may       }
40 1182 --                                                     { _erase the caret.                         }
40 1183 --         IF beActive THEN
40 1184 1-             BEGIN
40 1185 --             IF gConfiguration.hasScriptManager THEN
40 1186 --                 SetKeyScript(Font2Script(fTextStyle.tsFont));
40 1187 --             TEActivate(fHTE);
```

```
40 1188 --              gCurrTEView := SELF;
40 1189 -1              END
40 1190 --          ELSE
40 1191 1-              BEGIN
40 1192 --              TEDeactivate(fHTE);
40 1193 --              DoneTyping;
40 1194 --              fSpecsChanged := TRUE;
40 1195 -1              END;
40 1196 -0 A    END;
40 1197 --
40 1198 --      {$S TEOpen}
40 1199 --      {-----------------------------------+
40 1200 --      |   MakeTERecord                    |   Call this after grafPort is properly set
40 1201 --      +-----------------------------------}
40 1202 -- A    PROCEDURE TTEView.MakeTERecord;
40 1203 --      VAR
40 1204 --          anHTE:      TEHandle;
40 1205 --          dest:       Rect;
40 1206 --          oldPort:    GrafPtr;
40 1207 --          fi:         FailInfo;
40 1208 --
40 1209 -- B        PROCEDURE HdlMakeFailed(error: OSErr; message: LONGINT);
40 1210 0- B        BEGIN
40 1211 --              Free;
40 1212 -0 B        END;
40 1213 --
40 1214 --
40 1215 0- A    BEGIN
40 1216 --          GetPort(oldPort);
40 1217 --          SetPort(gWorkPort);
40 1218 --          SetPortTextStyle(fTextStyle);
40 1219 --
40 1220 --          dest.topLeft := fInset.topLeft;
40 1221 --          dest.right := fSize.h - fInset.right;
40 1222 --          dest.bottom := fSize.v - fInset.bottom;
40 1223 --
40 1224 --          IF fStyleType = kWithStyle THEN
40 1225 --              anHTE := TEStylNew(dest, dest)          { Open a styled record if requested         }
40 1226 --          ELSE
40 1227 --              anHTE := TENew(dest, dest);             { …otherwise, do it the old way             }
40 1228 --
40 1229 --          SetPort(oldPort);
40 1230 --
40 1231 --          CatchFailures(fi, HdlMakeFailed);           { In case we couldn't create the TEHandle.  }
40 1232 --          FailNIL(anHTE);                             { Make sure we actually created one         }
40 1233 --          fHTE := anHTE;                              { We did, so save off TE handle             }
40 1234 --          SetJustification(fJustification,
40 1235 --                           kDontRedraw);              { Set justification to requested value      }
40 1236 --          ChangeWrap(fAutoWrap, FALSE);               { Install auto wrap (or CR only)            }
40 1237 --          FailNoReserve;                              { Got to have some reserve tank             }
40 1238 --          Success(fi);
40 1239 --
40 1240 --          BeInPort(GetGrafPort);                      { Associate with real port                  }
40 1241 -0 A    END;
40 1242 --
40 1243 --      {$S TERes}
40 1244 --      {-----------------------------------+
40 1245 --      |   RecalcText                      |   Pretty simple, anymore
40 1246 --      +-----------------------------------}
40 1247 -- A    PROCEDURE TTEView.RecalcText;
40 1248 --
40 1249 0- A    BEGIN
40 1250 --          TECalText(fHTE);
40 1251 -0 A    END;
40 1252 --
40 1253 --
40 1254 --      {$S TENonRes}
40 1255 --      {-----------------------------------+
40 1256 --      |   Resize                          |
40 1257 --      +-----------------------------------}
40 1258 -- A    PROCEDURE TTEView.Resize (width, height: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
40 1259 --      VAR
40 1260 --          needCalText:    BOOLEAN;
40 1261 --          r:              Rect;
40 1262 --          oldSize:        VPoint;
40 1263 0- A    BEGIN
40 1264 --          oldSize := fSize;
40 1265 --          INHERITED Resize(width, height, invalidate);
40 1266 --          IF fHTE <> NIL THEN
40 1267 1-              BEGIN
40 1268 --              r.topLeft := fInset.topLeft;
40 1269 --              r.right := fSize.h - fInset.right;
40 1270 --              r.bottom := fSize.v - fInset.bottom;
40 1271 --
40 1272 --              needCalText := (r.right <> fHTE^^.destRect.right);
40 1273 --              StuffTERects(r);
40 1274 --              IF needCalText THEN
40 1275 2-                  BEGIN
40 1276 --                  RecalcText;
40 1277 --                  SynchView(kDontRedraw);
40 1278 --                  IF invalidate |
40 1279 --                     (fAutoWrap AND (fSize.h <> oldSize.h) AND (fSize.v <> oldSize.v)) THEN
40 1280 --                      ForceRedraw;
40 1281 -2                  END;
40 1282 -1              END;
40 1283 -0 A    END;
40 1284 --
40 1285 --
40 1286 --      {$S TERes}
```

```
40 1287 --      {-----------------------------------+
40 1288 --      |   ScrollSelectionIntoView         |
40 1289 --      +-----------------------------------}
40 1290 -- A    PROCEDURE TTEView.ScrollSelectionIntoView;
40 1291 --      CONST
40 1292 --          kMinAhead =     96;
40 1293 --      VAR
40 1294 --          selLoc:         Point;
40 1295 --          vhs:            VHSelect;
40 1296 --          newLoc:         Point;
40 1297 --          newSelLoc:      Point;
40 1298 --          newSelRect:     Rect;
40 1299 --          minToSee:       Point;
40 1300 --          visRect:        Rect;
40 1301 --          vNewSelRect:    VRect;
40 1302 --
40 1303 0- A    BEGIN
40 1304 --          IF (fScroller <> NIL) & Focus THEN          { Can't scroll selection if we don't have   }
40 1305 1-              BEGIN                                   { a scroller!                               }
40 1306 --              GetVisibleRect(visRect);
40 1307 --              CalcSelLoc(newSelRect);
40 1308 --      {$IFC qDebug}
40 1309 --              IF gIntenseDebugging THEN
40 1310 2-                  BEGIN
40 1311 --                  WRITE('Incoming SelLoc: '); WritePt(selLoc); WRITELN;
40 1312 --                  WRITE('Visible Rect was: '); WriteRect(visRect);
40 1313 --                  WRITE('; Sel Rect was: '); WriteRect(newSelRect); WRITELN;
40 1314 -2                  END;
40 1315 --      {$ENDC}
40 1316 --
40 1317 --              IF NOT RectsNest(visRect, newSelRect) THEN
40 1318 2-                  BEGIN
40 1319 --                  SetPt(minToSee,                     { Scroll the selection into view.           }
40 1320 --                          MIN(kMinAhead, fSize.h-newSelRect.left),
40 1321 --                          LengthRect(newSelRect, v));
40 1322 --      {$IFC qDebug}
40 1323 --                  IF gIntenseDebugging THEN
40 1324 3-                      BEGIN
40 1325 --                      WrLblRect('RevealRect: r', newSelRect);
40 1326 --                      WrLblPt(', minToSee', minToSee);
40 1327 --                      WRITELN;
40 1328 -3                      END;
40 1329 --      {$ENDC}
40 1330 --                  QDToViewRect(newSelRect, vNewSelRect);
40 1331 --                  RevealRect(vNewSelRect, minToSee, kRedraw);
40 1332 --                  IF Focus THEN ; { Refocus in newly-scrolled position }
40 1333 -2                  END;
40 1334 -1              END
40 1335 --          ELSE
40 1336 --      {$IFC NOT qNeedsROM128K}
40 1337 --              IF gConfiguration.hasROM128K THEN
40 1338 --      {$ENDC}
40 1339 --                  IF NOT fAutoWrap & (fHTE <> NIL) THEN
40 1340 --                      TESelView(fHTE);
40 1341 -0 A    END;
40 1342 --
40 1343 --
40 1344 --      {$S TENonRes}
40 1345 --      {-----------------------------------+
40 1346 --      |   SetJustification                |
40 1347 --      +-----------------------------------}
40 1348 -- A    PROCEDURE TTEView.SetJustification(newJust: INTEGER; redraw: BOOLEAN);
40 1349 0- A    BEGIN
40 1350 --          TESetJust(newJust, fhTE);
40 1351 --          fJustification := newJust;
40 1352 --          IF redraw THEN
40 1353 --              ForceRedraw;
40 1354 -0 A    END;
40 1355 --
40 1356 --      {$S TENonRes}
40 1357 --      {-----------------------------------+
40 1358 --      |   SetOneStyle                     |
40 1359 --      +-----------------------------------}
40 1360 -- A    PROCEDURE TTEView.SetOneStyle (theStart, theEnd, theMode: INTEGER; theStyle: TextStyle; redraw: BOOLEAN);
40 1361 --      VAR
40 1362 --          saveStart:      INTEGER;
40 1363 --          saveEnd:        INTEGER;
40 1364 --          savedPort:      GrafPtr;
40 1365 --          fInfo:          FontInfo;
40 1366 --          newStyle:       TextStyle;
40 1367 --
40 1368 0- A    BEGIN
40 1369 --          gFocusedView := NIL; { ??? THIS SHOULDN'T BE NECESSARY! ??? }
40 1370 --          IF Focus THEN ;
40 1371 --          IF fStyleType = kWithStyle THEN
40 1372 1-              BEGIN
40 1373 --              WITH fHTE^^ DO
40 1374 2-                  BEGIN
40 1375 --                  saveStart := selStart;
40 1376 --                  saveEnd := selEnd;
40 1377 -2                  END;
40 1378 --              SetSelect(theStart, theEnd, fHTE);
40 1379 --              TESetStyle(theMode, theStyle, redraw, fHTE);
40 1380 --              SetSelect(saveStart, saveEnd, fHTE);
40 1381 -1              END
40 1382 --          ELSE
40 1383 1-              BEGIN
40 1384 --              IF theMode = doAll THEN
40 1385 --                  newStyle := theStyle
```

```
40 1386 --              ELSE
40 1387 2-                  BEGIN
40 1388 --                  newStyle := fTextStyle;
40 1389 --                  IF BAND(theMode, doFont) <> 0 THEN
40 1390 3-                      BEGIN
40 1391 --                      newStyle.tsFont := theStyle.tsFont;
40 1392 --                      IF gConfiguration.hasScriptManager THEN     { …if Script Mgr is installed, change   }
40 1393 --                          KeyScript(Font2Script(newStyle.tsFont));    { …keybd input system to match new font }
40 1394 -3                      END;
40 1395 --                  IF BAND(theMode, doFace) <> 0 THEN
40 1396 --                      newStyle.tsFace := theStyle.tsFace;
40 1397 --                  IF BAND(theMode, doColor) <> 0 THEN
40 1398 --                      newStyle.tsColor := theStyle.tsColor;
40 1399 --                  IF BAND(theMode, addSize) <> 0 THEN
40 1400 --                      newStyle.tsSize := newStyle.tsSize + theStyle.tsSize
40 1401 --                  ELSE IF BAND(theMode, doSize) <> 0 THEN
40 1402 --                      newStyle.tsSize := theStyle.tsSize;
40 1403 -2                  END;
40 1404 --
40 1405 --              GetPort(savedPort);
40 1406 --              SetPort(gWorkPort);
40 1407 --              SetPortTextStyle(newStyle);
40 1408 --              GetFontInfo(fInfo);                             { Need to get font's height and ascent. }
40 1409 --              SetPort(savedPort);
40 1410 --
40 1411 --              WITH fHTE^^, newStyle, fInfo DO
40 1412 2-                  BEGIN
40 1413 --                  txSize := tsSize;
40 1414 --                  txFont := tsFont;
40 1415 --                  txFace := tsFace;
40 1416 --                  fontAscent := ascent;
40 1417 --                  lineHeight := ascent + descent + leading;
40 1418 --                  SetIfColor(tsColor);
40 1419 -2                  END;
40 1420 --              fTextStyle := newStyle;
40 1421 -1              END;
40 1422 --
40 1423 --          IF TRUE { (fStyleType = kWithoutStyle) OR (theStart <> theEnd) } THEN
40 1424 1-              BEGIN
40 1425 --              RecalcText;
40 1426 --              SynchView(redraw AND (fStyleType = kWithStyle));
40 1427 --              IF redraw AND (fStyleType = kWithoutStyle) THEN
40 1428 --                  ForceRedraw;
40 1429 -1              END;
40 1430 --          fSpecsChanged := TRUE;
40 1431 -0 A   END;
40 1432 --
40 1433 --      {$S TENonRes}
40 1434 --      {-----------------------------------+
40 1435 --      |   SetText                         |   Convenience routine, to avoid creating a handle
40 1436 --      +-----------------------------------}
40 1437 -- A   PROCEDURE TTEView.SetText(theText: Str255);
40 1438 --      VAR
40 1439 --          theTextHandle:      Handle;
40 1440 0- A   BEGIN
40 1441 --          IF fHTE <> NIL THEN                         { If we're replacing text, styles are kaput }
40 1442 1-              BEGIN
40 1443 --              FailOSErr(PtrToHand(@theText[1], theTextHandle, LENGTH(theText)));
40 1444 --              StuffText(theTextHandle);
40 1445 -1              END;
40 1446 -0 A   END;
40 1447 --
40 1448 --      {$S TENonRes}
40 1449 --      {-----------------------------------+
40 1450 --      |   ShowReverted                    |
40 1451 --      +-----------------------------------}
40 1452 -- A   PROCEDURE TTEView.ShowReverted; OVERRIDE;
40 1453 0- A   BEGIN
40 1454 --          RecalcText;
40 1455 --          fLastHeight := 0;
40 1456 --          INHERITED ShowReverted;
40 1457 -0 A   END;
40 1458 --
40 1459 --
40 1460 --      {$S TENonRes}
40 1461 --      {-----------------------------------+   Return TRUE if there is enough memory to hold the
40 1462 --      |   SpaceForStyles                  |       styles associated with the given range of text
40 1463 --      +-----------------------------------}
40 1464 -- A   FUNCTION  TTEView.SpaceForStyles(rangeStart, rangeEnd: LONGINT): BOOLEAN;
40 1465 --      VAR
40 1466 --          h:      Handle;
40 1467 0- A   BEGIN
40 1468 --          h := NewPermHandle(TENumStyles(rangeStart, rangeEnd, fHTE) * SIZEOF(ScrpSTElement) + 2);
40 1469 --          IF (h = NIL) THEN
40 1470 1-              BEGIN
40 1471 --              StdAlert(phStylesTooBig);
40 1472 --              SpaceForStyles := FALSE;
40 1473 -1              END
40 1474 --          ELSE
40 1475 1-              BEGIN
40 1476 --              DisposHandle(h);                        { Release memory back to the system         }
40 1477 --              SpaceForStyles := TRUE;
40 1478 -1              END;
40 1479 -0 A   END;
40 1480 --
40 1481 --      {$S TENonRes}
40 1482 --      {-----------------------------------+
40 1483 --      |   StuffStyles                     |
40 1484 --      +-----------------------------------}
```

```
40 1485 -- A   PROCEDURE TTEView.StuffStyles(theStyles: TEStyleHandle; theElements: STHandle);
40 1486 --     VAR
40 1487 --         oldStyles:      TEStyleHandle;
40 1488 --         oldElements:    STHandle;
40 1489 --         nullStyles:     NullStHandle;
40 1490 --         lhElements:     LHHandle;
40 1491 --
40 1492 0- A   BEGIN
40 1493 --         IF (fStyleType = kWithStyle) & (fHTE <> NIL) THEN
40 1494 1-             BEGIN
40 1495 --             oldStyles := GetStylHandle(fHTE);       { We have to save off the null style handle }
40 1496 --             nullStyles := oldStyles^^.nullStyle;    { ...because TE expects a valid one, and it's }
40 1497 --                                                     { ...pointless for us to recreate one for it  }
40 1498 --             lhElements := oldStyles^^.lhTab;        { Same goes for line heights table            }
40 1499 --             oldElements := oldStyles^^.styleTab;
40 1500 --
40 1501 --             SetStylHandle(theStyles, fHTE);
40 1502 --
40 1503 --             theStyles^^.nullStyle := nullStyles;    { Replace null style handle                   }
40 1504 --             theStyles^^.lhTab := lhElements;        { Replace line heights table handle           }
40 1505 --             theStyles^^.styleTab := theElements;    { Stuff STElements handle                     }
40 1506 --
40 1507 --             DisposHandle(Handle(oldStyles));
40 1508 --             DisposHandle(Handle(oldElements));
40 1509 --             RecalcText;
40 1510 -1             END;
40 1511 -0 A   END;
40 1512 --
40 1513 --
40 1514 --     {$S TENonRes}
40 1515 --     {-----------------------------------+   For consistency.  We don't do a recalc here since
40 1516 --     |   StuffText                       |       the user might immediately StuffStyles
40 1517 --     +-----------------------------------}
40 1518 -- A   PROCEDURE TTEView.StuffText(theText: Handle);
40 1519 --     VAR
40 1520 --         styles:     TEStyleHandle;
40 1521 0- A   BEGIN
40 1522 --         IF fHTE <> NIL THEN                         { If we're replacing text, styles are kaput }
40 1523 1-             BEGIN
40 1524 --
40 1525 --             WITH fHTE^^ DO
40 1526 2-                 BEGIN
40 1527 --                 IF fSavedTEHandle <> theText THEN
40 1528 3-                     BEGIN
40 1529 --                     IF fSavedTEHandle <> NIL THEN       { If there's already a saved handle, then... }
40 1530 --                         DisposHandle(fSavedTEHandle);   { ...we have no choice but to dispose it      }
40 1531 --                     fSavedTEHandle := hText;            { Save existing handle                       }
40 1532 -3                     END;
40 1533 --                 hText := theText;                   { Install new handle                         }
40 1534 --                 fText := theText;                   { Make a local copy, too                     }
40 1535 --
40 1536 --             {$H-}
40 1537 --                 teLength := GetHandleSize(hText);   { Tell TE how long we are                    }
40 1538 --             {$H+}
40 1539 -2                 END;
40 1540 --
40 1541 --             IF fStyleType = kWithStyle THEN         { Fix for styled TE.  Yuk.                   }
40 1542 2-                 BEGIN
40 1543 --                 styles := GetStylHandle(fHTE);
40 1544 --                 styles^^.runs[1].startChar := SUCC(fHTE^^.teLength);
40 1545 -2                 END;
40 1546 -1             END;
40 1547 -0 A   END;
40 1548 --
40 1549 --     {$S TENonRes}
40 1550 --     {-----------------------------------+
40 1551 --     |   StuffTERects                    |
40 1552 --     +-----------------------------------}
40 1553 -- A   PROCEDURE TTEView.StuffTERects(newTERect: Rect);
40 1554 --     VAR
40 1555 --         aFontInfo:  FontInfo;
40 1556 --
40 1557 0- A   BEGIN
40 1558 --
40 1559 --         GetFontInfo(aFontInfo);
40 1560 --         WITH newTERect DO
40 1561 --             right := Max(right, left + aFontInfo.widMax);
40 1562 --         WITH fHTE^^ DO
40 1563 1-             BEGIN
40 1564 --             destRect := newTERect;
40 1565 --             viewRect := newTERect;
40 1566 -1             END;
40 1567 -0 A   END;
40 1568 --
40 1569 --
40 1570 --     {$S TERes}
40 1571 --     {-----------------------------------+
40 1572 --     |   SynchView                       |
40 1573 --     +-----------------------------------}
40 1574 -- A   PROCEDURE TTEView.SynchView (redraw: BOOLEAN);
40 1575 --     VAR
40 1576 --         aRect:      Rect;
40 1577 --         theHeight:  INTEGER;
40 1578 --
40 1579 0- A   BEGIN
40 1580 --         theHeight := CalcRealHeight;
40 1581 --         IF fLastHeight <> theHeight THEN
40 1582 1-             BEGIN
40 1583 --             AdjustSize;                             { may need to grow view                      }
```

```
40 1584 --              fLastHeight := theHeight;               { Remember new height value                 }
40 1585 -1              END;
40 1586 --
40 1587 --          IF redraw THEN
40 1588 --              { First, make sure selection is visible (this conveniently focuses).  Then,
40 1589 --                  repair any extra feedback which TextEdit may have mashed. }
40 1590 --              ScrollSelectionIntoView;
40 1591 -0 A    END;
40 1592 --
40 1593 --
40 1594 --      {$S TENonRes}
40 1595 --      {-----------------------------------+
40 1596 --      |   WriteToDeskScrap                |
40 1597 --      +-----------------------------------}
40 1598 -- A    PROCEDURE TTEView.WriteToDeskScrap; OVERRIDE;
40 1599 --      VAR
40 1600 --          aHandle:    Handle;
40 1601 --
40 1602 0- A    BEGIN
40 1603 --          FailOSErr(PutDeskScrapData('TEXT', fText));
40 1604 --
40 1605 --          IF (fStyleType = kWithStyle) & SpaceForStyles(0, MAXINT) THEN
40 1606 1-              BEGIN
40 1607 --              SetSelect(0, MAXINT, fHTE);
40 1608 --              aHandle := Handle(GetStylScrap(fHTE));
40 1609 --              FailNIL(aHandle);
40 1610 --              FailOSErr(PutDeskScrapData('styl', aHandle));
40 1611 -1              END;
40 1612 -0 A    END;
40 1613 --
40 1614 --
40 1615 --      {$IFC qDebug}
40 1616 --          {$S TEDebug}
40 1617 --          {-----------------------------------+
40 1618 --          |   IdentifySoftware                |
40 1619 --          +-----------------------------------}
40 1620 -- A        PROCEDURE TTEView.IdentifySoftware; OVERRIDE;
40 1621 0- A        BEGIN
40 1622 --              WRITELN('UTEView of 13Nov87, Compiled on ', COMPDATE, ' @ ', COMPTIME);
40 1623 --              INHERITED IdentifySoftware;
40 1624 -0 A        END;
40 1625 --      {$ENDC}
40 1626 --
40 1627 --
40 1628 --      {$IFC qDebug}
40 1629 --          {$S TEFields}
40 1630 --          {-----------------------------------+
40 1631 --          |   Fields                          |
40 1632 --          +-----------------------------------}
40 1633 -- A        PROCEDURE TTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
40 1634 --                                                        fieldAddr: Ptr;
40 1635 --                                                        fieldType: INTEGER)); OVERRIDE;
40 1636 --
40 1637 0- A        BEGIN
40 1638 --              DoToField('TTEView', NIL, bClass);
40 1639 --              DoToField('fHTE', @fHTE, bTEHandle);
40 1640 --              DoToField('fText', @fText, bHandle);
40 1641 --              DoToField('fSavedTEHandle', @fSavedTEHandle, bTEHandle);
40 1642 --              DoToField('fInset', @fInset, bRect);
40 1643 --              DoToField('fKeyCmdNumber', @fKeyCmdNumber, bCmdNumber);
40 1644 --              DoToField('fMaxChars', @fMaxChars, bInteger);
40 1645 --              DoToField('fLastHeight', @fLastHeight, bInteger);
40 1646 --              DoToField('fTypingCommand', @fTypingCommand, bObject);
40 1647 --              {$H-}
40 1648 --              TextStyleFields('fTextStyle', fTextStyle, DoToField);
40 1649 --              {$H+}
40 1650 --              DoToField('fJustification', @fJustification, bInteger);
40 1651 --              DoToField('fAcceptsChanges', @fAcceptsChanges, bBoolean);
40 1652 --              DoToField('fStyleType', @fStyleType, bBoolean);
40 1653 --              DoToField('fAutoWrap', @fAutoWrap, bBoolean);
40 1654 --              DoToField('fFreeText', @fFreeText, bBoolean);
40 1655 --              DoToField('fSpecsChanged', @fSpecsChanged, bBoolean);
40 1656 --              DoToField('fScroller', @fScroller, bObject);
40 1657 --              DoToField('fLastLine', @fLastLine, bInteger);
40 1658 --              DoToField('fLastPageBreak', @fLastPageBreak, bInteger);
40 1659 --              INHERITED Fields(DoToField);
40 1660 -0 A        END;
40 1661 --      {$ENDC}
38  625 --      {$I UTEView.TTECommand.p}
```

```
41    1 --      {$P}
41    2 --      {UTEView.TTECommand.p}
41    3 --      {Copyright © 1984-1988 Apple Computer Inc.  All rights reserved.}
41    4 --
41    5 --      {*****************************************************************************************}
41    6 --      {*      T T E C o m m a n d                                                              *}
41    7 --      {*****************************************************************************************}
41    8 --
41    9 --      {$S TESelCommand}
41   10 --      {---------------------------------+
41   11 --      |   ITECommand                    |
41   12 --      +---------------------------------}
41   13 -- A    PROCEDURE TTECommand.ITECommand(itsTEView: TTEView; itsCmdNumber: CmdNumber; itsSaveText: BOOLEAN);
41   14 --      VAR
41   15 --          selChars:   INTEGER;
41   16 --          h:          Handle;
41   17 --          fi:         FailInfo;
41   18 --
41   19 --          {-----------------------------+
41   20 --          |   HdlInitFailed             |
41   21 --          +-----------------------------}
41   22 -- B       PROCEDURE HdlInitFailed(error: OSErr; message: LONGINT);
41   23 0- B       BEGIN
41   24 --            Free;
41   25 -0 B       END;
41   26 --
41   27 0- A    BEGIN
41   28 --          fTEView := itsTEView;
41   29 --          fHTE := itsTEView.fHTE;
41   30 --
41   31 --          WITH fHTE^^ DO
41   32 1-              BEGIN
41   33 --              fOldStart := selStart;
41   34 --              fOldEnd := selEnd;
41   35 --              selChars := selEnd - selStart;
41   36 -1              END;
41   37 --
41   38 --          fOldText := NIL;
41   39 --          fOldStyles := NIL;
41   40 --
41   41 --          fNewStart := 0;
41   42 --          fNewEnd := 0;
41   43 --          fNewText := NIL;
41   44 --          fNewStyles := NIL;
41   45 --
41   46 --          fPadding := NIL;
41   47 --          fTextPad := 0;
41   48 --          fStylePad := 0;
41   49 --
41   50 --          ICommand(itsCmdNumber, itsTEView.fDocument, NIL, NIL);
41   51 --          CatchFailures(fi, HdlInitFailed);
41   52 --
41   53 --          IF itsSaveText THEN
41   54 1-              BEGIN
41   55 --              h := NewPermHandle(selChars);
41   56 --              FailNIL(h);
41   57 --
41   58 --              IF selChars > 0 THEN
41   59 --                  BlockMove(Pointer(ORD(fHTE^^.hText^) + fOldStart), h^, selChars);
41   60 --
41   61 --              fOldText := h;
41   62 --              fTextPad := fOldStart - fOldEnd;
41   63 --              fPadding := NewPermHandle(0);
41   64 --              FailNIL(fPadding);
41   65 -1              END;
41   66 --
41   67 --          { TextEdit has this "feature" which it exercises if it runs out of memory.  It's
41   68 --            called DS number 25.  We'll try to avoid it by assuring that enough memory exists
41   69 --            to fulfill the request, but we won't die because of it.  This is a particularly
41   70 --            ugly situation - there could be >600K of style information associated with a 32K
41   71 --            block of text.  And to support undo, we've got to assume that there may momentarily
41   72 --            be THREE copies floating around, adding up to a total potential liability of almost
41   73 --            2 Meg for a single TE record.  The worst that can happen, though, is that the text
41   74 --            will be safe, but it won't have any styles associated with it. }
41   75 --
41   76 --          IF (itsTEView.fStyleType = kWithStyle) &
41   77 --            itsTEView.SpaceForStyles(fHTE^^.selStart, fHTE^^.selEnd) THEN
41   78 1-              BEGIN
41   79 --              fOldStyles := GetStylScrap(fHTE);
41   80 --              FailNIL(fOldStyles);
41   81 --              fStylePad := GetHandleSize(Handle(fOldStyles));
41   82 -1              END;
41   83 --
41   84 --          Success(fi);
41   85 --
41   86 -0 A    END;
41   87 --
41   88 --
41   89 --      {$S TEDoCommand}
41   90 --      {---------------------------------+
41   91 --      |   Free                          |
41   92 --      +---------------------------------}
41   93 -- A    PROCEDURE TTECommand.Free; OVERRIDE;
41   94 0- A    BEGIN
41   95 --          DisposIfHandle(fOldText);
41   96 --          DisposIfHandle(Handle(fOldStyles));
41   97 --          DisposIfHandle(fNewText);
41   98 --          DisposIfHandle(Handle(fNewStyles));
```

```
41   99 --          DisposIfHandle(fPadding);
41  100 --
41  101 --          INHERITED Free;
41  102 -0 A    END;
41  103 --
41  104 --
41  105 --      {$S TEDoCommand}
41  106 --      {-----------------------------------+
41  107 --      |   BanishOldText                   |
41  108 --      +-----------------------------------}
41  109 -- A    PROCEDURE TTECommand.BanishOldText;
41  110 0- A    BEGIN
41  111 --          IF fOldEnd > fOldStart THEN
41  112 --              TEDelete(fHTE);
41  113 --          SetHandleSize(fPadding, MAX(-(fTextPad+fStylePad), 0));
41  114 -0 A    END;
41  115 --
41  116 --
41  117 --      {$S TEDoCommand}
41  118 --      {-----------------------------------+
41  119 --      |   InstallNewText                  |
41  120 --      +-----------------------------------}
41  121 -- A    PROCEDURE TTECommand.InstallNewText;
41  122 --      VAR
41  123 --          savedSize:  LONGINT;
41  124 --          itsText:    Handle;
41  125 --
41  126 0- A    BEGIN
41  127 --          IF fNewEnd > fNewStart THEN
41  128 1-              BEGIN
41  129 --              itsText := fTEView.fText;
41  130 --              savedSize := GetHandleSize(itsText);
41  131 --
41  132 --              {$IFC qDebug}
41  133 --              IF fNewText = NIL THEN
41  134 --                  ProgramBreak('InstallNewText called with fNewText = NIL!');
41  135 --              {$ENDC}
41  136 --
41  137 --              MoveHHi(fNewText);                      { Prevent heap fragmentation for TEInsert   }
41  138 --              HLock(fNewText);
41  139 --
41  140 --              IF fTEView.fStyleType = kWithStyle THEN { If record has style, use it              }
41  141 --                  TEStylInsert(fNewText^,             { (It's okay for fNewStyles to be NIL here) }
41  142 --                    GetHandleSize(fNewText), fNewStyles, fHTE)
41  143 --              ELSE                                    { Otherwise, do it the old-fashioned way    }
41  144 --                  TEInsert(fNewText^, GetHandleSize(fNewText), fHTE);
41  145 --
41  146 --              HUnlock(fNewText);
41  147 --
41  148 --              IF GetHandleSize(itsText) <= savedSize THEN
41  149 --                  FailOSErr(memFullErr);
41  150 --
41  151 --              fTEView.fSpecsChanged := TRUE;
41  152 -1              END;
41  153 -0 A    END;
41  154 --
41  155 --
41  156 --      {$IFC qDebug}
41  157 --          {$S TEFields}
41  158 --          {-----------------------------------+
41  159 --          |   Fields                          |
41  160 --          +-----------------------------------}
41  161 -- A        PROCEDURE TTECommand.Fields (PROCEDURE DoToField (fieldName: Str255;
41  162 --                                                           fieldAddr: Ptr;
41  163 --                                                           fieldType: INTEGER)); OVERRIDE;
41  164 0- A        BEGIN
41  165 --              DoToField('TTECommand', NIL, bClass);
41  166 --              DoToField('fTEView', @fTEView, bObject);
41  167 --              DoToField('fHTE', @fHTE, bTEHandle);
41  168 --              DoToField('fOldStart', @fOldStart, bInteger);
41  169 --              DoToField('fOldEnd', @fOldEnd, bInteger);
41  170 --              DoToField('fOldText', @fOldText, bHandle);
41  171 --              DoToField('fOldStyles', @fOldStyles, bHandle);
41  172 --              DoToField('fNewStart', @fNewStart, bInteger);
41  173 --              DoToField('fNewEnd', @fNewEnd, bInteger);
41  174 --              DoToField('fNewText', @fNewText, bHandle);
41  175 --              DoToField('fNewStyles', @fNewStyles, bHandle);
41  176 --              DoToField('fPadding', @fPadding, bHandle);
41  177 --              DoToField('fTextPad', @fTextPad, bInteger);
41  178 --              DoToField('fStylePad', @fStylePad, bLongInt);
41  179 --              INHERITED Fields(DoToField);
41  180 -0 A        END;
41  181 --      {$ENDC}
41  182 --
41  183 --
41  184 --      {$S TEDoCommand}
41  185 --      {-----------------------------------+
41  186 --      |   RemoveAdditions                 |   Used for UNDO phase
41  187 --      +-----------------------------------}
41  188 -- A    PROCEDURE TTECommand.RemoveAdditions;
41  189 0- A    BEGIN
41  190 --          IF fNewText <> NIL THEN
41  191 1-              BEGIN
41  192 --              TESetSelect(fNewStart, fNewEnd, fHTE);
41  193 --              TEDelete(fHTE);
41  194 -1              END;
41  195 --          SetHandleSize(fPadding, MAX(fTextPad+fStylePad, 0));
41  196 -0 A    END;
41  197 --
```

```
41  198 --
41  199 --      {$S TEDoCommand}
41  200 --      {-----------------------------------+
41  201 --      |   RestoreSelection                |
41  202 --      +-----------------------------------}
41  203 -- A    PROCEDURE TTECommand.RestoreSelection;
41  204 0- A    BEGIN
41  205 --          TESetSelect(fOldStart, fOldEnd, fHTE);
41  206 -0 A    END;
41  207 --
41  208 --
41  209 --      {$S TEDoCommand}
41  210 --      {-----------------------------------+
41  211 --      |   ReviveDeletions                 |   Used for UNDO phase
41  212 --      +-----------------------------------}
41  213 -- A    PROCEDURE TTECommand.ReviveDeletions;
41  214 --      VAR
41  215 --          itsText:        Handle;
41  216 --          savedSize:      LONGINT;
41  217 --          nChars:         INTEGER;
41  218 0- A    BEGIN
41  219 --          TESetSelect(fOldStart, fOldStart, fHTE);    { so insert will take place at right point  }
41  220 --          nChars := fOldEnd - fOldStart;
41  221 --          IF nChars > 0 THEN
41  222 1-              BEGIN
41  223 --              itsText := fTEView.fText;
41  224 --              savedSize := GetHandleSize(itsText);
41  225 --
41  226 --              MoveHHi(fOldText);                      { Prevent heap fragmentation                }
41  227 --              HLock(fOldText);
41  228 --
41  229 --              IF fTEView.fStyleType = kWithStyle THEN { If record has style, use it               }
41  230 --                  TEStylInsert(fOldText^, nChars,     { (It's okay for fOldStyles to be NIL here) }
41  231 --                    fOldStyles, fHTE)
41  232 --              ELSE                                    { Otherwise, do it the old-fashioned way    }
41  233 --                  TEInsert(fOldText^, nChars, fHTE);
41  234 --
41  235 --              HUnlock(fOldText);
41  236 --
41  237 --              IF GetHandleSize(itsText) <= savedSize THEN
41  238 --                  FailOSErr(memFullErr);
41  239 --
41  240 --              fTEView.fSpecsChanged := TRUE;
41  241 -1              END;
41  242 -0 A    END;
41  243 --
41  244 --
41  245 --      {$S TEDoCommand}
41  246 --      {-----------------------------------+
41  247 --      |   DoMainFunction                  |   Used for DO and REDO
41  248 --      +-----------------------------------}
41  249 -- A    PROCEDURE TTECommand.DoMainFunction;
41  250 0- A    BEGIN
41  251 --          IF fCmdNumber <> cCopy THEN
41  252 --              BanishOldText;
41  253 --          InstallNewText;
41  254 --          IF fCmdNumber <> cCopy THEN
41  255 --              fTEView.SynchView(kRedraw);
41  256 -0 A    END;
41  257 --
41  258 --
41  259 --      {$S TEDoCommand}
41  260 --      {-----------------------------------+
41  261 --      |   DoIt                            |
41  262 --      +-----------------------------------}
41  263 -- A    PROCEDURE TTECommand.DoIt; OVERRIDE;
41  264 0- A    BEGIN
41  265 --          IF fTEView.Focus THEN ;                 {??? What if Focus fails}
41  266 --
41  267 --          DoMainFunction;
41  268 --      {$IFC qDebug}
41  269 --          IF gTEIntenseDebugging THEN
41  270 --              DumpTTECommand(SELF);
41  271 --      {$ENDC}
41  272 -0 A    END;
41  273 --
41  274 --
41  275 --      {$S TEDoCommand}
41  276 --      {-----------------------------------+
41  277 --      |   UndoIt                          |
41  278 --      +-----------------------------------}
41  279 -- A    PROCEDURE TTECommand.UndoIt; OVERRIDE;
41  280 0- A    BEGIN
41  281 --          IF fTEView.Focus THEN ;                 {??? What if Focus fails}
41  282 --
41  283 --          RemoveAdditions;
41  284 --          ReviveDeletions;
41  285 --          RestoreSelection;
41  286 --          IF fCmdNumber <> cCopy THEN
41  287 --              fTEView.SynchView(kRedraw);
41  288 --      {$IFC qDebug}
41  289 --          IF gTEIntenseDebugging THEN
41  290 --              DumpTTECommand(SELF);
41  291 --      {$ENDC}
41  292 -0 A    END;
41  293 --
41  294 --
41  295 --      {$S TEDoCommand}
41  296 --      {-----------------------------------+
```

```
41  297 --      |   RedoIt                          |
41  298 --      +-----------------------------------}
41  299 -- A   PROCEDURE TTECommand.RedoIt; OVERRIDE;
41  300 0- A   BEGIN
41  301 --          IF fTEView.Focus THEN ;                     {??? What if Focus fails}
41  302 --
41  303 --          RestoreSelection;
41  304 --          DoMainFunction;
41  305 --      {$IFC qDebug}
41  306 --          IF gTEIntenseDebugging THEN
41  307 --              DumpTTECommand(SELF);
41  308 --      {$ENDC}
41  309 -0 A   END;
41  310 --
41  311 --
41  312 --      (*****************************************************************************************)
41  313 --      (*      T T E C u t C o p y C o m m a n d                                                *)
41  314 --      (*****************************************************************************************)
41  315 --
41  316 --      {$S TESelCommand}
41  317 --      {-----------------------------------+
41  318 --      |   ITECutCopyCommand                |
41  319 --      +-----------------------------------}
41  320 -- A   PROCEDURE TTECutCopyCommand.ITECutCopyCommand(itsTEView: TTEView; itsCmdNumber: CmdNumber);
41  321 0- A   BEGIN
41  322 --          fClipCreated := FALSE;
41  323 --          ITECommand(itsTEView, itsCmdNumber, TRUE);
41  324 --          fChangesClipboard := TRUE;
41  325 --          fCausesChange := itsCmdNumber <> cCopy;
41  326 -0 A   END;
41  327 --
41  328 --
41  329 --      {$S TEDoCommand}
41  330 --      {-----------------------------------+
41  331 --      |   Free                            |
41  332 --      +-----------------------------------}
41  333 -- A   PROCEDURE TTECutCopyCommand.Free; OVERRIDE;
41  334 0- A   BEGIN
41  335 --          IF fClipCreated THEN
41  336 --              fOldText := NIL;
41  337 --          INHERITED Free;
41  338 -0 A   END;
41  339 --
41  340 --
41  341 --      {$S TEDoCommand}
41  342 --      {-----------------------------------+
41  343 --      |   DoIt                            |
41  344 --      +-----------------------------------}
41  345 -- A   PROCEDURE TTECutCopyCommand.DoIt; OVERRIDE;
41  346 --      VAR
41  347 --          clipTEView: TTEView;
41  348 --          clipHere:   BOOLEAN;
41  349 --          fi:         FailInfo;
41  350 --          clipStyle:  TextStyle;
41  351 --          itsSize:    VPoint;
41  352 --          itsMargins: Rect;
41  353 --
41  354 --          {--------------------------------+
41  355 --          |   HdlClipFailed                |
41  356 --          +--------------------------------}
41  357 -- B       PROCEDURE HdlClipFailed(error: OSErr; message: LONGINT);
41  358 0- B       BEGIN
41  359 --              clipTEView.Free;
41  360 -0 B       END;
41  361 --
41  362 0- A   BEGIN {TTECutCopyCommand.DoIt}
41  363 --          IF fTEView.Focus THEN ;                     {??? What if Focus fails}
41  364 --
41  365 --          SetTextStyle(clipStyle, applFont, [],       { Initial style same as virgin TEView       }
41  366 --                       12, gRGBBlack);
41  367 --
41  368 --          SetVPt(itsSize, 100, 50);                   { An arbitrary initial size.                }
41  369 --          SetRect(itsMargins, 10, 8, 10, 0);          { No bottom margin.                         }
41  370 --
41  371 --          New(clipTEView);                            { Create a new view for the clipboard       }
41  372 --          FailNIL(clipTEView);
41  373 --          WITH fTEView DO
41  374 --              clipTEView.ITEView(NIL, NIL,            { Initialize view                           }
41  375 --                gZeroVPt, itsSize,
41  376 --                sizeSuperView, sizeVariable,
41  377 --                itsMargins, clipStyle,
41  378 --                teJustLeft, fStyleType, fAutoWrap);
41  379 --          clipTEView.fAcceptsChanges := FALSE;        { This is a read-only view                  }
41  380 --
41  381 --          CatchFailures(fi, HdlClipFailed);           { Cut can eat into temp memory so users can }
41  382 --                                                      { _rescue text from overweight documents    }
41  383 --          IF NOT fCausesChange THEN                   { If Copy-ing, assure there's enough room   }
41  384 --              FailSpaceIsLow;
41  385 --          Success(fi);
41  386 --          clipTEView.StuffText(fOldText);
41  387 --          FailSpaceIsLow;
41  388 --
41  389 --          {??? GOT TO FIGURE OUT SOME WAY TO PRE-FLIGHT THIS! ??????????????????????????????????? }
41  390 --          IF clipTEView.fStyleType = kWithStyle THEN  { If record has style                      }
41  391 --              SetStylScrap(0, MAXINT, fOldStyles,     { _then put in the styles                   }
41  392 --                  FALSE, clipTEView.fHTE);
41  393 --          FailSpaceIsLow;
41  394 --
41  395 --          clipTEView.fFreeText := TRUE;               { Let TEView know it has to free the text   }
```

```
41  396 --
41  397 --          gApplication.ClaimClipboard(clipTEView);    { Okay to claim (will call RecalcText!)     }
41  398 --
41  399 --          fClipCreated := TRUE;                       { We be done                                }
41  400 --          DoMainFunction;                             { Do the actual cut/copy                    }
41  401 --
41  402 --      {$IFC qDebug}
41  403 --          IF gTEIntenseDebugging THEN
41  404 1-              BEGIN
41  405 --              DumpTERecord(clipTEView.fHTE);
41  406 --              DumpTTECommand(SELF);
41  407 -1              END;
41  408 --      {$ENDC}
41  409 -0 A   END;
41  410 --
41  411 --
41  412 --      {$S TEDoCommand}
41  413 --      {-----------------------------------+
41  414 --      |   ReviveDeletions                 |
41  415 --      +-----------------------------------}
41  416 -- A   PROCEDURE TTECutCopyCommand.ReviveDeletions; OVERRIDE;
41  417 0- A   BEGIN
41  418 --          IF fCmdNumber = cCut THEN
41  419 --              INHERITED ReviveDeletions;              { Don't do it for COPY                      }
41  420 -0 A   END;
41  421 --
41  422 --
41  423 --      (*****************************************************************************************)
41  424 --      (*      T T E P a s t e C o m m a n d                                                    *)
41  425 --      (*****************************************************************************************)
41  426 --
41  427 --      {$S TESelCommand}
41  428 --      {-----------------------------------+
41  429 --      |   ITEPasteCommand                 |
41  430 --      +-----------------------------------}
41  431 -- A   PROCEDURE TTEPasteCommand.ITEPasteCommand(itsTEView: TTEView);
41  432 --      { We can't use TEPaste because it clobbers the DeskScrap; the text would be recoverable
41  433 --        from the special TextEdit Scrap, but other types of non-TEXT scrap are permanently
41  434 --        lost, it seems }
41  435 --      VAR
41  436 --          savedPerm:      BOOLEAN;
41  437 --          newLength:      INTEGER;
41  438 --          newStyleLen:    LONGINT;
41  439 --          newText:        Handle;
41  440 --          newStyles:      Handle;
41  441 --          dataType:       ResType;
41  442 --          fi:             FailInfo;
41  443 --
41  444 --          {-------------------------------+
41  445 --          |   HdlPasteFailed              |
41  446 --          +-------------------------------}
41  447 -- B       PROCEDURE HdlPasteFailed(error: OSErr; message: LONGINT);
41  448 0- B       BEGIN
41  449 --              savedPerm := PermAllocation(savedPerm);
41  450 --              DisposIfHandle(newText);
41  451 --              DisposIfHandle(Handle(newStyles));
41  452 --              Free;
41  453 -0 B       END;
41  454 --
41  455 0- A   BEGIN
41  456 --          ITECommand(itsTEView, cPaste, TRUE);        { Perform stock initializations             }
41  457 --
41  458 --          savedPerm := FALSE;
41  459 --
41  460 --          newStyleLen := 0;                           { Assume there are no new styles            }
41  461 --          newStyles := NIL;
41  462 --          newText := NIL;
41  463 --
41  464 --          newText := NewPermHandle(0);                { Create a handle to receive clipboard data }
41  465 --          FailNIL(newText);
41  466 --          IF itsTEView.fStyleType = kWithStyle THEN
41  467 1-              BEGIN
41  468 --              newStyles := NewPermHandle(0);          { Same for handle to receive style info     }
41  469 --              FailNIL(Handle(newStyles));
41  470 -1              END;
41  471 --
41  472 --          CatchFailures(fi, HdlPasteFailed);
41  473 --
41  474 --          newLength := gApplication.GetDataToPaste(newText, dataType);
41  475 --
41  476 --          IF newLength > 0 THEN
41  477 1-              BEGIN
41  478 --      {$IFC qDebug}
41  479 --              IF dataType <> 'TEXT' THEN
41  480 --                  ProgramBreak('TEPasteCommand given some non-text from clipboard')
41  481 --              ELSE
41  482 --      {$ENDC}
41  483 2-                  BEGIN                               { Prime "new" values                        }
41  484 --                  fNewText := newText;
41  485 --                  fNewStart := fHTE^^.selStart;
41  486 --                  fNewEnd := fNewStart + newLength;
41  487 --                  fTextPad := newLength - (fOldEnd - fOldStart);
41  488 --
41  489 --                  IF itsTEView.fStyleType = kWithStyle THEN
41  490 3-                      BEGIN
41  491 --                      newStyleLen := gClipView.GivePasteData(newStyles, 'styl');
41  492 --                      IF newStyleLen > 0 THEN
41  493 4-                          BEGIN
41  494 --                          fNewStyles := StScrpHandle(newStyles);
```

```
41  495 --                          fStylePad :=                { Difference between old and new styles     }
41  496 --                            newStyleLen - fStylePad;
41  497 -4                         END;
41  498 -3                     END;
41  499 --
41  500 --                 savedPerm := PermAllocation(TRUE);  { fPadding remains a permanent handle_      }
41  501 --                 SetHandleSize(fPadding, MAX(fTextPad+fStylePad, 0));
41  502 --                 FailMemError;
41  503 --
41  504 --                 FailSpaceIsLow;
41  505 -2                 END;
41  506 -1             END;
41  507 --         Success(fi);
41  508 --         savedPerm := PermAllocation(savedPerm);     { Allocation state back to the way it was   }
41  509 -0 A    END;
41  510 --
41  511 --     (*****************************************************************************************)
41  512 --     (*      T T E S t y l e C o m m a n d                                                    *)
41  513 --     (*****************************************************************************************)
41  514 --
41  515 --     {$S TESelCommand}
41  516 --     {-----------------------------------+
41  517 --     |   ITEStyleCommand                 |
41  518 --     +-----------------------------------}
41  519 -- A   PROCEDURE TTEStyleCommand.ITEStyleCommand(itsTEView: TTEView; itsNewStyle: TextStyle;
41  520 --                         itsCmdNumber: CmdNumber; itsMode: INTEGER);
41  521 --     VAR
41  522 --         savedPerm:      BOOLEAN;
41  523 --         fi:             FailInfo;
41  524 --
41  525 0- A   BEGIN
41  526 --
41  527 --         ITECommand(itsTEView, itsCmdNumber, FALSE); { Perform stock initialization, sans text   }
41  528 --
41  529 --         fOldTextStyle := itsTEView.fTextStyle;
41  530 --         fNewTextStyle := itsNewStyle;
41  531 --
41  532 --         fMode := itsMode;
41  533 --         IF NOT gConfiguration.hasColorQD THEN
41  534 --             fMode := BAND(fMode, BNOT(doColor));    { Only do color change if we can        }
41  535 -0 A   END;
41  536 --
41  537 --     {$S TEDoCommand}
41  538 --     {-----------------------------------+
41  539 --     |   InstallOneStyle                 |
41  540 --     +-----------------------------------}
41  541 -- A   PROCEDURE TTEStyleCommand.InstallOneStyle(newStyl: TextStyle);
41  542 --
41  543 0- A   BEGIN
41  544 --         fTEView.SetOneStyle(fOldStart, fOldEnd, fMode, newStyl, kRedraw);   { Focus'es for us          }
41  545 -0 A   END;
41  546 --
41  547 --     {$S TEDoCommand}
41  548 --     {-----------------------------------+
41  549 --     |   InstallManyStyles               |
41  550 --     +-----------------------------------}
41  551 -- A   PROCEDURE TTEStyleCommand.InstallManyStyles(newStyls: StScrpHandle);
41  552 --
41  553 0- A   BEGIN
41  554 --         IF fTEView.Focus THEN ;
41  555 --         { No need to check for fStyleType, since we only get here if the record is stylish      }
41  556 --         SetStylScrap(fOldStart, fOldEnd, newStyls, TRUE, fHTE);
41  557 --         fTEView.RecalcText;                             { Might have changed number of lines    }
41  558 --         fTEView.SynchView(kRedraw);                     { Show corrected view                   }
41  559 --
41  560 --         fTEView.fSpecsChanged := TRUE;
41  561 -0 A   END;
41  562 --
41  563 --     {$S TEDoCommand}
41  564 --     {-----------------------------------+
41  565 --     |   DoIt                            |
41  566 --     +-----------------------------------}
41  567 -- A   PROCEDURE TTEStyleCommand.DoIt; OVERRIDE;
41  568 --
41  569 0- A   BEGIN
41  570 --         InstallOneStyle(fNewTextStyle);
41  571 --         fMode := BAND(fMode, BNOT(doToggle));           { Turn off toggle mode, if set          }
41  572 --     {$IFC qDebug}
41  573 --         IF gTEIntenseDebugging THEN
41  574 --             DumpTTECommand(SELF);
41  575 --     {$ENDC}
41  576 -0 A   END;
41  577 --
41  578 --     {$S TEDoCommand}
41  579 --     {-----------------------------------+
41  580 --     |   UndoIt                          |
41  581 --     +-----------------------------------}
41  582 -- A   PROCEDURE TTEStyleCommand.UndoIt; OVERRIDE;
41  583 --
41  584 0- A   BEGIN
41  585 --         IF fTEView.fStyleType = kWithoutStyle THEN
41  586 --             InstallOneStyle(fOldTextStyle)
41  587 --         ELSE
41  588 --             InstallManyStyles(fOldStyles);
41  589 --     {$IFC qDebug}
41  590 --         IF gTEIntenseDebugging THEN
41  591 --             DumpTTECommand(SELF);
41  592 --     {$ENDC}
41  593 -0 A   END;
```

```
41  594 --
41  595 --      {$S TEDoCommand}
41  596 --      {-----------------------------------+
41  597 --      |   RedoIt                          |
41  598 --      +-----------------------------------}
41  599 -- A    PROCEDURE TTEStyleCommand.RedoIt; OVERRIDE;
41  600 --
41  601 0- A    BEGIN
41  602 --          DoIt;
41  603 -0 A    END;
41  604 --
41  605 --      {$IFC qDebug}
41  606 --          {$S TEFields}
41  607 --          {-----------------------------------+
41  608 --          |   Fields                          |
41  609 --          +-----------------------------------}
41  610 -- A      PROCEDURE TTEStyleCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
41  611 --                                           fieldAddr: Ptr;
41  612 --                                           fieldType: INTEGER)); OVERRIDE;
41  613 0- A      BEGIN
41  614 --          DoToField('TTEStyleCommand', NIL, bClass);
41  615 --          DoToField('fMode', @fMode, bInteger);
41  616 --          {$H-}
41  617 --          TextStyleFields('fOldTextStyle', fOldTextStyle, DoToField);
41  618 --          TextStyleFields('fNewTextStyle', fNewTextStyle, DoToField);
41  619 --          {$H+}
41  620 --          INHERITED Fields(DoToField);
41  621 -0 A      END;
41  622 --      {$ENDC}
41  623 --
41  624 --      (*****************************************************************************************)
41  625 --      (*      T T E T y p i n g C o m m a n d                                                   *)
41  626 --      (*****************************************************************************************)
41  627 --
41  628 --      {$S TERes}
41  629 --      {-----------------------------------+
41  630 --      |   ITETypingCommand                |
41  631 --      +-----------------------------------}
41  632 -- A    PROCEDURE TTETypingCommand.ITETypingCommand(itsTEView: TTEView; itsFirstChar: Char);
41  633 --      VAR
41  634 --          fi:         FailInfo;
41  635 --
41  636 --          {-------------------------------+
41  637 --          |   HdlInitFailed               |
41  638 --          +-------------------------------}
41  639 -- B      PROCEDURE HdlInitFailed(error: OSErr; message: LONGINT);
41  640 0- B      BEGIN
41  641 --          Free;
41  642 -0 B      END;
41  643 --
41  644 0- A    BEGIN
41  645 --          ITECommand(itsTEView, cTyping, TRUE);
41  646 --
41  647 --          CatchFailures(fi, HdlInitFailed);
41  648 --
41  649 --          fNewStart := fHTE^^.selStart;               { Start and end are the same              }
41  650 --          fNewEnd := fNewStart;
41  651 --
41  652 --          fNewText := NewPermHandle(0);               { Allocate an empty block for text        }
41  653 --          FailNIL(fNewText);
41  654 --
41  655 --          fCompleted := FALSE;                        { We've only just begun…                  }
41  656 --          fFirstChar := itsFirstChar;                 { Save character for Doit                 }
41  657 --          Success(fi);
41  658 -0 A    END;
41  659 --
41  660 --
41  661 --      {$S TERes}
41  662 --      {-----------------------------------+
41  663 --      |   Free                            |
41  664 --      +-----------------------------------}
41  665 -- A    PROCEDURE TTETypingCommand.Free; OVERRIDE;
41  666 0- A    BEGIN
41  667 --          IF fTEView.fTypingCommand = SELF THEN
41  668 --              fTEView.fTypingCommand := NIL;
41  669 --          INHERITED Free;
41  670 -0 A    END;
41  671 --
41  672 --
41  673 --      {$S TERes}
41  674 --      {-----------------------------------+
41  675 --      |   DoNormalChar                    |   Handle any typed character except a backspace
41  676 --      +-----------------------------------}
41  677 -- A    PROCEDURE TTETypingCommand.DoNormalChar(aChar: CHAR);
41  678 0- A    BEGIN
41  679 --          FailOSErr(PtrAndHand(Ptr(SUCC(ORD(@aChar))),    { Append char to end of fNewText        }
41  680 --            fNewText, 1));
41  681 --          fNewEnd := SUCC(fNewEnd);                   { Bump both end of "selection"            }
41  682 --          fTextPad := SUCC(fTextPad);                 { …and padding value                      }
41  683 --
41  684 --          SetHandleSize(fPadding,                     { This SetHandleSize can't grow the handle, }
41  685 --            MAX(-(fTextPad+fStylePad), 0));           { …so it shouldn't fail.                  }
41  686 -0 A    END;
41  687 --
41  688 --      {$S TERes}
41  689 --      {-----------------------------------+
41  690 --      |   BkSpcLeft                       |   Handle backspace to the left of original selection
41  691 --      +-----------------------------------}
41  692 --      { User has backspaced to the left of the original starting point.  First, copy the
```

```
41  693 --          character (which may be more than one byte long if we are using a non-Roman script)
41  694 --          to a temporary buffer.  The assumption is that no character will ever be longer
41  695 --          than four bytes.  Sorry, folks, MacApp does not support typing in any script with
41  696 --          more than 4 billion characters.
41  697 --        Next, copy the character to the front of fOldText, and adjust fOldStart, fNewStart,
41  698 --          and fNewEnd.  Note that we do NOT check for MemSpaceIsLow, since we want to let the
41  699 --          user delete characters. }
41  700 --
41  701 -- A    PROCEDURE TTETypingCommand.BkSpcLeft(theText: Handle; curStart: INTEGER);
41  702 --      TYPE
41  703 --          TSPtr = ^TextStyle;
41  704 --      VAR
41  705 --          savedSize:      INTEGER;
41  706 --          theHeight:      INTEGER;
41  707 --          theAscent:      INTEGER;
41  708 --          oldSize:        LONGINT;
41  709 --          whoCares:       LONGINT;
41  710 --          aTextStyle:     TSPtr;
41  711 --          savedChar:      PACKED ARRAY [0..3] OF CHAR;
41  712 --          delStyle:       TextStyle;
41  713 --      {$IFC qDebug}
41  714 --          savedPerm:      BOOLEAN;
41  715 --      {$ENDC}
41  716 --
41  717 0- A    BEGIN
41  718 --          savedSize := 1;
41  719 --          IF gConfiguration.hasScriptManager THEN
41  720 --              WHILE CharByte(theText^, curStart-savedSize) > 0 DO
41  721 --                  savedSize := SUCC(savedSize);
41  722 --          curStart := curStart-savedSize;
41  723 --
41  724 --      {$IFC qDebug}
41  725 --          IF savedSize > 4 THEN
41  726 --              ProgramBreak('TTETypingCommand.AddCharacter: Character > 4 bytes');
41  727 --      {$ENDC}
41  728 --          IF savedSize = 1 THEN                       { Slight speed optimization for normal case }
41  729 --      {$IFC qRangeCheck}{$R-}{$ENDC}
41  730 --              savedChar[0] := CharsHandle(theText)^^[curStart]
41  731 --      {$IFC qRangeCheck}{$R+}{$ENDC}
41  732 --          ELSE
41  733 --              BlockMove(Ptr(ORD(theText^) + curStart), @savedChar, savedSize);
41  734 --
41  735 --          IF fTEView.fStyleType = kWithStyle THEN     { Only do this if styles are around          }
41  736 1-             BEGIN
41  737 --              TEGetStyle(curStart, delStyle,          { Get the style of the deleted character    }
41  738 --                theHeight, theAscent, fHTE);          { (1 or 4 bytes, it's all only one style)   }
41  739 --
41  740 --              IF NOT EqualBlocks(@delStyle,           { If style doesn't match first in the list  }
41  741 --                @fOldStyles^^.scrpStyleTab[0].scrpFont,
41  742 --                SIZEOF(TextStyle)) THEN
41  743 2-                BEGIN                                 { _then insert new style at head of list    }
41  744 --                fTEView.fSpecsChanged := TRUE;        { User backspaced into new style!           }
41  745 --
41  746 --                oldSize :=                            { Make room for the new style element       }
41  747 --                  GetHandleSize(Handle(fOldStyles));
41  748 --                SetHandleSize(Handle(fOldStyles),
41  749 --                  oldSize + SIZEOF(ScrpSTElement));
41  750 --                FailMemError;
41  751 --                fStylePad := fStylePad + SIZEOF(ScrpSTElement);
41  752 --
41  753 --              {$H-}
41  754 --                WITH fOldStyles^^.scrpStyleTab[0] DO
41  755 --                    BlockMove(@scrpStartChar,         { Move entire array up one element's size   }
41  756 --                      Ptr(ORD(@scrpStartChar)+SIZEOF(ScrpSTElement)),
41  757 --                      oldSize - SIZEOF(fOldStyles^^.scrpNStyles));
41  758 --              {$H+}
41  759 --
41  760 --                  fOldStyles^^.scrpNStyles :=       { One more style                            }
41  761 --                    SUCC(fOldStyles^^.scrpNStyles);
41  762 --                WITH fOldStyles^^.scrpStyleTab[0] DO
41  763 3-                    BEGIN
41  764 --                    scrpHeight := theHeight;        { Fill in the blanks                        }
41  765 --                    scrpAscent := theAscent;
41  766 --                    aTextStyle := TSPtr(@scrpFont);
41  767 --                    aTextStyle^ := delStyle;
41  768 -3                    END;
41  769 -2                END;
41  770 --
41  771 --              WITH fOldStyles^^.scrpStyleTab[0] DO
41  772 --                  scrpStartChar := PRED(scrpStartChar);   { Regardless, back off offset by one   }
41  773 -1              END;
41  774 --
41  775 --          SetHandleSize(fPadding, GetHandleSize(fOldText) + savedSize + fStylePad);
41  776 --          FailMemError;
41  777 --          whoCares := Munger(fOldText, 0, NIL, 0, @savedChar, savedSize);
41  778 --          fOldStart := curStart;                      { Treat this as though original selection   }
41  779 --          fNewStart := curStart;                      { _had included this character              }
41  780 --          fNewEnd := curStart;
41  781 --          fTextPad := fOldStart - fOldEnd;
41  782 -0 A    END;
41  783 --
41  784 --      {$S TERes}
41  785 --      {-----------------------------------+
41  786 --      |   BkSpcRight                      |   Handle backspace to the right of original selection
41  787 --      +-----------------------------------}
41  788 -- A    PROCEDURE TTETypingCommand.BkSpcRight(theText: Handle; curStart: INTEGER);
41  789 --      VAR
41  790 --          savedSize:      INTEGER;
41  791 0- A    BEGIN
```

```
41  792 --          savedSize := 1;
41  793 --          IF gConfiguration.hasScriptManager THEN
41  794 --              WHILE CharByte(theText^, curStart - savedSize) > 0 DO
41  795 --                  savedSize := SUCC(savedSize);
41  796 --          SetHandleSize(fPadding, MAX(-(fTextPad - savedSize + fStylePad), 0));
41  797 --          FailMemError;
41  798 --          fNewEnd := fNewEnd - savedSize;
41  799 --          fTextPad := fTextPad - savedSize;
41  800 --
41  801 --          SetHandleSize(fNewText, fNewEnd-fNewStart); { Shouldn't fail as we're only shrinking it }
41  802 -0 A    END;
41  803 --
41  804 --
41  805 --      {$S TERes}
41  806 --      {-----------------------------------+
41  807 --      |   AddCharacter                    |
41  808 --      +-----------------------------------}
41  809 --      { ??? All this handle munging is expensive.  Better would be to accumulate memory in
41  810 --        "chunks" of, say, 16 bytes so that this checking need not happen every time through.
41  811 --        Fortunately, the normal cases are not that bad. }
41  812 --
41  813 -- A    PROCEDURE TTETypingCommand.AddCharacter(aChar: CHAR);
41  814 --      VAR
41  815 --          theText:        Handle;
41  816 --          curSelStart:    INTEGER;
41  817 --          curSelEnd:      INTEGER;
41  818 --          savedPerm:      BOOLEAN;
41  819 --          fi:             FailInfo;
41  820 --
41  821 --          {-------------------------------+
41  822 --          |   HdlCharFailed               |
41  823 --          +-------------------------------}
41  824 -- B        PROCEDURE HdlCharFailed(error: OSErr; message: LONGINT);
41  825 0- B       BEGIN
41  826 --              savedPerm := PermAllocation(savedPerm);
41  827 -0 B       END;
41  828 --
41  829 0- A    BEGIN {TTETypingCommand.AddCharacter}
41  830 --          WITH fHTE^^ DO                              { Get handy info about the text handle      }
41  831 1-              BEGIN
41  832 --              curSelStart := selStart;
41  833 --              curSelEnd := selEnd;
41  834 --              theText := hText;
41  835 -1              END;
41  836 --          CatchFailures(fi, HdlCharFailed);
41  837 --          savedPerm := PermAllocation(TRUE);
41  838 --
41  839 --          { Update the fNewText handle and other information.  Note that because of backspace,
41  840 --              this can be tricky.}
41  841 --
41  842 --          IF aChar <> chBackspace THEN                { Not a backspace.  Do the right thing      }
41  843 --              DoNormalChar(aChar)
41  844 --
41  845 --          ELSE IF (curSelStart <= fOldStart) AND      { User typed backspace so keep in synch     }
41  846 --                  (curSelStart > 0) AND (curSelStart = curSelEnd) THEN
41  847 --              BkSpcLeft(theText, curSelStart)         { Handle backspace to left of start         }
41  848 --
41  849 --          ELSE IF fNewEnd > fNewStart THEN            { Delete 1 character from end of fNewText   }
41  850 --              BkSpcRight(theText, curSelStart);       { Handle backspace to right of start        }
41  851 --
41  852 --          savedPerm := PermAllocation(savedPerm);
41  853 --          Success(fi);
41  854 --
41  855 --          { Let TextEdit have the character, as either 1) we're adding a byte, so we know there
41  856 --              is a reserve tank, so the worst this will do is eat into it a little, or 2) we're
41  857 --              deleting a character, which can only decrease memory usage. }
41  858 --          TEKey(aChar, fHTE);
41  859 --
41  860 --          fTEView.SynchView(kRedraw);                 { Now clean up the view.                    }
41  861 --
41  862 --      {$IFC qDebug}
41  863 --          IF gTEIntenseDebugging THEN
41  864 1-              BEGIN
41  865 --              WRITELN('fOldText:');
41  866 --              WWAddText(fOldText^, GetHandleSize(fOldText)); WRITELN;
41  867 --              WRITELN('fNewText:');
41  868 --              WWAddText(fNewText^, GetHandleSize(fNewText)); WRITELN;
41  869 --              DumpTTECommand(SELF);
41  870 -1              END;
41  871 --      {$ENDC}
41  872 --
41  873 -0 A    END;
41  874 --
41  875 --
41  876 --      {$S TERes}
41  877 --      {-----------------------------------+
41  878 --      |   DoIt                            |
41  879 --      +-----------------------------------}
41  880 -- A    PROCEDURE TTETypingCommand.DoIt; OVERRIDE;
41  881 0- A    BEGIN
41  882 --          AddCharacter(fFirstChar);
41  883 --      {$IFC qDebug}
41  884 --          IF gTEIntenseDebugging THEN
41  885 --              DumpTTECommand(SELF);
41  886 --      {$ENDC}
41  887 -0 A    END;
41  888 --
41  889 --
41  890 --      {$S TEDoCommand}
```

```
41  891 --      {-----------------------------------+
41  892 --      |   UndoIt                          |
41  893 --      +-----------------------------------}
41  894 -- A    PROCEDURE TTETypingCommand.UndoIt; OVERRIDE;
41  895 0- A    BEGIN
41  896 --          CompleteTyping;
41  897 --          INHERITED UndoIt;
41  898 --      {$IFC qDebug}
41  899 --          IF gTEIntenseDebugging THEN
41  900 --              DumpTTECommand(SELF);
41  901 --      {$ENDC}
41  902 -0 A    END;
41  903 --
41  904 --
41  905 --      {$S TERes}
41  906 --      {-----------------------------------+
41  907 --      |   CompleteTyping                  |   Mark as no more typing allowed & fix up styles
41  908 --      +-----------------------------------}
41  909 -- A    PROCEDURE TTETypingCommand.CompleteTyping;
41  910 --      VAR
41  911 --          i:          INTEGER;
41  912 --          offset:     LONGINT;
41  913 0- A    BEGIN
41  914 --          fCompleted := TRUE;
41  915 --
41  916 --          IF fTEView.fStyleType = kWithStyle THEN
41  917 --              WITH fOldStyles^^ DO
41  918 1-                  BEGIN
41  919 --                  offset := -scrpStyleTab[0].scrpStartChar;
41  920 --                  IF offset > 0 THEN
41  921 --                      FOR i := 0 TO scrpNStyles - 1 DO
41  922 --                          scrpStyleTab[i].scrpStartChar :=
41  923 --                            scrpStyleTab[i].scrpStartChar + offset;
41  924 -1                  END;
41  925 --      {$IFC qDebug}
41  926 --          IF gTEIntenseDebugging THEN
41  927 --              DumpTTECommand(SELF);
41  928 --      {$ENDC}
41  929 -0 A    END;
41  930 --
41  931 --
41  932 --      {$IFC qDebug}
41  933 --          {$S TEFields}
41  934 --          {-----------------------------------+
41  935 --          |   Fields                          |
41  936 --          +-----------------------------------}
41  937 -- A        PROCEDURE TTETypingCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
41  938 --                                            fieldAddr: Ptr;
41  939 --                                            fieldType: INTEGER)); OVERRIDE;
41  940 0- A        BEGIN
41  941 --              DoToField('TTETypingCommand', NIL, bClass);
41  942 --              DoToField('fCompleted', @fCompleted, bBoolean);
41  943 --              DoToField('fFirstChar', @fFirstChar, bBoolean);
41  944 --              INHERITED Fields(DoToField);
41  945 -0 A        END;
41  946 --      {$ENDC}
38  626 --
38  627 --      END.
```

```
42    1 --    {$P}
42    2 --    { UViewCoords.p }
42    3 --    { Copyright © 1987-1988 by Apple Computer, Inc.  All rights reserved. }
42    4 --
42    5 --    UNIT UViewCoords;
42    6 --
42    7 --    INTERFACE
42    8 --
42    9 --    {-------------------------------------+
42   10 --    |   Uses                              |
42   11 --    +-------------------------------------}
42   12 --    USES
42   13 --        {$LOAD MacIntf.LOAD}
42   14 --            MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
42   15 --        {$LOAD}
42   16 --            UMAUtil;
42   17 --
42   18 --    {-------------------------------------+
42   19 --    |   Global constants                  |
42   20 --    +-------------------------------------}
42   21 --    CONST
42   22 --        kMaxCoord           =       30000;  { largest possible QuickDraw view coordinate
42   23 --                                              (QuickDraw maximum minus slop
42   24 --                                              for size of screen) }
42   25 --
42   26 --    {-------------------------------------+
42   27 --    |   Global routines                   |
42   28 --    +-------------------------------------}
42   29 --    { Calculations with VPoints }
42   30 --
42   31 --    PROCEDURE PtToVPt (thePt: Point; VAR theVPt: VPoint);
42   32 --
42   33 --    FUNCTION VPtToPt (theVPt: VPoint): Point;
42   34 --
42   35 --    PROCEDURE AddVPt (srcVPt: VPoint; VAR dstVPt: VPoint);
42   36 --
42   37 --    PROCEDURE SubVPt (srcVPt: VPoint; VAR dstVPt: VPoint);
42   38 --
42   39 --    PROCEDURE SetVPt (VAR vPt: VPoint; h, v: VCoordinate);
42   40 --
42   41 --    FUNCTION EqualVPt (pt1, pt2: VPoint): BOOLEAN;
42   42 --
42   43 --
42   44 --    { Calculations with VRects }
42   45 --
42   46 --    PROCEDURE RectToVRect (theRect: Rect; VAR theVRect: VRect);
42   47 --
42   48 --    PROCEDURE VRectToRect (theVRect: VRect; VAR theRect: Rect);
42   49 --
42   50 --    PROCEDURE SetVRect (VAR r: VRect; left, top, right, bottom: VCoordinate);
42   51 --
42   52 --    PROCEDURE OffsetVRect (VAR r: VRect; dh, dv: VCoordinate);
42   53 --
42   54 --    PROCEDURE InsetVRect (VAR r: VRect; dh, dv: VCoordinate);
42   55 --
42   56 --    FUNCTION  PtInVRect (pt: VPoint; r: VRect): BOOLEAN;
42   57 --
42   58 --    FUNCTION  SectVRect (src1, src2: VRect; VAR dstRect: VRect): BOOLEAN;
42   59 --
42   60 --    FUNCTION  EmptyVRect (r: VRect): BOOLEAN;
42   61 --
42   62 --    FUNCTION  EqualVRect (rectA, rectB: VRect): BOOLEAN;
42   63 --
42   64 --    FUNCTION  LengthVRect (r: VRect; vhs: VHSelect): VCoordinate;
42   65 --
42   66 --    PROCEDURE PinVRect (r: VRect; VAR pt: VPoint);
42   67 --
42   68 --    PROCEDURE UnionVRect (src1, src2: VRect; VAR dstRect: VRect);
42   69 --
42   70 --    PROCEDURE Pt2VRect (topLeft, botRight: VPoint; VAR dstRect: VRect);
42   71 --
42   72 --    IMPLEMENTATION
42   73 --
42   74 --    {$I UViewCoords.incl.p}
42   75 --
42   76 --    END { UViewCoords }.
```

```
                        175 (32)  192*(32)  197=(32)   40*(39)   41 (39)
  XFrameOval            404*(18)  488 (18)
  XFrameRRect           412*(18)  491 (18)
  XLBusyDelay            51*( 4)  209 ( 4)
  XLBusyImage            52*( 4)  128 ( 4)
  XLCursorInit           53*( 4)   79 ( 4)  177 ( 4)
  xLoc                  354*(37)  371=(37)  372 (37)
  XOrRgn               1592 (18)

-Y-
  y                     278*(25)
  YellowBit              72*( 6)  100 ( 6)  293*(26)  321 (26)
  yellowColor           134 ( 6)  805 (26)

-Z-
  ZeroScrap             225 (16)  394 (16)
  Zoom                 1969*(13) 1632 (16) 1002*(19)
  zoomDocProc          1270 (16)
  ZoomWindow           1011 (19)
  ZPAOC8                 37 (32)   38*(32)
  ZPPAOC8                37*(32)   59 (32)

*** End PasRef: 4200 id's   41540 references
```