Mesa

ProgrammerÕs Guide

Release 1.5

Athena Design, Inc.
17 Saint MaryÕs Court
Boston, MA 02146

Copyright

Copyright ' 1992 & 1993	Athena Design, Inc. 17 Saint MaryÕs Court Boston, MA 02146

All rights reserved. First edition printed 1993.

Written by: Derek Fromhein, Greg Hudson, Linda Julien, and David Pollak
Edited by: Greg Hudson, Linda Julien and David Pollak

1. Introduction

There are many ways to customize Mesa to your specific needs, from writing a simple macro or script, to seamlessly adding your own functions, to creating a custom application that makes use of MesaÕs functionality.

AddIns

FunctionAddIns

Mesa provides many built-in functions that can be used in formulas, but you may need other functions that are specific to your use of a spreadsheet. Examples of this include bond pricing functions, scientific and statistical functions that would not be used by most spreadsheet users, or some proprietary calculations. AddIns provide the method of adding these functions to Mesa. Users enter AddIn functions into formulas the same way they enter built-in functions.

EventAddIns

AddIns add more functions that can be used in formulas. EventAddIns process user events and generate events. An event is something that changes the worksheet. For example, if a user selects a range, the EventAddIn will receive an event. If market data changes, the EventAddIn will generate an event that will change the corresponding value in the worksheet. EventAddIns can place items in the AddIn submenu. EventAddIns allow you to seamlessly extend MesaÕs functionality.

AdaptorAddIns

AdaptorAddIns allow you to write database adaptors to connect Mesa to any SQL database system. An AdaptorAddIn must implement a set of methods that glue Mesa to the SQL database systemÕs client library. The structure of the AdaptorAddIn is similar to the Sybase client library.

MOLI

The Mesa Object Library Interface allows other programs to access Mesa worksheets. Custom programs can contain live, editable spreadsheet views and graphs, insert values into a worksheet, get values from worksheets, and perform many other functions. If you are writing a custom application that contains a graph or has need for a spreadsheet-like data entry screen, you can use MOLI to easily include these features. MOLI also allows you to exchange values with a Mesa worksheet. MOLI can wait for an event to take place in a worksheet (e.g., the result of a calculation exceeding a pre-set level), and cause the custom program to do something (e.g., alert the user or execute a trade.)

2. AddIns

AddIns allow you to extend the functionality of Mesa using Objective-C. There are three kinds of AddIns: FunctionAddIns, which allow you to add new functions for formulas and MScript scripts, EventAddIns, which allow you to ÒlistenÓ to events in a Mesa worksheet and take action when they occur, and AdaptorAddins, which allow you to write SQL database adaptors.

Installing AddIns

There are two ways to install an AddIn. The simplest is to place it in the `/LocalLibrary/MesaAddIns` folder. The second way is to load it into Mesa via the Preferences panel. Select **Info → Preferences**, click on Add, and an Open Panel will appear. Select the AddIn to load, and Mesa will remember the location of the AddIn and load it every time itÕs launched.

FunctionAddIns

AddIn functions allow the user to seamlessly extend the functionality of Mesa. While Mesa provides a large set of built-in functions such as cos(), sin(), sum(), etc., there may be some functions specific to your application that have not been included. With FunctionAddIns, these functions can be written and used from within Mesa as though they were built in.

Mesa function AddIns must be written in Objective-C and must be a subclass of MesaAddIn. They must also respond to two methods: +funcName and execute: numberOfParams:sheet:. The factory method +funcName must return a unique name that the function will be called with. The second method, execute:numberOfParams:sheet:, is called when the cell containing the registered function name is to be evaluated. The items on the stack will be in reverse order from how they are passed to the function.

There are two other methods that can be implemented. These are beginRecalc and endRecalc. The first, beginRecalc, is called just before a recalculation is to occur. This gives the programmer the option to do pre-calculation set up such as executing a stored procedure in a database. The second, endRecalc, is called after the entire sheet has finished recalculation. This is the place where clean up operations should occur.

It is very important to remember that a function AddIn lives very close to the heart of Mesa. You should always remember to free any memory that you have allocated, and never access things that do not belong to the AddIn itself. Much of the underlying code that holds Mesa data could change at any time Ñ do not rely on anything that is not documented. It is extremely important to follow good programming practice and use only the functions we provide to access data from Mesa.

Note: gdb, the GNU debugger supplied with NEXTSTEP, does not allow the tracing or debugging of dynamically loaded objects. You must debug AddIns using printf() statements. You do not need to restart Mesa each time you make a change to your AddIn. To re-load all AddIns (and EventAddIns), select **Info** → **Preferences** and then click on OK.

Simple example:

One function that is not in Mesa, but might be desired, is a conversion from celsius to fahrenheit. This would take as an argument the current degrees in celsius and would return the corresponding fahrenheit value. One could type `=ctof(0.0)` in a cell, and the value displayed would be 32.0. LetÕs write it.

The first thing we have to do is include the header files that are needed. So we would include `MesaAddIn.h` and `MesaError.h` and any other header that we might need. Secondly, we need to create both a header and a Objective-C file for our function. LetÕs call them `ctof.h` and `ctof.m`. Next, we need to define what object we inherit from Ñ in this case MesaAddIn. Finally, we need to define what methods we will respond to: in this case only +funcName and execute: numberOfParams:sheet:.

Our files will look like this:

ctof.h:

```
#import "MesaAddIn.h"
#import <libc.h>

@interface ctofAddIn : MesaAddIn
{
}

+ (char *)funcName;
- execute:(void *)stack numberOfParams:(int)num sheet:(void
    *)sh;

@end
```

and ctof.m:

```
#import "ctof.h"
#import <libc.h>

@implementation ctofAddIn

+ (char *)funcName
{
}

- execute:(void *)stack numberOfParams:(int)num sheet:(void
    *)sh
{
}
@end
```

We now need to flesh out the two methods. First weÕll have `+funcName` return the name that it will be referenced with in the spreadsheet. We will call it CTOF, so `+funcName` will look like

```
+ (char *)funcName:
{
    return "CTOF";
}
```

Note: You may use upper or lower case letters for function names, but by convention, upper case letters are used.

Next we need to have the execute:numberOf-Params:sheet: method do the conversion of the first argument that is passed to it.

```
- execute:(void *)stack numberOfParams:(int)num sheet:(void
    *)sh
{
    AddInValue aiv;
    double n = 0;
    int error = noError;

    initAddInValue(&aiv);
    if (num != 1) pushErrorOnStack(stack,badFormulaError);
    else
    {
        popValueFromStack(stack,&aiv);
        n = getAddInValueNumber(&aiv,&error);
        if (error != noError) pushErrorOnStack(stack,error);
        else pushNumberOnStack(stack,(9.0 * n)/5.0 + 32.0);
    }
    freeAddInValue(&aiv);
    return self;
}
```

We now need to compile the program into object code that Mesa can read. We start by opening a terminal window and changing to our current directory. Now type: `cc -O ctof.m -c`

This will create `ctof.o` that Mesa can load in and run.

Try running Mesa and typing `=ctof(100)` into a cell. It should be replaced by `212.0`.

Note: Note: Do not strip the .o Þle. Mesa relies on the symbol table in the .o Þle to properly load it at run time.

Advanced Topics

Now that we have created a simple Mesa function AddIn, we are ready to explore some more options in creating AddIns. If you have a need to create many AddIns, Mesa simplifies this greatly by allowing a single subclass of MesaAddIn to define multiple functions. In this case there are three methods that the programmer must implement. They are +numFuncs that returns the number of functions defined in the AddIn, +funcName: that returns the name of the function for the corresponding number, and execute:numberOfParams:sheet:funcNumber: that is called when a cell that contains the function that corresponds to the number defined by the given function name needs to be recalculated. These methods replace the +funcName and execute:numberOfParams:sheet: methods. An AddIn should not contain both a +funcName and a +funcName: (note the colon) method.

Placing multiple functions in a single AddIn is easier to administer because there are fewer AddIn files. It also resolves the issue of linking in multiple copies of the same libraries. Creating a single AddIn with all the functions defined inside it and linking the libraries once to the AddIn will solve these problems. Again we will create a simple example.

Simple Example

We will now create a Mesa AddIn that defines not only a function to convert from Celsius to Fahrenheit, but also from Fahrenheit to Celsius. The basic structure is the same as the previous one except it uses the new methods for defining multiple functions.

```
conv.h:
#import "MesaAddIn.h"

@interface ConvAddIn : MesaAddIn
{
}

+ (int)numFuncs;
+ (char *)funcName:(int)functionNumber;
- execute:(void *)stack numberOfParams:(int)num sheet:(void
   *)sh funcNumber:(int)fn;

@end

conv.m:

#import "conv.h"
#import <libc.h>

@implementation ConvAddIn

+ (int)numFuncs
{
   return 2;
}

+ (char *)funcName:(int)functionNumber;
{
   static char *funcs[] = {"FTOC","CTOF"};

   return funcs[functionNumber];
}
```

```
- execute:(void *)stack numberOfParams:(int)num sheet:(void
   *)sh funcNumber:(int)fn
{
   AddInValue aiv;
   double n = 0;
   int error = noError;

   initAddInValue(&aiv);

   if (num != 1) pushErrorOnStack(stack,badFormulaError);
   else
   {
      popValueFromStack(stack,&aiv);
      n = getAddInValueNumber(&aiv,&error);
      if (error != noError) pushErrorOnStack(stack,error);
      else
         switch(fn)
         {
            case 0:
               pushNumberOnStack(stack,(5.0 * (n - 32.0))/
   9.0);
               break;
            case 1:
               pushNumberOnStack(stack,(9.0 * n)/5.0 + 32.0);
               break;
         }
   }
   freeAddInValue(&aiv);
   return self;
}
@end
```

Creating multi-function AddIns is nearly as simple as creating single function AddIns. One thing to note is that if your AddIn is large enough to have multiple files, they should be combined with the following command: `ld -r foo1.o foo2.o foo3.o -o out.o`

Methods

Factory Methods

funcName

+ (char *)funcName

Either this method or the methods funcName: and numFuncs must be implemented. This method returns the name for an individual function.

funcName:

+ (char *)funcName:(int)functionNumber

This method returns the name for the AddIn function numbered functionNumber. This method must be defined for multiple function AddIns.

numFuncs

+ (int)numFuncs

If there is more than one function defined in an AddIn, then the total number of functions must be returned here. This method must be implemented for multiple function AddIns.

Class Methods

beginRecalc

- beginRecalc

This method is called when a recalculation is about to begin. Pre-execution initialization should be done in this method, such as the query of a database before a calculation.

endRecalc

- endRecalc

This method is called when a recalculation has just ended. Post-recalculation clean-up should be implemented in this method.

execute:numberOfParams:sheet:

- execute:(void *)stack numberOfParams:(int)num sheet:(void *)sheet

  This is the method that is called when there is a single defined function in the AddIn. This method is called when the current worksheet is about to be recalculated and a cell has been found with the function name defined by +funcName. To pop the arguments off of the stack use popValueFromStack(). For more information about stack manipulation, see ÒManipulating Items on the AddIn stackÓ on page 13. If there are values to be returned, they should be pushed onto the stack with one of pushNumberOnStack(), pushStringOnStack(), pushAddressOnStack(), pushRangeOnStack(), or pushErrorOnStack(). You should always be certain to remove all the values on the stack that have been passed to the function.

execute:numberOfParams:sheet:funcNumber:

- execute:(void *)stack numberOfParams:(int)num sheet:(void *)sheet funcNumber:(int)funcNum

  This method will be called when a cell that contains one of the functions named in +funcName: needs to be recalculated. This method will only be called if +numFuncs is defined. To pop the arguments off of the stack, use popValueFromStack(). For more information about stack manipulation, see ÒManipulating Items on the AddIn stackÓ on page 13. If there are values to be returned, they should be pushed onto the stack with one of pushNumberOnStack(), pushStringOnStack(), pushAddressOnStack(), pushRangeOnStack(), or pushErrorOnStack(). You should always be certain to remove all the values on the stack that have been passed to the function.

Other Methods

next:

- (const char *)next:(const char *)string

This method defines a list for MesaÕs NEXT() function. It should return the next logical string for string, or NULL if no next string is available.

Functions that can be used by AddIns

Manipulating Items on the AddIn stack

numberOfItemsOnStack()

int numberOfItemsOnStack(void *stack)

If the stack is passed to this function, it will return the total number of items on the stack. This is different from the number of items on the stack for your function. The number of parameters passed to your function is given in the numberOfParameters:`(int)` variable.

popValueFromStack()

void popValueFromStack(void *stack, AddInValue *value)

Pops the top element off of stack. Remember that the stack is in reverse order. I.e., If your function is called with `=foo(1,2,3,4)`, the first pop will take 4 off the stack, the next 3, and so on.

Note: You must free the value you pop off the stack using freeAddInValue() in order to avoid memory leaks.

pushAddressOnStack()

void pushAddressOnStack(void *stack, short row, short col, void *sheet)

Pushes a cell address with coordinates row and col onto stack. The sheet argument is either the sheet that the function was called from or the sheet that is referred to by the address in a value that the function was passed as a parameter. This is the void pointer in the AddInValue.values.ad.cells structure element.

pushErrorOnStack()

void pushErrorOnStack(void *stack, int error)

Pushes the error error onto stack. This is typically used when an incorrect number of parameters is passed to the function or if the calculation yields an error. For a list of FunctionAddIn Errors see the Appendix.

pushNumberOnStack()

void pushNumberOnStack(void *stack, double value)

Pushes the number value onto stack. This is generally used at the end of execution of the function, and value is typically the return value.

pushRangeOnStack()

void pushRangeOnStack(void *stack, short ur, short uc, short lr, short lc, void *sheet)

Pushes a cell range specified by the coordinates ur, uc, lr, and lc onto stack. The sheet argument is either the sheet that the function was called from or the sheet that is referred to by the address in a value that the function was passed as a parameter. This is the in the AddInValue.values.ad.cells structure element. The range is specified as Top Row, Left Column, Bottom Row, Right Column.

pushStringOnStack()

void pushStringOnStack(void *stack, const char *string)

Pushes string onto stack. This is generally used at the end of execution of the function, and string is the typically the return value. Mesa makes a copy of the string that is passed to it. The string may be of any length and must be null-terminated.

Extracting Values

freeAddInValue()

void freeAddInValue(AddInValue *value)

Use this function to free an AddInValue allocated with initAddInValue().

Note: Remember to free all memory you allocate in order to avoid memory leaks.

getAddInValueNumber()

double getAddInValueNumber(AddInValue *value, int *error)

Returns the numerical value of value. A value popped off the stack may contain an address or a range. The getAddInValueNumber() function converts the AddInValue to a double. It also returns an error code in error that should be compared with the AddIn errors. If the error code is anything other than noError, the returned value is not valid. For a list of AddIn errors, see ÒÓ on pageXI.

getAddInValue()

void getAddInValue(AddInValue *value1, AddInValue *value2, int *error)

A value popped off the stack may contain an address or a range. You may want to be guaranteed a value that is either a string or a number. getAddInValue() converts AddInValues that are addresses or ranges to either strings, numbers, or errors. Call getAddInValue() with the value popped off the stack as value1 and an initialized AddInValue as value2. The error value will be returned as error. If the error is equal to noError, the returned value will be either a number or a string (the type is designated in the AddInValue.type field.) If the first AddInValue is an address or a range, then all referenced cells will be recalculated.

initAddInValue()

void initAddInValue(AddInValue *value)

Use this function to allocate storage space for value so that it can be popped off of the stack.

Note: Remember to free all memory that you allocate in order to avoid memory leaks.

Accessing the Sheet

getSheetFromPointer()

id getSheetFromPointer(void *sheet)

This function returns the id of sheet.

Note: Never call Mesa EventAddIn calls with this returned sheet id. This value can be cached for use within an EventAddIn. Only the Mesa functions outlined in this section should be used from within a function AddIn.

getValueForCell()

```
int getValueForCell(void *sheet, short row, short col,
    int *error, AddInValue *value)
```

Returns 1 if the cell specified by the coordinates row and col exists in sheet, 0 if not. The sheet argument should come from the AddInValue.values.ad.cells structure element. A range pushed on the stack may refer to a sheet other than the current worksheet (e.g., a linked worksheet.) If the cell exists, it will be returned in value. The error is returned in error. This function can be used to retrieve the value of an array of cells. This is valuable if a parameter to your function refers to a range of cells on the worksheet and you need to retrieve the value of each cell.

Types

`numberAddInValue`	The AddInValue is of type and the value is located in AddInValue.values.number.
`stringAddInValue`	The AddInValue is of type char * and is located in AddInValue.values.string. Do not change this variable.
`addressAddInValue`	The row and column of the address are in row1 and col1 and the worksheet is in AddInValue.values.ad.cells. To convert this to a double, use getAddInValueNumber(). To convert to a value (string, double, or error), use getAddInValue().
`rangeAddInValue`	The row and column of the address on the cell in the upper left corner are in row1 and col1, the address of the cell in the lower right is in row2 and col2, and the worksheet is in AddInValue.values.ad.cells. To convert this to a double, use getAddInValueNumber(). To convert to a value (string, double, or error), use getAddInValue(). If you want to collect data from all the cells in the range, use getValueForCell() for each cell in the range.
`errorAddInValue`	The AddInValue is of type Error.

EventAddIns

EventAddIns provide a way for custom programmers to extend the capability of Mesa from within Mesa itself. EventAddIns allow a custom programmer to create an AddIn to ÒlistenÓ to events that occur in the spreadsheet. These events can include the opening of new worksheets, mouse clicks, range selection, cell editing, and most any other significant event that might occur when a user uses the program. The programmer subscribes to the various events by simply implementing the method calls that are to be listened for. Any manipulation that a user can do to a worksheet can be done by the EventAddIn. EventAddIns can also add their own menu structures to the Mesa menu. Along with adding to the Mesa menu structure, they can also put up Panels, Alerts, and do most anything that a stand-alone application can do from within Mesa itself.

The creation of an EventAddIn is not much more difficult than the creation of a function AddIn. There is one method that all EventAddIns must implement, which names the AddIn. The method +eventName returns a text string that gives the EventAddIn a unique name. It is very important that your EventAddIn has a unique name. You can create an AddIn that is both a function AddIn and an EventAddIn by implementing the +eventName and either the +funcName or +funcName: methods.

When creating an EventAddIn, remember that you are working very close to the heart of Mesa. Thus, errors that you make can cause Mesa to unexpectedly quit.

Note: When exiting, always remember to free any memory allocated in order to avoid memory leaks. This is extremely important, since an EventAddIn can be loaded and unloaded at any point during the use of Mesa.

EventAddIns need to be written in Objective-C and should be a subclass of Object.

Methods

Class Methods

eventName

+ (char *)eventName

This method must be implemented. It returns a unique name for the EventAddIn.

Delegate Methods

A message will be sent to the EventAddIn at the appropriate time if the following methods are implemented. Nothing more than implementing the appropriate methods needs to be done. These methods are the way in which the EventAddIn is informed of what is happening to the worksheet.

addInScriptChanged:

- addInScriptChanged:sender

Sent when the EventAddIn header changes

baseFormatChanged:

- baseFormatChanged:sender

Sent when the global sheet format changes.

dataEntered:string:intoCellRow:col:

- dataEntered:sender string:(char *)string intoCellRow:(int)row col:(int)col

Sends the input string string and cell address row and col of data entered by the user.

- doubleClick:onCellRow:col:

- doubleClick:sender onCellRow:(int)row col:(int)col

  Gives the cell address row and col of a double-click of the mouse.

labelTableUpdated:

- labelTableUpdated:sender

  Informs the AddIn of a change to the Label Index.

pageWillPrint:page:of:pageRect:viewRect:

- pageWillPrint:sender page:(int)num of:(int)of pageRect:(NXRect *)rect viewRect:(NXRect *)view

  Before a page is printed, this method is called. num is the number of the page that is about to be printed, of is the total number of pages, rect is a rectangle the size of the page, and view is a rectangle the size of the visible (spreadsheet) area that will be printed.

rangeCleared:upperRow:col:lowerRow:col:

- rangeCleared:sender upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Informs the AddIn that the range specified by ur, uc, lr, and lc has been cleared.

rangeCopied:fromUpperRow:col: lowerRow:col:toUpperRow:col: lowerRow:col:

- rangeCopied:sender fromUpperRow:(int)ur1 col:(int)uc1 lowerRow:(int)lr1 col:(int)lc1 toUpperRow:(int)ur2 col:(int)uc2 lowerRow:(int)lr2 col:(int)lc2

  Gives the source range ur1, uc1, lr1, lc1 and destination range ur2, uc2, lr2, and lc2 for a range copy operation.

rangeDidRecalc:upperRow:col:lowerRow:col:

- rangeDidRecalc:sender upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Informs the AddIn that the range specified by ur, uc, lr, and lc has been recalculated.

rangeFormatChanged:upperRow:col:lowerRow:col:

- rangeFormatChanged:sender upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Informs the AddIn of a change in the format of the range specified by ur, uc, lr, and lc.

rangeMoved:fromUpperRow:col:lowerRow:col:toUpperRow:col: lowerRow:col:

- rangeMoved:sender fromUpperRow:(int)ur1 col:(int)uc1 lowerRow:(int)lr1 col:(int)lc1 toUpperRow:(int)ur2 col:(int)uc2 lowerRow:(int)lr2 col:(int)lc2

  Gives the source range ur1, uc1, lr1, lc1 and destination range ur2, uc2, lr2, and lc2 for a range move operation.

rangeSelected:upperRow:col:lowerRow:col:

- rangeSelected:sender upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Gives the source range ur1, uc1, lr1, lc1 and destination range ur2, uc2, lr2, and lc2 for a range select operation.

sheetBecameTop:

- sheetBecameTop:sender

  Sent when the worksheet has become the topmost (current) worksheet.

sheetDidClose:

- sheetDidClose:sender

  Sent when the worksheet has closed.

sheetDidOpen:

- sheetDidOpen:sender

  Sent when the worksheet has opened.

sheetDidPrint:

- sheetDidPrint:sender

  Sent when the worksheet has completed printing.

sheetDidRecalc:

- sheetDidRecalc:sender

  Sent when the worksheet has completed recalculation.

sheetDidRedisplay:

- sheetDidRedisplay:sender

  Sent when the worksheet has been re-displayed.

sheetResignedTop:

- sheetResignedTop:sender

  Sent when the worksheet has resigned its position as the topmost (current) worksheet.

sheetSizeChanged:toWidth:height:

- sheetSizeChanged:sender toWidth:(int)width height:(int)height

  Sent when the worksheet size has changed to width columns and height rows.

sheetWillClose:

- sheetWillClose:sender

  Sent when the worksheet is about to close.

sheetWillPrint:

- sheetWillPrint:sender

  Sent when the worksheet is about to be printed.

Sheet Methods

These methods can be sent to the ÒsenderÓ of the delegate methods.

addLabel:upperRow:col:lowerRow:col:

- addLabel:(char *)label upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Assigns the name label to the range specified by ur, uc, lr, and lc.

associateLabel:with:and:num:offset:orientation:

- associateLabel:(char *)label with:(char **)tags and:(void *)mv num:(int)num offset:(int)offset orientation:(int)orient

  This is an extremely powerful but complex method. Please read carefully. It associates an array of strings tags with the label label. Mesa will then look for that string in the range of the label and place the corresponding value at a specified offset from the cell the string is found in. The offset can be either in the horizontal or vertical direction as specified by orient.

associateRangeUpperRow:col:lowerRow:col:with:and:num: offset:orientation:

- associateRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc with:(char **)tags and:(void *)mv num:(int)num offset:(int)offset orientation:(int)orient

  This is an extremely powerful but complex method. Please read carefully. It associates an array of strings tags with the range specified by ur, uc, lr, and lc. Mesa will then look for that string in the range and place the corresponding value at a specified offset from the cell the string is found in. The offset can be either in the horizontal or vertical direction as specified by orient.

changed

- (BOOL)changed

  Returns whether the worksheet has changed since the last save operation.

copyRangeUpperRow:col:lowerRow:col:toUpperRow:col: lowerRow:col:

- copyRangeUpperRow:(int)sur col:(int)uc1 lowerRow:(int)lr1 col:(int)lc1 toUpperRow:(int)ur2 col:(int)uc2 lowerRow:(int)lr2 col:(int)lc2

  Copies the information from the source range specified by ur1, uc1, lr1, and lc1 to the destination range specified by ur2, uc2, lr2, and lc2. If the destination range is smaller than the source range, the data will be clipped. If the destination range is larger than the source range, the information will be repeated to fill the destination range.

createGraphWithRect:type:upperRow:col:lowerRow:col:

- (char *)createGraphWithRect:(NXRect *)rect type:(int)type upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

Creates a graph of type type in rect, using the data in the range specified by ur, uc, lr, and lc. For a list of constants for graph types, see the table ÒGraph TypesÓ on pagH.

currentRangeUpperRow:col:lowerRow:col:

- currentRangeUpperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr col:(int *)lc

Returns the currently selected range, placing the coordinates in ur, uc, lr, and lc.

deleteLabel:

- deleteLabel:(char *)name

Deletes the label name from the Label Index.

deleteScript:

- deleteScript:(char *)name

Deletes the MScript script name.

doRecalc

- doRecalc

Forces the worksheet to recalculate itself.

findLabel:upperRow:col:lowerRow:col:

- (BOOL)findLabel:(char *)name upperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr col:(int *)lc

Finds the range coordinates ur, uc, lr, and lc for the range label name. Returns TRUE if the label is found, FALSE if itÕs not.

getCellStringRow:col:

- (char *)getCellStringRow:(int)row col:(int)col

  Returns the actual string typed into the cell address row and col by the user. Mesa stores both the user-entered string and its ÒvalueÓ Ñ this method returns the input string, not the value.

getEventHeader:

- (char *)getEventHeader:(char *)name

  This method returns the event header for the EventAddIn name. The event header is ASCII text that can be edited by the user in the Event Inspector. Information specific to the worksheet can be stored in the event header for later use by the EventAddIn.

getLabelNumber:

- (char *)getLabelNumber:(int)n

  Returns the name of the nth label in the worksheet.

getMOLIValue:row:col:

- getMOLIValue:(void *)value row:(int)row col:(int)col

  Retrieves value from the spreadsheet from the cell address row and col. The item will be in a structure pointed to by value. The structure is in this format:

```
enum {stringMOLIValue, numberMOLIValue, errorMOLIValue};

typedef struct _MOLIValue {
    int type;
    short row,col;
    union {
        char *string;
        double number;
        int error;
        } values;
    } MOLIValue;
```

Note: When you are done with the information, call freeMOLIValue(int n, MOLIValue *value) to free the structure.

getMOLIValues:num:upperRow:col:lowerRow:col:

- getMOLIValues:(void **)values num:(int *)num upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Constructs an array of values from the spreadsheet from the range specified by ur, uc, lr, and lc, placing the pointer to this array in values and the total number of values in num. The array that is returned will be sparse (i.e., only cells that exist will be returned). Remember to test the address of each value and its type. For the format of the MOLIValue structure, see ÒgetMOLIValue:row:col:Ó on page 25.

  Note: When you are done with the information, call freeMOLIValue(int n, MOLIValue *value) to free each structure in the array, and then call free() on the array itself.

getReportNumber:

- (char *)getReportNumber:(int)n

  Returns the name of the nth defined report in the worksheet.

getTheGraphColor:element:

- (NXColor)getTheGraphColor:(char *)name element:(int)n

  Returns the color of the nth element of the graph name.

getTheGraphColor:item:

- (NXColor)getTheGraphColor:(char *)name item:(int)item

  Returns the color of the item given by the constant item from the graph name. For a table of graph items that have color, see ÒGraph Item Constants with ColorÓ on page II.

getTheGraphDouble:item:

- (double)getTheGraphDouble:(char *)name item:(int)item

  Returns the double value of the item given by the constant item from the graph name. For a table of graph items that have double values, see ÒGraph Item Constants with Double ValuesÓ on page III.

getTheGraphFont:item:

- getTheGraphFont:(char *)name item:(int)item

  Returns the font of the item given by the constant item from the graph name. For a table of graph items that have fonts, see ÒGraph Item Constants with FontsÓ on pageIV.

getTheGraphInt:item:

- (int)getTheGraphInt:(char *)name item:(int)item

  Returns the integer value of the item given by the constant item from the graph name. For a table of graph items that have integer values, see ÒGraph Item Constants with Integer ValuesÓ on pageV.

getTheGraphRange:item:range::::

- getTheGraphRange:(char *)name item:(int)item range:(int *)ur :(int *)uc :(int *)lr :(int *)lc

  Returns the range reference of the item given by the constant item from the graph name, placing the coordinates in ur, uc, lr, and lc. For a table of graph items that have range references, see ÒGraph Item Constants with Range ReferencesÓ on pageIV.

getTheGraphString:item:

- (char *)getTheGraphString:(char *)name item:(int)item

  Returns the string value of the item given by the constant item from the graph name. For a table of graph items that have string values, see ÒGraph Item Constants with String ValuesÓ on pageIII.

miniturizeWindow:

- miniturizeWindow:(int)win

  Miniaturizes window numbered win. Note the spelling of the method.

**moveRangeUpperRow:col:lowerRow:col:toUpperRow:col:
lowerRow:col:(int)**

- moveRangeUpperRow:(int)ur1 col:(int)uc1 lowerRow:(int)lr1 col:(int)lc1 toUpperRow:(int)ur2 col:(int)uc2 lowerRow:(int)lr2 col:(int)lc2

  Copies the information from the source range specified by ur1, uc1, lr1, and lc1 to the destination range specified by ur2, uc2, lr2, and lc2. The ranges must be of the same size or an error will result.

nameOfQuery:

- (char *)nameOfQuery:(int)n

  Returns the name of the nth Sybase query in the worksheet.

nameOfScript:

- (char *)nameOfScript:(int)n

  Returns the name of the nth MScript script in the worksheet.

numberOfLabels

- (int)numberOfLabels

  Returns the total number of labels defined in the worksheet.

numQueries

- (int)numQueries

  Returns the total number of Sybase queries that are defined by the Query Inspector in the worksheet.

numReports

- (int)numReports

  Returns the total number of defined Reports in the worksheet.

numScripts

- (int)numScripts

  Returns the total number of defined MScript scripts in the worksheet.

performQuery:

- performQuery:(char *)name

  Performs the Sybase query name. Note that the query must be defined and built with the Query Inspector before being used in an EventAddIn.

printReport:

- printReport:(char *)name

  Prints the pre-defined Report name.

putMOLIValue:row:col:

- putMOLIValue:(void *)value row:(int)row col:(int)col

  Places value in the spreadsheet in the cell address row and col. The value should be in an structure pointed to by value. Mesa removes the specified cell and then inserts the supplied information. Remember to set the row and col of the MOLIValues. If the cell is not in the address specified, it will not be inserted. The structure is in the format:

```
enum {stringMOLIValue, numberMOLIValue, errorMOLIValue};

typedef struct _MOLIValue {
    int type;
    short row,col;
    union {
        char *string;
        double number;
        int error;
        } values;
    } MOLIValue;
```

Note: When you are done with the information, call freeMOLIValue(int n, MOLIValue *value) to free the structure.

putMOLIValues:num:upperRow:col:lowerRow:col:

- putMOLIValues:(void *)values num:(int)num upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Places an array of num MOLIValues values in the worksheet. The values will be placed in the worksheet in the range specified by ur, uc, lr, and lc. Mesa removes all cells from that range and then inserts the supplied cells. Make sure to set the row and col for each of the MOLIValues. If a cell is not in the range specified, it will not be inserted.

Note: When you are done with the information, call freeMOLIValue(int n, MOLIValue *value) to free each structure in the array, then call free() on the array itself.

recalcRangeUpperRow:andCol:lowerRow:andCol:

- recalcRangeUpperRow:(int)ur andCol:(int)uc lowerRow:(int)lr andCol:(int)lc

Recalculates the range of cells specified by ur, uc, lr, and lc.

redrawAll

- redrawAll

Forces all open views into the worksheet to redraw themselves.

resetColSizeFrom:to:

- resetColSizeFrom:(int)first to:(int)last

Set the column size of the range of columns specified by first and last to the default width for the sheet.

resetRowSizeFrom:to:

- resetRowSizeFrom:(int)first to:(int)last

Set the row size of the range of rows specified by first and last to the default height for the sheet.

runScript:

- runScript:(char *)name

Runs the pre-defined MScript script name.

save:

- save:sender

Saves the worksheet with its current name. Send self as sender.

saveAs:

- saveAs:sender

  Displays a Save panel for the user to name the file and saves the worksheet under the new name.

scriptFunc:num:return:

- (int)scriptFunc:(AddInValue *)values num:(int)num
  return:(AddInValue *)return

  Allows an AddIn to be called from MScript. values is an array of AddIn values which are the parameters. num is the number of values. The sheet may be modified from an Event AddIn. If this method returns TRUE, it has set value to be the value to push back on the stack. This method differs from a Function AddIn because it can modify other cells during its execution.

setAutoRecalcOn:

- setAutoRecalcOn:(int)state

  Sets AutoRecalc on if state is 1 or off if state is 0.

setBestColSizeFrom:to:

- setBestColSizeFrom:(int)first to:(int)last

  SmartSizes the range of columns specified by first and last, to the best width for the data they contain.

setBestRowSizeFrom:to:

- setBestRowSizeFrom:(int)first to:(int)last

  SmartSizes the range of rows specified by first and last, to the best height for the data they contain.

setBkgNXColor:

- setBkgNXColor:(NXColor)color

  Sets the worksheetÕs background color tocolor.

setCellString:row:col:

- setCellString:(char *)string row:(int)row col:(int)col

  Sets the input string of the cell at row and col to string. This enters the string as though it were typed by the user. updateDataEntry should be called after this method in order to update what is displayed on the screen. See ÒupdateDataEntryÓ on page 41.

setChanged:

- setChanged:(BOOL)bool

  Sets whether the worksheet has been changed. This method should be called when you make a change to the worksheet.

setClearBkg:

- setClearBkg:(int)state

  Sets the background of the sheet clear if state is 1 or opaque if state is 0. This is useful for pasting EPS images of worksheet ranges into other applications.

setColSize:fromCol:to:

- setColSize:(int)size fromCol:(int)first to:(int)last

  Set the width of the range of columns specified by first and last. The argument size is in pixels.

setColWidth:

- setColWidth:(int)width

  Sets the global column width of the worksheet to width pixels.

setEventHeader:forAddIn:

- setEventHeader:(char *)string forAddIn:(char *)name

  This method sets the event header for the EventAddIn name to string. The event header is ASCII text that can be edited by the user in the Event Inspector. Information specific to the worksheet can be stored in the event header for later use by the EventAddIn.

setGrid:

- setGrid:(int)state

  Sets the worksheetÕs grid on ifstate is 1 or off if state is 0.

setGridNXColor:

- setGridNXColor:(NXColor)color

  Sets the grid color of the worksheet to color.

setMaxNumberOfColumns:

- setMaxNumberOfColumns:(int)cols

  Changes the worksheet width to cols columns.

setMaxNumberOfRows:

- setMaxNumberOfRows:(int)rows

  Changes the worksheet height to rows columns.

setPathName:

- setPathName:(char *)path

  Sets the path for the worksheet to path.

setRangeToDefaultsUpperRow:col:lowerRow:col:

- setRangeToDefaultsUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Sets the range specified by ur, uc, lr, and lc to the formatting defaults for its Style Template.

setRangeUpperRow:col:lowerRow:col:toAlignment:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toAlignment:(int)align

  Sets the range specified by ur, uc, lr, and lc to the alignment align. The alignment options are

  ¥ leftMesaAlign

  ¥ rightMesaAlign

  ¥ centerMesaAlign

  ¥ smartMesaAlign

¥ÞllMesaAlign

You can view the different alignments from within Mesa by changing the alignment of a range with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toAltColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toAltColor:(NXColor)color

Sets the alternate text color (the color of negative values) in the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toBaseFormat:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBaseFormat:(int)b

Sets the range to one of the Eight Style Template formats. You can view different templates from within Mesa by changing the Style Templates for a range of cells with the Cell Style Inspector. The Style Template attributes can be defined with the Sheet Inspector.

setRangeUpperRow:col:lowerRow:col:toBaseline:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBaseline:(int)state

Sets the range specified by ur, uc, lr, and lc to baseline alignment if state is 1, or non-wrap if state is 0.

setRangeUpperRow:col:lowerRow:col:toBkgColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBkgColor:(NXColor)color

Sets the background color of the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toBorder:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBorder:(int)type

  Sets the border type of the range specified by ur, uc, lr, and lc to type. For a list of the border type constants, see ÒBordersÓ on pageIX. You can view different border types from within Mesa by changing the Border Type for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toBorderColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBorderColor:(NXColor)color

  Sets the border color of the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toBottomBorder:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBottomBorder:(int)type

  Sets the Bottom of the range specified by ur, uc, lr, and lc to type. For a list of the border type constants, see ÒBordersÓ on page IX. You can see what they will look like from within Mesa by changing the Border Type for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toColor:(NXColor)color

  Sets the text color of the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toDefaultAlignment:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDefaultAlignment:(int)state

  Sets the range specified by ur, uc, lr, and lc to its default text alignment if state is 1, or resets it to the text alignment stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultAltColor:

```
- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr
    col:(int)lc toDefaultAltColor:(int)state
```

Sets the range specified by ur, uc, lr, and lc to its default alternate color if state is 1, or resets it to the alternate color stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultBkg:

```
- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr
    col:(int)lc toDefaultBkg:(int)state
```

Sets the range specified by ur, uc, lr, and lc to its default background color if state is 1, or resets it to the background color stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultBorder:

```
- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr
    col:(int)lc toDefaultBorder:(int)state
```

Sets the range specified by ur, uc, lr, and lc to its default border type if state is 1, or resets it to the border type stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultBorderColor:

```
- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr
    col:(int)lc toDefaultBorderColor:(int)state
```

Sets the range specified by ur, uc, lr, and lc to its default border color if state is 1, or resets it to the border color stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultDisplayFormat:

```
- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr
    col:(int)lc toDefaultDisplayFormat:(int)state
```

Sets the range specified by ur, uc, lr, and lc to its default display format if state is 1, or resets it to the display format stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultFont:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDefaultFont:(int)state

  Sets the range specified by ur, uc, lr, and lc to its default font if state is 1, or resets it to the font stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultTextColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDefaultTextColor:(int)state

  Sets the range specified by ur, uc, lr, and lc to its default text color if state is 1, or resets it to what is stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDefaultUnderline:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDefaultUnderline:(int)state

  Sets the range specified by ur, uc, lr, and lc to its default underline type if state is 1, or resets it to the underline type stored in the manual formatting of each cell in the range if state is 0.

setRangeUpperRow:col:lowerRow:col:toDisplayFormat:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDisplayFormat:(int)format

  Sets the Display Format for the range specified by ur, uc, lr, and lc to format. For a list of formatting constants, see ÒDisplay FormatsÓ on pageVIII. You can view different display formats from within Mesa by changing the Display Format for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toFont:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toFont:font

  Sets the font of the range specified by ur, uc, lr, and lc to font.

setRangeUpperRow:col:lowerRow:col:toHasBkgColor:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toHasBkgColor:(int)state

  Sets the range specified by ur, uc, lr, and lc to have a background color if state is 1, or a clear background if state is 0.

setRangeUpperRow:col:lowerRow:col:toLeftBorder:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toLeftBorder:(int)type

  Sets the Left border type of range specified by ur, uc, lr, and lc to type. For a list of the border type constants, see ÒBordersÓ on pageX. You can view different borders from within Mesa by changing the Border Type for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toMode:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toMode:(int)mode

  Sets the input mode of the cells in the range specified by ur, uc, lr, and lc to mode. For a list of input type constants, see ÒInput TypeÓ on pageX.

setRangeUpperRow:col:lowerRow:col:toPrecision:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toPrecision:(int)prec

  Sets the display precision for the range specified by ur, uc, lr, and lc to prec. The precision must be an integer between 0 and 15.

setRangeUpperRow:col:lowerRow:col:toRightBorder:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toRightBorder:(int)type

  Sets the Right border type of the range specified by ur, uc, lr, and lc to type. For a list of the border type constants, see ÒBordersÓ on pageX. You can view different borders from within Mesa by changing the Border Type for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toTopBorder:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toTopBorder:(int)type

  Sets the Top border type of the range specified by ur, uc, lr, and lc to type. For a list of the border type constants, see ÒBordersÓ on pageX. You can view different borders from within Mesa by changing the Border Type for a range of cells with the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toUnderline:

- (int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toUnderline:(int)type

  Sets the Right underline type of the range specified by ur, uc, lr, and lc to type. You can view different underline types from within Mesa by changing the underline type for a range of cells with the Font view of the Cell Style Inspector.

setRangeUpperRow:col:lowerRow:col:toWrap:

- setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toWrap:(int)state

  Sets the range specified by ur, uc, lr, and lc to wrap alignment if state is 1, or non-wrap if state is 0.

setRowHeight:

- setRowHeight:(int)height

  Sets the global row height of the worksheet to height pixels.

setRowSize:fromRow:to:

- setRowSize:(int)size fromRow:(int)first to:(int)last

  Set the height of the range of rows specified by first and last. The argument size is in pixels.

setTheGraph:element:color:

- setTheGraph:(char *)name element:(int)n color:(NXColor)color

  Sets the nth element of the graph name to color.

setTheGraph:item:color:

- setTheGraph:(char *)name item:(int)item color:(NXColor)color

  Sets the item given by the constant item on the graph name to color. For a table of graph items that have color, see ÒGraph Item Constants with ColorÓ on pageII.

setTheGraph:item:doubleVal:

- setTheGraph:(char *)name item:(int)item doubleVal:(double)value

  Sets the item given by the constant item on the graph name to the double value value. For a table of graph items that have double values, see ÒGraph Item Constants with Double ValuesÓ on pageIII.

setTheGraph:item:font:

- setTheGraph:(char *)name item:(int)item font:font

  Sets the item given by the constant item on the graph name to font. For a table of graph items that have fonts, see ÒGraph Item Constants with FontsÓ on pageIV.

setTheGraph:item:intValue:

- setTheGraph:(char *)name item:(int)item intValue:(int)value

  Sets the item given by the constant item on the graph name to the integer value value. For a table of graph items that have integer values, see ÒGraph Item Constants with Integer ValuesÓ on pageV.

setTheGraph:item:range::::

- setTheGraph:(char *)name item:(int)item range:(int)ur :(int)uc :(int)lr :(int)lc

  Sets the item given by the constant item on the graph name to the range specified by ur, uc, lr, and lc. For a table of graph items that may have range references, see ÒGraph Item Constants with Range ReferencesÓ on pageIV.

setTheGraph:item:stringVal:

- setTheGraph:(char *)name item:(int)item stringVal:(char *)string

  Sets the item given by the constant item on the graph name to the string value string. For a table of graph items that have string values, see ÒGraph Item Constants with String ValuesÓ on pageIII.

sheetTitle

- (char *)sheetTitle

  Returns the title of the worksheet.

totalColumns

- (int)totalColumns

  Returns the total number of columns in the worksheet.

totalRows

- (int)totalRows

  Returns the total number of rows in the worksheet.

updateDataEntry

- updateDataEntry

  Flushes the input buffer to the text field. This method is useful in conjunction with the setCellString:row: col: method. See ÒsetCellString:row:col:Ó on page32.

Application Methods

These methods are sent to [NXApp].

addMenuItem:target:action:

- addMenuItem:(char *)name target:target action: (SEL)action

  Adds the menu item name to MesaÕs AddIn menu with a target of target and action action. Returns the id for the item added.

getOpenSheets

- getOpenSheets

  Returns a list of open sheet objects. Neither it nor its contents need to be freed.

newWorksheet

- newWorksheet

  Opens a new (Untitled) worksheet.

openMesaFile:

- openMesaFile:(const char *)path

  Opens the Mesa worksheet path. A full pathname must be given.

pathToAddin:

- (const char *)pathToAddin:sender

  Returns the path to the AddIn.

removeMenuItem:

- removeMenuItem:item

  Removes the menu item whose id is item.

setSubmenu:forItem:

- setSubmenu:myMenu forItem:myItem

  Sets a submenu of id myMenu for the menu item of id myItem.

topSheet

- topSheet

  Returns the id for the topmost (current) worksheet.

AdaptorAddIns

Mesa supports a database server adaptor, which allows the creation of AdaptorAddIns that will link to SQL servers other than Sybase (the built-in default). The Factory and Instance methods listed below must be implemented by your adaptor as listed.

Factory Methods

databaseName

+ (char *)databaseName

This method should return the name of the adaptor, e.g. ÒORACLEÓ or ÒTERADATAÓ.

Instance Methods

beginSQLTransaction

- beginSQLTransaction

Begins a set of SQL transactions. This method should implement the code that sets up the error handler.

connectToSQLServer:user:password:

- (int)connectToSQLServer:(char *)server user:(char *)user password:(char *)pass

Connects to the SQL server named server (may be blank) for the user user (if blank, the current user's login name) and the password pass. It returns YES if the connection was successful and NO if the connection failed.

convertSQLColumn:num:string:

- (int)convertSQLColumn:(int)col num:(double *)num string:(char **)string

Converts the colth column of data in the current row of returned data into a number or string. Returns the type of data in the column. nullSQLValue denotes that no valid or convertible data existed in the column. numberSQLValue is for numeric values, and the actual numeric value is stored in the double *num. stringSQLValue is for string values. The correct amount of space must be allocated using malloc() and the data must be copied into the string (remember to null terminate the string). Then set *string to the string pointer. Mesa will free() the malloc()Õed memory. If the information is in date format, convert it to a date serial number (i.e. the number of days since Jan 1, 1970), place the number in *value and return dateSQLValue.

convertSQLComputedColumn:alt:num:string:

- (int)convertSQLComputedColumn:(int)col alt:(int)alt num:(double *)num string:(char **)string

This method is the same as convertSQLColumn:num: string:, but operates on computed columns. See ÒconvertSQLColumn:num:string:Ó on page43.

endSQLTransaction

- endSQLTransaction

Ends the SQL transaction. This method should close the connection with the server and remove any error handlers.

execSQLString:

- execSQLString:(char *)string

Executes the SQL statement string.

getNameOfSQLColumn:

- (char *)getNameOfSQLColumn:(int)num

Returns the name of the numth column in the query, where columns are numbered 1 to N.

getNextSQLRow

- (int)getNextSQLRow

Retrieves the next row of data returned from the SQL server. Return noMoreSQLRows if there are no more rows of data. Return moreSQLRows if the row of data being returned is not a computed row. If the next row is a computed row, return a positive number denoting the computed type of the row to be used in the getNumSQLComputedColumns: and convertSQLComputedColumn:alt:num:string: methods. See ÒgetNumSQLComputedColumns:Ó on page 45 and ÒconvertSQLComputedColumn:alt:num:string:Ó on page 44.

getNumSQLColumns

- (int)getNumSQLColumns

Return the number of columns of data being returned from this query.

getNumSQLComputedColumns:

- (int)getNumSQLComputedColumns:(int)result

Returns the number of columns in the computed result result. result is the number returned by the method getNextSQLRow. See ÒgetNextSQLRowÓ on page 44.

getSQLResult

- (int)getSQLResult

Returns the result of the query. Return noMoreSQLResults if there is no more data coming back from the server. Return SQLSuccess if there is data and SQLFailed if the query failed (note that noMoreSQLResults must be returned on subsequent calls).

Application methods

This methods is implemented by NXApp:

handleSQLError:

- handleSQLError:(char *)errorString

Correctly handles an error condition by placing the error errorString in a dialog box, in the range of data, both, or neither, depending on the settings in the given query. The SQL Adaptor should message this method from the error handlers.

3. MOLI

Mesa Object Library Interface

The Mesa Object Library Interface (MOLI) allows custom programs to easily communicate with Mesa and to transfer of data into and out of a Mesa worksheet. Applications of this include real-time data acquisition, automation of report generation, and access to the numerical and graphical functions of Mesa. MOLI can greatly reduce the time to complete a custom programming project. MOLI gives a great deal of flexibility to custom application programmers; with it they can create integrated applications that can make use of the functionality of Mesa without the user even knowing that a spreadsheet is running alongside their custom application.

As an example, we might create a simple application that allows a stock trader to watch specific types of stocks. This might involve two applications that make simultaneous use of MOLI: a real time data feed that places the updated stock prices into a worksheet, and a second one that allows the trader to look at specific prices. The data feed and Mesa could be hidden from the user, who would only see the custom application. This custom application could be as simple or as complicated as required. Once data is fed into the worksheet and any recalculations that have been set up are done by Mesa, the traderÕs application will be updated and might put up an alert that a price had fallen below or risen above a certain level. This provides an easy way for a professional-looking custom application to distill large volumes of data into a more manageable form.

A consistent object-oriented interface allows programmers to have live spreadsheet views in their custom application that can include not only ranges of cells, but also graphs, in a fraction of the time it would normally take. Both ranges and graphs that are displayed in the custom programs can be manipulated just as they can be in a Mesa worksheet.

Applications that make use of MOLI should be written in Objective-C. Source files should include the header file `MesaObjectLibraryInterface.h`, and your project should include the object file `MesaObjectLibraryInterface.o`.

Add `MesaObjectLibraryInterface.o` to your project using ProjectBuilder. To do this, double-click on the Other Sources icon in ProjectBuilder.

MOLI includes three classes that you can either use directly or subclass. They are MesaObject, MesaListen, and MesaView. In the MesaProgramming/NeXTSTEP3.0 folder there is folder called MOLIDemo. It contains a sample program called MOLIDemo.app that makes use of MOLI and simulates the feeding of data into a worksheet that in turn recalculates clientsÕ portfolios and creates graphs based on those numbers

MOLI Classes

MesaListen

With the vast amount of data that most people must process, it is critical that we have a way for programs to help us sort through the irrelevant data and get to the data that we need to work with. The `signal()` function inside Mesa provides exactly that capability. Imagine that you are trading gold, and while your stock feed may provide hundreds of prices, you would only like to be alerted when the price of gold in Zurich is 1% greater than the price in London. The `signal()` function sends a range of cells to a waiting MesaListen object in your custom program when the given ÒsignalÓ becomes true. With our gold example, we would write in a Mesa worksheet:

```
=signal(zurich_price / 101% > london_price,
   "BuyZurich", zurich_price);
```

This would alert us when the price of gold in zurich is 1% above the price in London, and would send the range of cells in the third argument to the MesaListen object, in this case the range that is contained in the label `zurich_price`. In this case, both `zurich_price` and `london_price` are labels Ñ you could replace them with cell references. The signal will be sent to your MesaListen object via the gotMessage:num: forUpperRow:upperCol:lowerRow:lowerCol: method.

Methods

initToPort:

- initToPort:(char *)name

Passes the name of the port, name, to be listened to. The port name name should be alphanumeric, start with a letter, and be a maximum of 32 characters.

free

- free

Disconnects the object from its worksheet and frees it.

gotMessage:num:forUpperRow:upperCol:lowerRow:lowerCol:

- gotMessage:(MOLIValue *)mv num:(int)num
  forUpperRow:(int)ur upperCol:(int)uc lowerRow:(int)lr
  lowerCol:(int)lc

This method should be overridden in a subclass to perform the given tasks of the MesaListen object. This message is sent to the object when it receives a signal, and passes the data in the range that is specified by the rectangular coordinates ur, uc, lr, lc. The default method returns self.

Simple Example:

Create a new application with ProjectBuilder (for NEXTSTEP 3.0) and create a Text Field in the main window. Add your MesaListen Object and connect the Text Field to the id priceTextField, and the main window to the id myWindow.

Now we are ready to write the code for this simple example. The first thing to do is add our initialization code to our appDidInit: method

```
myListen = [[MyListen alloc] initToPort:"BuyZurich"];
```

and in appWillTerminate: we need to free the object:

```
[myListen free];
```

Now all we have to do is create our subclass of MesaListen.

```
MyListen.h:
@interface MyListen : MesaListen
{
   id myWindow;
   id priceTextField;
}

- gotMessage:(MOLIValue *)mv num:(int)num forUpperRow:(int)ur
   upperCol:(int)uc lowerRow:(int)lr lowerCol:(int)lc;
@end

MyListen.m:

@implementation MyListen
- gotMessage:(MOLIValue *)mv num:(int)num forUpperRow:(int)ur
   upperCol:(int)uc lowerRow:(int)lr lowerCol:(int)lc
{
   [super gotMessage:mv num:num forUpperRow:ur upperCol:uc
   lowerRow:lr
      lowerCol:lc];

   [myWindow makeKeyAndOrderFront:self];  // bring application
   to front

   // display the current gold price
   if (mv -> type == numberMOLIValue)
      [priceTextField setDoubleValue:mv -> values.number];

   return self;
   }
@end
```

With this code we have a custom application that alerts the user when the price of gold in Zurich is 1% greater than the price in London, by coming to the front and displaying the current price in Zurich.

MesaObject

The MesaObject class allows an external program to connect to a worksheet that is open in Mesa and perform operations on that worksheet. When you initialize a MesaObject object you must specify a path to a valid Mesa file, or create a new worksheet. If Mesa is not running when you initialize your object, the workspace is asked to run Mesa.

Note: Mesa must be in a standard applications directory (typically either `/LocalApps` or `~/Apps`) for the workspace to Þnd it. If the workspace can not Þnd Mesa, a NULL pointer will be returned. This is a condition that should always be checked for.

Please use only the functions and methods documented. These are the only ones that will be supported in the future. If you need functionality that is not provided, please send e-mail or call.

freeMOLIValue

void freeMOLIValue(int num,MOLIValue *mv)

This function frees the memory allocated for returned values from a message to getValues:num:upperRow:col:lowerRow:col: or getValues:forLabel:num:upperRow:col:lowerRow:col:. Call this function with the number of items returned, num, and the array that they were placed in, mv. Remember to do this in order to avoid memory leaks.

Factory Methods

Unless otherwise noted, the following methods return an error value if they are not successful. The error values can be found in ÒMOLI ErrorsÓ on pagXII.

getErrorText:

+ (const char *)getErrorText:(int)error

Given a MOLIError error, this method returns a string containing the error text. This string should not be modified and does not need to be freed.

hide

+ (int)hide

Hides the Mesa application.

isCellAddress:row:col:

+ (int)isCellAddress:(char *)cell row:(int *)row col:(int *)col

Converts the string cell to a row and column address and places the coordinates in row and col. Returns the number of characters in the address, or 0 if the string is not a valid address.

isRange:upperRow:col:lowerRow:col:

+ (int)isRange:(char *)range upperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr col:(int *)lc

Converts the string range to its row and column coordinates and places the coordinates in ur, uc, lr, and lc. Returns the number of characters in the range, or 0 if the string is not a valid range.

listOpenSheets:num:

+ (int)listOpenSheets:(char ***)sheets num:(int *)num

Constructs an array of the pathnames of all the open worksheets, placing a pointer to this array in sheets and the number of sheets in num. Worksheets with the name ÒUntitledÓ have never been saved.

Note: When you are done with the array returned in sheets, you must free (using free()) each string contained in the array, as well as the array itself, in order to avoid memory leaks.

MOLIVersion

+ (int)MOLIVersion

Returns the current MOLI version.

Other Methods

Unless otherwise noted, the following methods return an error value if they are not successful. The error values can be found in ÒMOLI ErrorsÓ on pagXII.

Note: The worksheet is not recalculated or redisplayed after information is changed using a MesaObject method. To keep the information on the worksheet current, you must either redisplay or recalculate the sheet after you have changed it.

---

addLabel:forRangeUpperRow:col:lowerRow:col:

- (int)addLabel:(char *)name forRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

Assigns the range specified by the rectangular coordinates ur, uc, lr, and lc to the range label name.

associateLabel:withItems:andValues:offset:numItems: orientation:

- (int)associateLabel:(const char *)name withItems:(char **)items andValues:(MOLIValue *)mv offset:(int)offset numItems:(int)num orientation:(int)orient

Associates a set of string items with a set of values. The label name is a Mesa named range defining where the look up should take place on the worksheet. The items argument is an array of strings containing, for example, stock names. The values in mv are values to be placed at offset rows or columns from the top or left of the matching cell. There are num items in the list and the association will be either horizontalOrientation or verticalOrientation. An example of this is used in the MOLIDemo program to pump stock prices into the Mesa worksheet.

clearColPageBreaks::

- (int)clearColPageBreaks:(int)first :(int)last

  Clears all page breaks in the columns between first and last, inclusive.

clearRowPageBreaks::

- (int)clearRowPageBreaks:(int)first :(int)last

  Clears all page breaks in the rows between first and last, inclusive.

createGraph:at:upperRow:col:lowerRow:col:name:

- (int)createGraph:(int)type at:(NXRect *)rect
  upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc
  name:(char **)name

  Creates a graph of type type at the place in the worksheet specified by rect and returns the name name. The data used for the base range of the graph is specified by ur, uc, lr, and lc. This method can be used in conjunction with getEPSForGraph:in: to generate graphs from supplied data and then to load them into your application. The constants for graph types can be found in the table ÒGraph TypesÓ on pageI I.

delegate

- delegate

  Returns the current delegate for the MesaObject.

displaySheet

- (int)displaySheet

  Redisplays the worksheet.

free

- free

Disconnects the object from its worksheet and frees it.

Note: You must call this method or freeAndClose, or Mesa will continue to keep a link open for this sheet.

freeAndClose

- freeAndClose

Disconnects the object from its worksheet, frees the object, and closes the sheet. This object must be the last link to the worksheet. Views and other objects that are linked to the sheet must be closed first in order to prevent closing a worksheet out from under another object.

getCell::forPoint:inRangeUpperRow:col:lowerRow:col:

- (int)getCell:(int *)row :(int *)col forPoint:(NXPoint *)point inRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

Converts the point point into a cell row and col number based on the range of the view, specified by ur, uc, lr, and lc.

getCell::forPoint:inLabel:

- (int)getCell:(int *)row :(int *)col forPoint:(NXPoint *)point inLabel:(char *)name

Converts the point point into a cell row and col number based on the range of the view, specified by the range label name.

getEPSForLabel:in:

- (int)getEPSForLabel:(char *)name in:(id *)vp

Retrieves an EPS image for the range of cells specified by the label name. The id vp is set to an NXImage object if the message is successful, or NULL if not. You can use composite:toPoint: to place the image into a view in your custom application.

Note: You must send a free message to the NXImage object when you are done with it, in order to avoid memory leaks.

getEPSForUpperRow:col:lowerRow:col:into:

- (int)getEPSForUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc into:(id *)vp

Retrieves an EPS image for the range of cells specified by ur, uc, lr, and lc. The id vp is set to an NXImage object if the message is successful or NULL if not. You can use composite:toPoint: to place the image into a view in your custom application.

Note: You must send a free message to the NXImage object when you are done with it, in order to avoid memory leaks.

getEPSForGraph:in:

- (int)getEPSForGraph:(char *)name in:(id *)vp

Retrieves an EPS image for the graph specified by the range label name. The id vp is set to an NXImage object if the message is successful or NULL if not. You can use composite:toPoint: to place the image into a view in your custom application.

Note: You must send a free message to the NXImage object when you are done with it, in order to avoid memory leaks.

getEventHeader:into:

- (int)getEventHeader:(char *)name into:(char **)header

Retrieves the event header for the AddIn name and places it in header.

Note: You must free (using free()) the string returned in header when you are done with it, in order to avoid memory leaks.

getEventHeaders:

- (int)getEventHeaders:(char **)headers

Retrieves the event header information for all AddIns and places it in headers.

Note: You must free (using free()) the string returned in headers when you are done with it, in order to avoid memory leaks.

getGraph:element:color:

- (int)getGraph:(char *)name element:(int)n color:(NXColor *)color

Retrieves the color of element n of the graph named by name and places it in color.

getGraph:item:color:

- (int)getGraph:(char *)name item:(int)item color:(NXColor *)color

Retrieves the color of item item in the graph named by name. For a table of graph item constants, see ÒGraph Item Constants with ColorÓ on pageII.

getGraph:item:doubleValue:

- (int)getGraph:(char *)name item:(int)item doubleValue:(double *)value

Places into value the double value of item item from the graph named by name. For a table of graph item constants, see ÒGraph Item Constants with Double ValuesÓ on pageIII.

getGraph:item:font:

- (int)getGraph:(char *)name item:(int)item font:(id *)font

Places into font the font of item item on the graph named by name. For a table of graph item constants, see ÒGraph Item Constants with FontsÓ on pageIV.

getGraph:item:intValue:

- (int)getGraph:(char *)name item:(int)item intValue:(int *)value

Places into value the integer value of the item item from the graph named by name. For a table of graph item constants, see ÒGraph Item Constants with Integer ValuesÓ on pageV.

getGraph:item:stringValue:

- (int)getGraph:(char *)name item:(int)item stringValue:(char **)value

Places into value the string value of the item item from the graph named by name. For a table of graph item constants, see ÒGraph Item Constants with String ValuesÓ on pageIII.

getGraph:item:upperRow:col:lowerRow:col:

```
- (int)getGraph:(char *)name item:(int)item
    upperRow:(int *)ur col:(int *)uc
    lowerRow:(int *)lr col:(int *)lc
```

Retrieves the range reference for the item item from the graph name and places its coordinates in ur, uc, lr, and lc. For a table of graph item constants, see ÒGraph Item Constants with Range ReferencesÓ on pageIV.

getGraphs:num:

```
- (int)getGraphs:(char ***)graphlist num:(int *)num
```

Constructs an array of the named graphs, placing a pointer to this array in graphlist and the number of graphs in num.

Note: When you are done with the array returned in graphlist, you must free (using free()) each string contained in the array, as well as the array itself, in order to avoid memory leaks.

getInput:forCell::

```
- (int)getInput:(char **)string forCell:(int)row :(int)col
```

Places in string the user-typed input string for the cell at row row and column col.

Note: You must free (using free()) the string returned in string when you are done with it, in order to avoid memory leaks.

getLabels:ranges:num:

```
- (int)getLabels:(char ***)labels ranges:(short **)ranges
    num:(int *)num
```

Constructs an array of the names and ranges of the labels for the worksheet, placing a pointer to the array of names in labels, a pointer to the array of ranges in ranges, and the number of labels in num. The array of ranges contains four entries for each label, for the upper row, left column, lower row, and right column of the range. An example follows:

```
char **labels;
short *ranges;
int num,x;

if ([myobject getLabels:&label ranges:&ranges num:&num] ==
    MOLISuccess) {
    for (x = 0; x < num; x++) {
        // print the info
        printf("%d. %s upperRow:%d col:%d lowerRow:%d col:%d\n",
            x + 1,labels[x],ranges[x * 4],ranges[x * 4 + 1],
            ranges[x * 4 + 2], ranges[x * 4 + 3]);

        // free the string
        free(labels[x]);
        }

    // free the arrays
    free(labels);
    free(ranges);
}
```

Note: When you are done with the arrays returned in labels and ranges, you must free (using free()) each of the strings contained in the array returned in labels, as well as the arrays returned in both labels and ranges, in order to avoid memory leaks.

getMaxRows:andColumns:

- (int)getMaxRows:(int *)rows andColumns:(int *)cols

Retrieves the maximum dimensions of the worksheet, placing the width in rows and the height in cols.

getQueryNames:num:

- (int)getQueryNames:(char ***)names num:(int *)num

Constructs an array of the names of the named queries contained in the worksheet, placing a pointer to this array in names and the number of queries in num. An example follows:

```
char **queries;
int num,x;

if ([myobject getQueryNames:&queries num:&num] == MOLISuccess)
    {
    for (x = 0; x < num; x++) {
        // print the info
        printf("Query %s\n",queries[x]);

        // free the string
        free(queries[x]);
        }

    // free the arrays
    free(queries);
    }
```

Note: When you are done with the array returned in names, you must free (using free()) each string contained in the array, as well as the array itself, in order to avoid memory leaks.

getRangeUpperRow:col:lowerRow:col:forLabel:

- (int)getRangeUpperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr col:(int *)lc forLabel:(char *)label

Retrieves the range coordinates for the given label, placing them in ur, uc, lr, and lc. If label is not found, the method returns MOLILabelNotFoundError. For a list of possible MOLI errors, see ÒMOLI ErrorsÓ on page XII.

getRect:forCell::inRangeUpperRow:col:lowerRow:col:

- (int)getRect:(NXRect *)rect forCell:(int)row :(int)col inRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Retrieves a rectangle rect for a cell specified by row and col, given the range coordinates for the view in ur, uc, lr, and lc.

getRect:forCell::inLabel:

- (int)getRect:(NXRect *)rect forCell:(int)row :(int)col inLabel:(char *)name

  Retrieves a rectangle rect for a cell specified by row and col, given the range for the view in the range label name.

getReportNames:num:

- (int)getReportNames:(char ***)names num:(int *)num

  Constructs an array of the names of the named reports for the worksheet, placing a pointer to this array in names and the number of reports in num. An example follows:

```
char **reports;
int num,x;

if ([myobject getReportNames:&reports num:&num] == MOLISuccess)
    {
    for (x = 0; x < num; x++) {
        // print the info
        printf("Report %s\n",reports[x]);

        // free the string
        free(reports[x]);
        }

    // free the arrays
    free(reports);
    }
```

Note: When you are done with the array returned in names, you must free (using free()) each string contained in the array, as well as the array itself, in order to avoid memory leaks.

getScriptNames:num:

- (int)getScriptNames:(char ***)names num:(int *)num

Constructs an array of the named MScript scripts for the worksheet, placing a pointer to this array in names and the number of scripts in num.

Note: When you are done with the array returned in names, you must free (using free()) each string contained in the array, as well as the array itself, in order to avoid memory leaks.

getValues:num:upperRow:col:lowerRow:col:

- (int)getValues:(MOLIValue **)mv num:(int *)num upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

Retrieves an array of values from the spreadsheet from the range specified by ur, uc, lr, and lc. The number of items will be placed in num and the items will be in an array pointed to by mv. The array that is returned will be sparse (i.e., only cells that exist will be returned). Remember to test the address of each value and the type. Each element in the array is in the following format:

```
enum {stringMOLIValue, numberMOLIValue, errorMOLIValue};

typedef struct _MOLIValue {
    int type;
    short row,col;
    union {
        char *string;
        double number;
        int error;
        } values;
    } MOLIValue;
```

Note: When you are done with the information, call freeMOLIValue(int num, MOLIValue *mv) to free the array.

getValues:forLabel:num:(int *)upperRow:col:lowerRow:col:

- (int)getValues:(MOLIValue **)mv
  forLabel:(const char *)name num:(int *)num
  upperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr
  col:(int *)lc

  Retrieves an array of values from the spreadsheet from the named range name. The total number of items will be placed in num and the items will be in an array pointed to by mv, and the bounds of the range will be returned in ur, uc, lr, and lc. The array that is returned will be sparse (i.e., only cells that exist will be returned). Remember to test the address of each value and the type. Each element in the array is in the following format:

```
enum {stringMOLIValue, numberMOLIValue, errorMOLIValue};

typedef struct _MOLIValue {
    int type;
    short row,col;
    union {
        char *string;
        double number;
        int error;
        } values;
    } MOLIValue;
```

Note: When you are done with the information, call freeMOLIValue(int n,MOLIValue *mv) to free the array.

hideCols::

- (int)hideCols:(int)first :(int)last

  Hides the range of columns specified by first and last. The first column cannot be hidden.

hideRows::

- (int)hideRows:(int)fr :(int)lr

  Hides the range of rows specified by first and last. The first row cannot be hidden.

initToNewWorksheet

- initToNewWorksheet

This method is used instead of the init method. It connects the object to a new (Untitled) worksheet. If Mesa cannot be run or if the worksheet cannot be opened, NULL is returned.

initToWorksheet:

- initToWorksheet:(const char *)sheet

This method is used instead of the init method. It connects the object to a worksheet with the name sheet. The full, absolute pathname must be supplied. If Mesa cannot be run or if the worksheet cannot be opened, NULL is returned.

makeKeyAndOrderFront

- (int)makeKeyAndOrderFront

Brings a window in the worksheet to the front of the window list. If Mesa is not the active application, it becomes the active application.

mesaVersion:

- (int)mesaVersion:(int *)v

Returns the version number of the copy of Mesa that the object is attached to. For example, Mesa 1.2 has version number 121.

miniturize:

- (int)miniturize:(int)win

Miniaturizes the window of the worksheet numbered win. Note the spelling of the method.

performSQLQuery:

- (int)performSQLQuery:(char *)name

Performs the pre-defined SQL query named name. You must first build the query with MesaÕs Query Inspector or with the setQuery:server::password:: user:database:sql:destination:setRange: formatting: method.

printReport:

- (int)printReport:(char *)name

  This method prints the report named name. You must first create the named report. The custom application can place report data into the worksheet, and then print it out using a pre-made report format.

putValues:num:upperRow:col: lowerRow:col:

- (int)putValues:(MOLIValue *)mv num:(int)num upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Takes an array of values and places it in the worksheet. The array of values is mv and there are num of them. They will be placed in the worksheet in the range specified by ur, uc, lr, and lc. Mesa removes all data from that range and then inserts the supplied information. Make sure to set the row and col for each cell. If a cell is not in the range, it will not be inserted.

putValuesAndPreserveFormatting:num:upperRow:col: lowerRow:col:

- (int)putValuesAndPreserveFormatting:(MOLIValue *)mv num:(int)num upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Takes an array of values and places it in the worksheet. The array of values is mv and there are num of them. They will be placed in the worksheet in the range specified by ur, uc, lr, and lc. Mesa changes only the values for the cells, thus preserving the formatting. Make sure to set the row and col for each cell. If a cell is not in the range, it will not be inserted.

recalc

- (int)recalc

  Recalculates and redisplays the worksheet. The worksheet is not recalculated or redisplayed after information is changed using a MesaObject method. To keep the information on the worksheet current, you must either redisplay or recalculate the sheet after you have changed it.

resetRowRangeSize::

- (int)resetRowRangeSize:(int)first :(int)last

  Sets the range of rows specified by first and last to the default height for their style template.

resetColRangeSize::

- (int)resetColRangeSize:(int)first :(int)last

  Sets the range of columns specified by first and last to the default width for their style template.

removeLabel:

- (int)removeLabel:(char *)name

  Removes the label name from the label index, such that the name is no longer associated with its range.

runMScript:

- (int)runMScript:(char *)name

  Runs the MScript script name. The script must be pre-defined in MesaÕs MScript Inspector or with thesetScript:: autoExecute:beforeClose: method.

saveSheet

- (int)saveSheet

  Saves the worksheet to disk using its current file name.

saveSheetAs:

- (int)saveSheetAs:(const char *)name

  Renames the worksheet to name and saves it to disk.

setAutoRecalc:

- (int)setAutoRecalc:(int)state

  Sets whether AutoRecalc is enabled for the worksheet. If state is 1, the sheet will recalculate automatically. If state is 0, the sheet must be recalculated manually.

setBkgColor:

- (int)setBkgColor:(NXColor)color

  Sets the background color of the sheet to color.

setBkgClear:

- (int)setBkgClear:(int)state

  Sets whether the background color of the worksheet is clear. If state is 1, the background of the sheet will be clear, and EPS images of ranges that are pasted into other applications will be clear. If state is 0, the background will be opaque.

setCellHeight:

- (int)setCellHeight:(int)height

  Sets the cell height of the worksheet to height pixels.

setCellWidth:

- (int)setCellWidth:(int)width

  Sets the cell width of the worksheet to width pixels.

setColor:atPoint:inRangeUpperRow:col:lowerRow:col:flags:

- (int)setColor:(NXColor)color atPoint:(NXPoint *)point inRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc flags:(int)flags

  Sets the text color of the cell at point point in the range specified by ur, uc, lr, and lc to color. Pass the flags element of the NXEvent record as flags or 0 if this method is not being called from within an event handling method (e.g., mouseDown:, keyDown:, etc.).

setColor:atPoint:inLabel:flags:

- (int)setColor:(NXColor)color atPoint:(NXPoint *)point inLabel:(char *)name flags:(int)flags

  Sets the text color of the cell at point point in the range specified by the range label name to color. Pass the flags element of the NXEvent record as flags or 0 if this method is not being called from within an event handling method (e.g., mouseDown:, keyDown:, etc.).

setColor:atPoint:inGraph:flags:

- (int)setColor:(NXColor)color atPoint:(NXPoint *)point inGraph:(char *)name flags:(int)flags

  Sets the color of the graph element at point in graph name to color. Pass the flags element of the NXEvent record as flags or 0 if this method is not being called from within an event handling method (e.g., mouseDown:, keyDown:, etc.)

setColPageBreak:

- (int)setColPageBreak:(int)col

  Sets a vertical page break after column number col.

setColRange::toSize:

- (int)setColRange:(int)first :(int)last toSize:(int)width

  Sets the width of the columns specified by first and last to width points.

setDelegate:

- setDelegate:del

  Sets the delegate for the MesaObject to del.

setEventHeaderTo:forAddIn:

- (int)setEventHeaderTo:(char *)string forAddIn:(char *)name

  Sets the event header for the AddIn name to string.

setFont:atPoint:inRangeUpperRow:col:lowerRow:col:

- (int)setFont:font atPoint:(NXPoint *)point inRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

  Sets the font of the cell at point in the range specified by ur, uc, lr, and lc to font.

setFont:atPoint:inLabel:

- (int)setFont:font atPoint:(NXPoint *)point inLabel:(char *)name

  Sets the font of the cell at point in the range specified by the range label name to font.

setFont:atPoint:inGraph:

- (int)setFont:font atPoint:(NXPoint *)point inGraph:(char *)name

  Sets the font of the graph element at point in graph name to font.

setGraph:element:color:

- (int)setGraph:(char *)name element:(int)n color:(NXColor)color

  Sets the color of element n in graph name to color.

setGraph:item:color:

- (int)setGraph:(char *)name item:(int)it color:(NXColor)nc

  Sets the color of item item in graph name to color. For a table of graph item constants, see ÒGraph Item Constants with ColorÓ on pageII.

setGraph:item:doubleValue:

- (int)setGraph:(char *)name item:(int)item doubleValue:(double *)value

  Sets the double value of the item item in graph name to value. For a table of graph item constants, see ÒGraph Item Constants with Double ValuesÓ on pageIII.

setGraph:item:font:

- (int)setGraph:(char *)name item:(int)item font:font

  Sets the font of item item in graph name to font. For a table of graph item constants, see ÒGraph Item Constants with FontsÓ on pageIV.

setGraph:item:intValue:

- (int)setGraph:(char *)name item:(int)item intValue:(int *)value

  Sets the integer value of item item in graph name to value. For a table of graph item constants, see ÒGraph Item Constants with Integer ValuesÓ on pageV.

setGraph:item:stringValue:

- (int)setGraph:(char *)name item:(int)item stringValue:(char *)value

  Sets the string value of item item in graph name to value. Note that the string value should be less than 2000 characters long. For a table of graph item constants, see ÒGraph Item Constants with String ValuesÓ on pageII.

setGraph:item:upperRow:col:lowerRow:col:

- (int)setGraph:(char *)name item:(int)item upperRow:(int *)ur col:(int *)uc lowerRow:(int *)lr col:(int *)lc

  Sets the range reference for item item in graph name to the range specified by ur, uc, lr, and lc. For a table of graph item constants, see ÒGraph Item Constants with Range ReferencesÓ on pageIV.

setGraph:toType:

- (int)setGraph:(char *)name toType:(int)type

  Changes the graph name to type type. The integers for graph types can be found in ÒGraph TypesÓ on pageI.

setGraph:toOrientation:

- (int)setGraph:(char *)name toOrientation:(int)orient

  Changes the data orientation for the graph name. Possible orientations are horizontalOrientation and verticalOrientation.

setGraph:toLegend:

- (int)setGraph:(char *)name toLegend:(int)legend

  Sets the legend location for graph name to legend. The integers for legend types can be found in ÒGraph LegendsÓ on page I.

setGridOn:

- (int)setGridOn:(int)state

  Sets whether the worksheet grid is on or off. If state is 1, the grid is visible. If state is 0, the grid is not visible.

setGridColor:

- (int)setGridColor:(NXColor)color

Sets the color of the worksheet grid to color.

setInput:forCell::

- (int)setInput:(char *)string forCell:(int)row :(int)col

Sets the input string of the given cell to string. The input is parsed based on the input type for the cell and the input string string (i.e., number, date, any, etc.)

setLabel:toFont:

- (int)setLabel:(char *)name toFont:font

Sets the font for all the cells in the named range name to font.

setLabel:toColor:

- (int)setLabel:(char *)name toColor:(NXColor)color

Sets the color for all the cells in the named range name to color.

setLabel:toBkgColor:

- (int)setLabel:(char *)name toBkgColor:(NXColor)color

Sets the background color of all the cells in the named range name to color.

setNumberOfCols:

- (int)setNumberOfCols:(int)cols

Sets the maximum number of columns for the worksheet to cols.

setNumberOfRows:

- (int)setNumberOfRows:(int)rows

Sets the maximum number of rows for the worksheet to rows.

setQuery:server::password::user:database:sql:destination:
setRange:formatting:

- (int)setQuery:(char *)name server:(char *)server :(char *)st password:(char *)pw :(int)pp user:(char *)user database:(char *)db sql:(char *)sql destination:(char *)dest setRange:(char *)range formatting:(int)flags

Creates a query named name and fills in the parameters. The server name is server and st is the name of the database object (blank = SYBASE; otherwise it runs from an AddIn). The flag to prompt for a password is pp. The raw query string is sql. The destination range is dest Ñ send a zero length string if you do not expect return data. The range name to set with the extent of the query is range. The flags argument contains various different flags OR'ed together. For a table of query flags, see ÒQuery FlagsÓ on page X.

setRangeUpperRow:col:lowerRow:col:toFont:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toFont:font

Sets the font for all the cells in the range specified by ur, uc, lr, and lc to font.

setRangeUpperRow:col:lowerRow:col:toColor:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toColor:(NXColor)color

Sets the text color for all the cells in the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toBkgColor:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBkgColor:(NXColor)color

Sets the background color for all the cells in the range specified by ur, uc, lr, and lc to color.

setRangeUpperRow:col:lowerRow:col:toBorder:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBorder:(int)border

Sets the range specified by ur, uc, lr, and lc to a given border type. The border in is made up by ORÕing the following constants:

¥ CHANGETOPBORDER

¥ SETTOPBORDER

¥ CHANGEBOTTOMBORDER

¥ SETBOTTOMBORDER

¥ CHANGELEFTBORDER

¥ SETLEFTBORDER

¥ CHANGERIGHTBORDER

¥ SETRIGHTBORDER

If the "CHANGE" bit is true, the border is either set or cleared depending on whether the "SET" bit is true. For example, to clear the border around a cell, the value would be:

```
CHANGETOPBORDER|CHANGEBOTTOMBORDER|CHANGELE
    FTBORDER|CHANGERIGHTBORDER
```

To put a border under a set of cells:

```
CHANGEBOTTOMBORDER|SETBOTTOMBORDER
```

setRangeUpperRow:col:lowerRow:col:toBorderThickness:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBorderThickness:(int)thick

Sets the border thickness for all the cells in the range specified by ur, uc, lr, and lc to the thickness thick. The constants for border thickness can be found in ÒBordersÓ on page IX.

setRangeUpperRow:col:lowerRow:col:toBorderColor:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toBorderColor:(NXColor)color

Sets the border color for all the cells in the range specified by ur, uc, lr, and lc to color.

setGraph:toRect:

- (int)setGraph:(char *)name toRect:(NXRect *)rect

  Places the graph name in a new rectangle rect on the worksheet.

setRangeUpperRow:col:lowerRow:col:toDisplay:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toDisplay:(int)display

  Sets the display type for the range of cells specified by ur, uc, lr, and lc to display. The constants for display type can be found in ÒDisplay FormatsÓ on pageVIII.

setRangeUpperRow:col:lowerRow:col:toPrecision:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toPrecision:(int)prec

  Sets the precision for a range of cells specified by ur, uc, lr, and lc to prec.

setRangeUpperRow:col:lowerRow:col:toAlignment:

- (int)setRangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc toAlignment:(int)align

  Sets the alignment for a range of cells specified by ur, uc, lr, and lc to align. The constants for alignment type can be found in ÒAlignmentsÓ on pageIX.

setRowPageBreak:

- (int)setRowPageBreak:(int)row

  Sets a horizontal page break after row number row.

setRowRange::toSize:

- (int)setRowRange:(int)first :(int)last toSize:(int)height

  Sets the height of the rows specified by first and last to height points.

setScript::autoExecute:beforeClose:

- (int)setScript:(char *)name :(char *)string autoExecute:(int)auto beforeClose:(int)close

  Sets script name to the string string and sets the autoexecute flag to auto and the Run Before Close flag to close. Remember to include line feeds at the end of each line in the script.

setSheetChanged:

- (int)setSheetChanged:(BOOL)change

  Sets the sheet-changed flag. It should be issued with a change value of TRUE after any command that changes the worksheet.

Delegate Methods:

sheetDidRecalc

- sheetDidRecalc:sender

  The sheetDidRecalc: method is sent to the MesaObject's delegate after each recalculation of the worksheet.

sheetDidRedisplay:

- sheetDidRedisplay:sender

  This message is sent to the delegate each time the worksheet is redisplayed.

showCols::

- (int)showCols:(int)first :(int)last

  Unhides the range of columns specified by first and last.

showRows::

- (int)showRows:(int)first :(int)last

  Unhides the range of rows specified by first and last.

smartSizeColRange::

- (int)smartSizeColRange:(int)first :(int)last

  Smartsizes the range of columns specified by first and last.

smartSizeRowRange::

- (int)smartSizeRowRange:(int)first :(int)last

  Smartsizes the range of rows specified by first and last.

sortARangeUpperRow:col:lowerRow:col:by:key1::key2:: key3::hasTitles:

- (int)sortARangeUpperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc by:(int)rows key1:(int)k1 :(int)d1 key2:(int)k2 :(int)d2 key3:(int)k3 :(int)d3 hasTitles:(int)hasTitles

  Sorts the specified range by rows if rows is 1 or by columns if rows is 0. The data is sorted by key1, key2, and key3, in descending or ascending order according to d1, d2, and d3. If hasTitles is TRUE, the first row or column (as appropriate) is not sorted.

MesaView

The MesaView class allows you to connect to both ranges of cells and graphs, and have them display themselves in your custom application. These views can be edited just as though the views were a part of Mesa, this includes Drag and Drop of colors and fonts. Graphs can also be created for a MesaView through the use of a MesaObject.

Methods

acceptColor:atPoint:

- acceptColor:(NXColor)color atPoint:(NXPoint *)point

  This is the implementation for accepting the color color at the point point. In general, this method will not need to be overridden.

acceptFont:atPoint:

- acceptFont:font atPoint:(NXPoint *)point

This is the implementation for accepting the font font dropped at the point point. In general, this method will not need to be overridden. You will need to implement an extension to the standard NEXTSTEP font well such that this method is supported.

canEdit

- (BOOL)canEdit

Returns whether or not the current view can be edited. The default is FALSE.

free

- free

When you are finished with a MesaView, it must be freed. Failure to free memory allocated will cause memory leaks, and MesaÕs connection to this view will remain open.

initFrame:toSheet:upperRow:col:lowerRow:col:(int)

- initFrame:(NXRect *)rect toSheet:(char *)sheet upperRow:(int)ur col:(int)uc lowerRow:(int)lr col:(int)lc

This method creates a non-editable view rect into the worksheet sheet for the coordinates ur, uc, lr, and lc. The method call will look like:

```
myDataView = [[myView alloc] initFrame:&re toSheet:"~/files/
   foo.Mesa" upperRow: 0 col: 0 lowerRow:10 col:12]; With
   myView a subclass of MesaView.
```

initFrame:toSheet:toLabel:

- initFrame:(NXRect *)rect toSheet:(char *)sheet toLabel:(char *)name

This method creates a non-editable view rect into the worksheet sheet for the range label name. The method call will look like:

```
myDataView = [[myView alloc] initFrame:&re toSheet:"~/files/
   foo.Mesa" toLabel:"myData"];  With myView a subclass of
   MesaView.
```

initFrame:toSheet:toGraph:

- initFrame:(NXRect *)rect toSheet:(char *)sheet toGraph:(char *)name

  This method creates a non-editable view rect into the worksheet sheet for the graph name. The method call will look like:

```
myGraphView = [[myView alloc] initFrame:&re toSheet:"~/files/
    foo.Mesa" toGraph:"Graph 1"];  With myView a subclass of
    MesaView;
```

setCanEdit:

- setCanEdit:(BOOL)state

  Sets whether the current view can be edited. If state is TRUE, the view will be editable.

Delegate Methods:

sheetDidRedisplay:

- sheetDidRedisplay:sender

  This message is sent to the delegate each time the sheet is redisplayed.

4. Appendix A

MOLI and AddIn Constants

Graph Constants

Graph Legends

Legend Location	Constant
No Legend	noMesaLegend
Left Legend	leftMesaLegend
Right Legend	rightMesaLegend
Top Legend	topMesaLegend
Bottom Legend	bottomMesaLegend

Graph Types

Graph Type	Constant	Graph Type	Constant
Scatter Chart	scatterMesaGraphType	Column Chart	columnMesaGraphType
Bar Chart	barMesaGraphType	Pie Chart	pieMesaGraphType
Stacked Col-umn Chart	sColumnMesaGraphType	Stacked Bar Chart	sBarMesaGraphType
Area Chart	areaMesaGraphType	High/Low/ Close Chart	HLCMesaGraphType
High/Low Chart	HLMesaGraphType	XY Chart	XYMesaGraphType
3D Bar Char	d3BarMesaGraphType	Ribbon Chart	ribbonMesaGraphType
Pyramid Chart	pyramidMesaGraphType	3D Area Chart	d3AreaMesaGraphType

Graph Item Constants with Color

Item	Constant
background color	bkgColorGraphItem
border color	borderColorGraphItem
legend color	legendColorGraphItem
3D color	threeDColorGraphItem
base color	baseColorGraphItem

Graph Item Constants with Double Values

Items	Constants
X-axis label rotation	xLabelRotGraphItem
Y-axis label rotation	yLabelRotGraphItem
X-axis maximum	xMaxGraphItem
X-axis minimum	xMinGraphItem
Y-axis maximum	yMaxGraphItem
Y-axis minimum	yMinGraphItem
3D view angle, X direction	xRotGraphItem
3D view angle, Y direction	yRotGraphItem

Graph Item Constants with String Values

Items	Constants
X-axis title	xTitleGraphItem
Y-axis title	yTitleGraphItem
First title line	þrstTitleGraphItem
Second title line	secondTitleGraphItem
Top title range	topTitleGraphItem
Side title range	sideTitleGraphItem

Graph Item Constants with Fonts

Items	Constants
X-axis title	xTitleGraphItem
Y-axis title	yTitleGraphItem
First title line	þrstTitleGraphItem
Second title line	secondTitleGraphItem
X-axis labels	xLabelGraphItem
Y-axis labels	yLabelGraphItem
Legend	legendGraphItem

Graph Item Constants with Range References

Items	Constants
Data range	baseRangeGraphItem
Top title range	topTitleGraphItem
Side title range	sideTitleGraphItem
Extra ranges	1-6

Graph Item Constants with Integer Values

Item	Constant	Value
Background is clear	hasClearBkgGraphItem	0 or 1
Width of border in points	borderWidthGraphItem	0 to 20
Legend is clear	hasClearLegendGraphItem	0 or 1
3D scales are clear	hasClear3DGraphItem	0 or 1
Label positions	labelPositionGraphItem	centerGraphLabelPosition or groupGraphLabelPosition
Labels in top row	topRowLabelsGraphItem	0 or 1
Labels in left column	leftColumnLabelsGraphItem	0 or 1
Display format for labels	displayFormatGraphItem	see the enumerated display formats, e.g. generalMesaFormat
Display format decimal precision	precisionGraphItem	0 to 15
X-axis title is a cell reference	xTitleIsCellRefGraphItem	0 or 1
Y-axis title is a cell reference	yTitleIsCellRefGraphItem	0 or 1
First line of title is a cell reference	þrstTitleIsCellRefGraphItem	0 or 1

Graph Item Constants with Integer Values

Item	Constant	Value
Second line of title is a cell reference	secondTitleIsCellRefGraphItem	0 or 1
Orientation of data	orientationGraphItem	horizontalOrientation or verticalOrientation
Y-axis major tick type	yMajTickGraphItem	Tick type constants: strikeThroughGraphTickMark, innerGraphTickMark, or outerGraphTickMark
Y-axis major tick size	yMajTickSizeGraphItem	any
X-axis major tick type	xMajTickGraphItem	Tick type constants
X-axis major tick size	xMajTickSizeGraphItem	any
Y-axis minor tick type	yMinTickGraphItem	Tick type constants
Y-axis minor tick size	yMinTickSizeGraphItem	any
X-axis minor tick type	xMinTickGraphItem	Tick type constants
X-axis minor tick size	xMinTickSizeGraphItem	any
Auto-scaling	autoScaleGraphItem	0 or 1
Logarithmic scales	logScaleGraphItem	0 or 1
X-axis major grid lines	xMajHairGraphItem	0 or 1

Graph Item Constants with Integer Values

Item	Constant	Value
X-axis minor grid lines	xMinHairGraphItem	0 or 1
Y-axis major grid lines	yMajHairGraphItem	0 or 1
Y-axis minor grid lines	yMinHairGraphItem	0 or 1
X-axis border	hasXBorderGraphItem	0 or 1
Y-axis border	hasYBorderGraphItem	0 or 1
Bar column sizes	barColumnSizeGraphItem	0 to 100
Bar column lines	hasBarColumnLineGraphItem	0 or 1
Bar column line width	barColumnLineWidthGraphItem	1 to 20
Scatter graph type	scatterGraphTypeGraphItem	scatterGraphPointsOnly, scatterGraphPointsAndLine, scatterGraphLineOnly
Single X range for X-Y plot	singleXRangeGraphItem	0 or 1
Candlestick High/Low/ Close graph	candleStickGraphItem	0 or 1
Wire frame	wireFrameGraphItem	0 or 1
Legend location	legendLocationGraphItem	noMesaLegend, leftMesaLegend, rightMesaLegend, topMesaLegend, or bottomMesaLegend

Display Formats

Display Format	Constant
General	generalMesaFormat
Fixed decimal	ÞxedDecimalMesaFormat
ScientiÞc	scientiÞcMesaFormat
Currency	currencyMesaFormat
Comma	commaMesaFormat
Chart	chartMesaFormat
Percent	percentMesaFormat
Text	textMesaFormat
Hidden	hiddenMesaFormat
Day-Month-Year	dayMonthYearMesaFormat
Day-Month	dayMonthMesaFormat
Month-Year	monthYearMesaFormat
12-Hour Hour:Minutes:Seconds	dateHourMinSecMesaFormat
12-Hour Hour:Minutes	dateHourMinMesaFormat
Day/Month/Year	dateIntl1MesaFormat
Date/Month	dateIntl2MesaFormat
24-Hour Hour:Minutes:Seconds	timeIntl1MesaFormat
24-Hour Hour:Minutes	timeIntl2MesaFormat

Borders

Thickness	Constant
No Border	noMesaBorder
Thin Border	thinMesaBorder
Thick Border	thickMesaBorder

Alignments

Alignment	Constant
left	leftMesaAlign
right	rightMesaAlign
center	centerMesaAlign
smart	smartMesaAlign
þll	þllMesaAlign

Input Type

Input Type	Constant
Any	anyMesaMode
Formula	formulaMesaMode
Number	numberMesaMode
String	stringMesaMode
Date	dateMesaMode
Unprotected	unprotectedMesaMode

Query Flags

Flag	Constant
Include column titles	includeColumnTitlesQueryInfo
Display errors in dialog	displayErrorsInDialogQueryInfo
Place errors in range	placeErrorsInRangeQueryInfo
Ignore computed rows	ignoreComputedRowsQueryInfo
Run before recalc	runBeforeRecalcQueryInfo
Set style	setStyleQueryInfo
Page break after computed rows	breakAfterComputedRowsQueryInfo
Set height of rows	setHeightOfRowsQueryInfo

FunctionAddIn Errors

Constant	Meaning
noError	The operation was successful.
badInputError	Data was entered incorrectly into a cell.
badFormulaError	A bad or incorrect number of parameters were supplied to a function.
stackUnderßowError	You attempted to pop the stack when it was empty.
labelNotFoundError	The named range label was not found.
circularError	The formula in the cell refers to cells that refer to the cell.
divideByZeroError	Divide by zero.
mathError	The result of a math function is undeÞned.
naError	Generated using the @NA constant.
rangeError	A parameter to a function is out of range.
badAddressError	The address string supplied to the @@() function was incorrect.
errorError	The function or macro resulted in an error.
inherit1Error	The SAME() function was nested more than 64 deep.
inherit2Error	The SAME() function refers to a cell that doesnÕt contain a formula.
lastError	This constantÕs value is the number of errors.

MOLI Errors

Constant	Meaning
MOLISuccess	The operation was successful.
MOLISendError	The message could not be sent to Mesa. The application has probably terminated.
MOLIReceiveError	No answer was received in 30 seconds. The application may have crashed or is very busy.
MOLILabelNotFoundError	The named range was not found in the range look up table.
MOLISheetClosedError	The sheet was closed and the link should be shut down.
MOLINotImplError	This method is currently not implemented. This error should not be returned. It is used for internal debugging purposes only.
MOLIGraphNotFoundError	The named graph was not found in the worksheet's list of graphs.
MOLIOutOfRangeError	The range of cells cannot be converted to EPS because they exist outside of the current view.
MOLIReportNotFound	The named report is not in the worksheet's list of reports.

MOLI Data Types

Constant	Type
stringMOLIValue	String value
numberMOLIValue	Number value
errorMOLIValue	Error value