Professional Programming Environment for NOS/VE

CONTROL DATA

Usage

60486613

Professional Programming Environment for NOS/VE

Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60486613

Manual History

Revision	System Version/ PSR Level	Product Version	Date
A	1.2.2/678	1.0	June 1987
B	1.3.1/700	1.1	April 1988

This is revision B. It documents the Professional Programming Environment product with NOS/VE release 1.3.1 in April 1988.

This revision documents PPE support of the CYBIL and FORTRAN Version 2 (VECTOR_FORTRAN) compilers.

This revision documents support of the user's editor prolog. Users can now specify an editor prolog on the Tailor Options screen to define PPE editor function keys.

Minor technical and editorial changes were also made.

Contents

Contents

About This Manual

About This Manual

This manual describes the Professional Programming Environment (PPE), a software development tool available under the CONTROL DATA® Network Operating System/Virtual Environment (NOS/VE) executing on a CDC® CYBER 180 computer.

While using PPE, the user can execute several other NOS/VE development tools, including the Full-Screen Editor (FSE), the Source Code Utility (SCU), the Full-Screen Debug utility, and the object library generator. These tools are fully described in other manuals listed in appendix B; the means of accessing these tools within a PPE session is described in this manual.

Audience

This manual is written for use by all PPE users. The primary user is expected to be a programmer working as part of a multi-person programming project that is developing a product for use under the NOS/VE operating system. The product under development must be written in one of the languages supported by PPE. PPE supports the NOS/VE CYBIL, COBOL, FORTRAN Version 1, and FORTRAN Version 2 compilers.

You will find PPE easiest to use if you are familiar with the Full-Screen Editor (FSE), the Full-Screen Debug interface, the Source Code Utility (SCU), and the Programming Environment. However, the object-oriented user interface and context-sensitive online help make PPE easy to learn even if you have little NOS/VE experience.

Organization

This manual is organized into four chapters. The first two chapters introduce you to PPE concepts and help you complete your first PPE session. The third chapter gives step-by-step procedures for performing the primary PPE operations. The fourth chapter provides two quick-reference listings, the first for PPE screens and the second for PPE commands.

Appendixes provide a glossary, a list of related manuals, the ASCII character set table, a description of the PPE catalog structure, a summary of the PPE function key labels, and suggestions for using PPE to create and maintain software.

Conventions

blue	Within examples of interactive sessions, user input is shown in blue print. System output is shown in black print.
UPPERCASE	In PPE command syntax, uppercase indicates a statement keyword or character that must be written as shown. For purposes of examples, however, lowercase is used.
lowercase	In PPE command syntax, lowercase indicates a name, number, symbol, or entity that you must supply.
Initial Caps	All screen names are initial capped.
numbers	All numbers are assumed to be base 10 unless otherwise noted.
Quick Reference format	The PPE commands and screens are described in Quick Reference format. The purpose, format, and special remarks of each command or function are described. Also, in most cases, an example is demonstrated.

Submitting Comments

The last page of this manual is a comment sheet. Please tell us about any errors you find in this manual and any problems you have using it.

If the comment sheet in this manual has been used, please send your comments to us at this address:

Control Data Corporation
Technology and Publications Division
P.O. Box 3492
Sunnyvale, California 94088-3492

Please include the following information with your comments:

The manual title and publication number (PPE Usage, 60486613) and the revision letter from the page footer.

Your system's PSR level (if you know it).

Your name, your company's name and address, your work phone number, and whether you want a reply.

Also, if you have access to SOLVER, the CDC online facility for reporting problems, you can use it to submit comments about this manual. When it prompts you for a product identifier for your report, please specify SW8.

In Case of Trouble

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help beyond that provided in the documentation or find that the product does not perform as described, call us at one of the following numbers and a support analyst will work with you.

From the USA and Canada: (800) 345-9903

From other countries: (612) 851-4131

The preceding numbers are for help on product usage. Address questions about the physical packaging and/or distribution of printed manuals to Literature and Distribution Services at the following address:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

or you can call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

Introduction 1

Introduction 1

The Professional Programming Environment (PPE) is a software development tool available under the NOS/VE operating system executing on a CYBER 180 computer.

PPE uses a full-screen, object-oriented interface. The user can select objects on the screen by positioning the cursor on the object. Actions are selected by pressing function keys or by entering SCL commands on the home line of the screen. (Full-screen terminal definition is described in chapter 2.)

PPE Capabilities

This section lists the primary PPE capabilities. The concepts underlying these capabilities are described later in this chapter.

As a programming environment, PPE integrates the programming tasks, including:

- Editing source text.
- Compiling source text.
- Debugging source text.
- Executing object code.

In addition, PPE can coordinate the activities of a multi-person programming project. To do so, it provides these capabilities:

- Full-screen interface to Source Code Utility (SCU) deck and modification creation.
- Extraction and transmittal of SCU decks and modifications within a source library hierarchy. It enforces interlocks to ensure that only one copy of a deck can be changed.
- Expansion and compilation of the product source, including copying decks from higher levels of the hierarchy when a deck is not present at the lowest level.
- Tracing of compilation errors to the decks containing the source.
- Maintenance of an object library at each level of the hierarchy. Each object library contains the compiled code for the source decks at that level.

- Partial builds of the product, expanding and compiling only those source decks that have changed.
- Execution of the product version at the current level of the hierarchy, using object modules at higher levels as needed.

PPE Limitations

PPE use has the following limitations:

- All code for the product being developed must be written in one language.
- The programming languages supported are NOS/VE FORTRAN Version 1 and NOS/VE COBOL.
- PPE does not support SCU features or groups, nor does it provide a method of changing modification or deck header information. To assign features and decks to groups (or change other header information), you can enter SCU subcommands on the home line or use SCU directly. For more information, see the NOS/VE Source Code Management Usage manual.
- PPE does not provide a method of using SCU selection criteria files.

Background Concepts

This section discusses concepts that you should understand before using PPE. Read this section if you are not already familiar with these topics:

- NOS/VE Source Code Utility (SCU)
- PPE catalog hierarchy
- Build process

NOS/VE Source Code Utility (SCU)

The Source Code Utility (SCU) is the NOS/VE utility PPE uses to maintain source text on source libraries. PPE provides a full-screen interface for creating and selecting decks and modifications. For more information, see the NOS/VE Source Code Management manual.

The following paragraphs give a brief description of SCU decks and modifications.

SCU Decks

SCU manipulates source text as units called decks. Each SCU source library contains a collection of decks. Each deck contains a sequence of source lines. A deck can be referenced by name. Source text is edited one deck at a time.

To compile selected decks, PPE calls SCU to produce the input file for the build processor (the compiler). SCU expands the selected decks. The expansion process includes processing any SCU directives embedded in the deck and then writing the deck text to a file for compilation.

Two SCU directives frequently embedded in decks are the copy directives, *COPY and *COPYC. When a copy directive is processed, the text from the deck named on the directive is copied into the deck text at the point where the copy directive is embedded. The *COPY directive copies unconditionally; the *COPYC directive copies only if the deck has not been copied into the deck before.

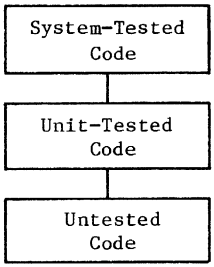
Each deck has an expand attribute. If the attribute is set to TRUE, the deck can be selected for expansion; if the attribute is set to FALSE, the deck can be expanded only when it is copied into another deck.

SCU Modifications

SCU maintains a history of changes made to the source text in a source library. It does so by assigning each change to the modification in effect when the change is made. This includes line additions, replacements, and deletions.

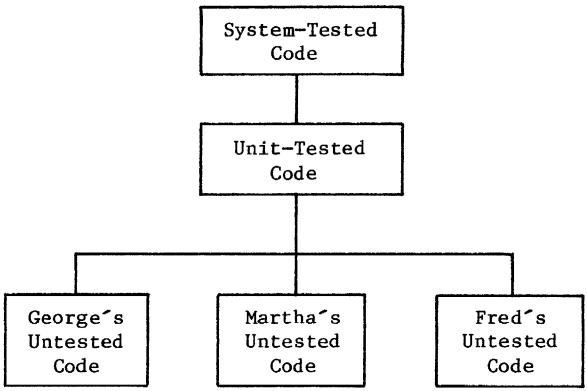
PPE Catalog Hierarchy

To help coordinate multi-person programming projects, PPE supports an environment which maintains several versions of the product. Each version contains product code at a certain state of development, such as untested, unit-tested, and system-tested. These product versions form a hierarchy; code progresses up the hierarchy as it passes stages of evaluation.



PPE implements this hierarchy of levels as a hierarchy of NOS/VE file catalogs. Each catalog contains all files applicable to its product version, including a source library, an object library, and various listing and data files. (The files are listed in appendix D.)

When a person joins a project that uses PPE, he links his PPE catalog to the PPE catalog hierarchy of the project. In this way his catalog becomes a catalog at the lowest level of the project hierarchy.



Extracting Decks

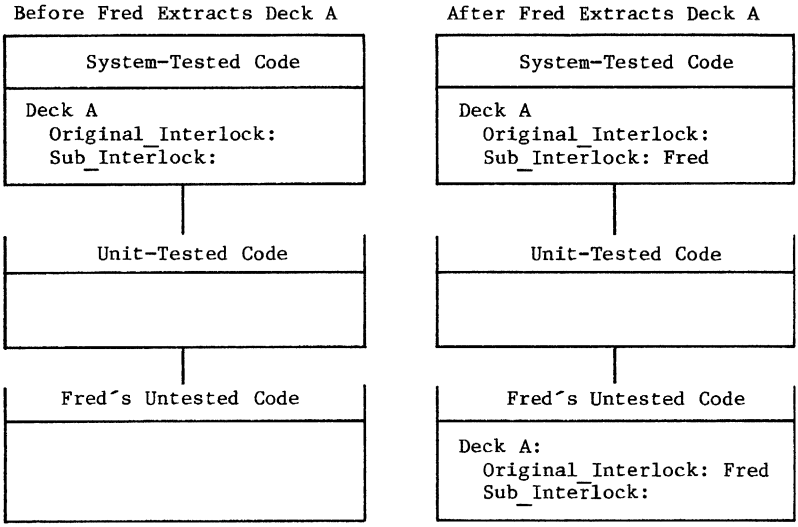
PPE users can create new decks in their PPE source libraries or extract existing decks from higher levels of the hierarchy. PPE extracts a deck by these steps:

1. It searches for the deck, starting at the next higher level and going up the hierarchy until it finds the deck.

2. It then makes a copy of the deck and adds it to the source library at the lowest level of the hierarchy. It does not extract the deck to any intermediate levels of the hierarchy.

An interlock is set on the deck when it is extracted. This prevents other users from extracting the deck until after it has been transmitted. When an interlock is set, the user's identification and the date and time of the interlock are stored in two deck header fields: the sub_interlock field in the higher-level source library copy and the original_interlock field in the lower-level source library copy.

The following illustration shows the interlocks before and after Fred extracts deck A:



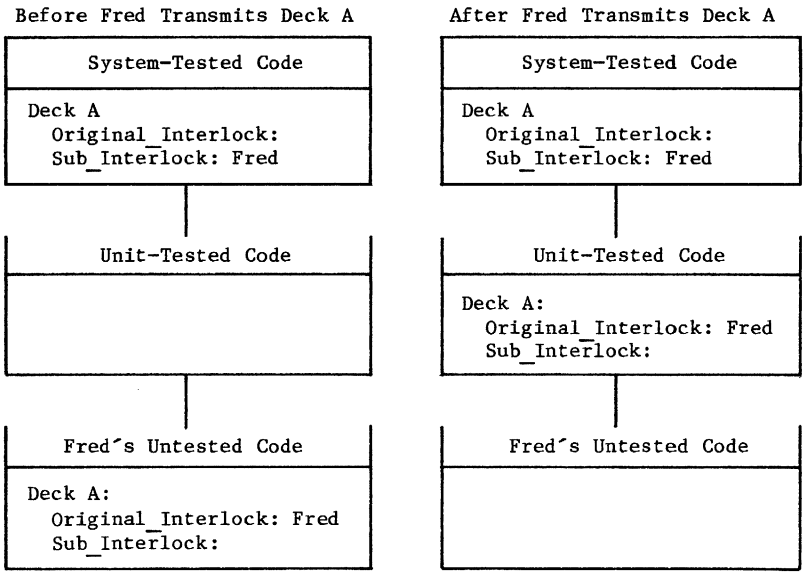
Transmitting Decks

To transmit a deck means to move the deck from the lowest level where it currently resides to the next higher level. You can only transmit decks from the your level of the environment catalog hierarchy to the source library at the next higher level.

When PPE transmits a deck, it copies the deck from the lowest level and combines the copy with the source library at the next higher level. If a copy of the deck already exists at the higher level, it is replaced by the lowest-level copy if deck interlocks match. If the deck at the next level does not have a matching interlock value, the transmit fails. After a successful transmittal, the deck is deleted from the lowest-level source library.

When a transmitted deck has the same name as a deck already existing in the higher-level source library, it must be a copy of the deck extracted from the higher-level source library. The deck interlock is enforced; this means that the original_interlock value in the lower-level deck must match the sub_interlock value in the higher-level deck; otherwise, the lower-level deck copy is not allowed to replace the higher-level copy and the transmittal fails. If the interlock values match, the transmittal succeeds; the interlock is cleared and the lower-level deck copy is deleted.

The following illustrates transmitting decks. When Fred finishes his changes to deck A, he transmits it to the Unit-Tested level:

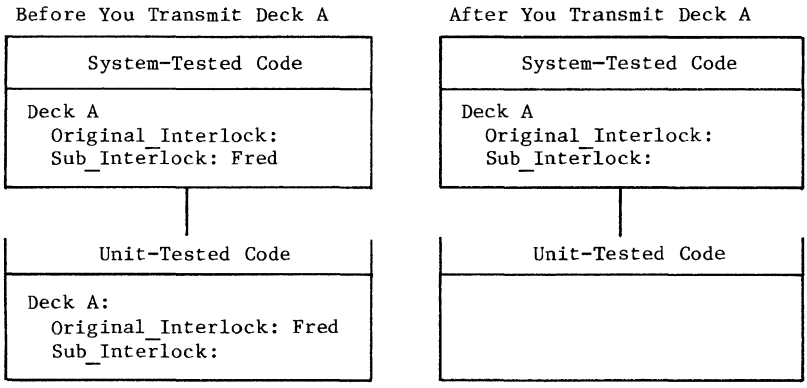


The diagram above shows the PPE hierarchy as you would see it if you executed PPE for the catalog containing Fred's Untested Code. The view has three levels with Fred's Untested Code at the bottom. The hierarchy view presented by PPE always has the catalog for which PPE is executed as the bottom level of the hierarchy.

If you executed PPE specifying the catalog containing the Unit-Tested Code from the diagram above, you would see only two hierarchy levels: the System-Tested Code and the Unit-Tested Code levels. PPE provides no information about hierarchy levels below the level for which it is executed.

If you executed PPE for the Unit-Tested Code level, the hierarchy would appear as shown below.

To transmit deck A from the Unit-Tested Code level to the System-Tested Code level, you must execute PPE from the Unit-Tested Code level. You can only transmit a deck from your level, and it can only be transmitted to the next higher level.



The transmittal checks that the interlock fields match. Because they do, the transmittal takes place; the Unit-Tested Deck A copy replaces the System-Tested Deck A copy, the Deck A interlock is cleared, and the Deck A copy is deleted from the Unit-Tested Code level.

Build Process

The build process transforms source code into executable object code. The build process has two steps: source text expansion and compilation. These steps are described in this section.

At completion of the build process, you are shown a build report. If the build failed, the report describes the errors. If the build succeeded, the report lists the output files available for examination.

Although you can initiate the PPE build process by pressing a single function key, you can, before doing so, provide additional input for each step of the process. The following paragraphs describe the build steps in more detail and discuss the additional input you can provide.

Source Text Expansion

SCU expands the source text in the decks to produce the input file for the build processor.

As described earlier under SCU Decks, part of the expansion process involves copying other decks into the decks being expanded. SCU searches for each deck to be copied in the following locations in the order shown:

1. In the PPE source library at the current level.

2. In the PPE source library at each higher level upwards.

3. In the source libraries listed in the alternate base library list in the order the library files are listed.

A single alternate base library list is used for all builds at a level. You can change the contents of the list at any time.

Compilation

The second step in the build process is the processing of the expanded text by the build processor. Currently, this is always compilation by a compiler. The compiler generates an object module for each compilation unit. The object modules are stored in the object library for the current PPE level.

You can select the parameter values used by the build processor. PPE maintains a library of parameter lists. For example, FORTRAN parameter lists are available for high and low optimization and debug mode. You can select the parameter list used and, if desired, change parameter values.

For more information about compiler parameter values, you can use the HELP command to take you to the parameter discussion in the compiler online manual.

Product Execution

After the product has been successfully built, it is available for execution testing. You execute the product by pressing the function key or entering the command RUN. (Or, you can tailor PPE so it automatically executes the product at the end of each successful build.)

PPE allows you to specify the parameter list passed to the product when it is executed. If the product requires access to one or more attached data files, you can attach those files by entering the SCL command ATTACH_FILE on the home line.

You can designate one of two purposes for a run: normal or interactive debug. An interactive debug run executes the product under control of the Full-Screen Debug utility.

Execution begins at the default starting procedure (the last transfer symbol loaded) unless you specify another starting procedure.

You can also specify the program library list and command list entries. The command list entries are command libraries to be added to the command list. The command list determines the SCL commands available during the PPE session. The program library list is a list of additional object libraries from which object modules can be loaded as needed. Modules are loaded to satisfy external references. To find a module containing an entry point that matches an external reference, the loader searches the following locations in the order presented:

1. In the PPE object library at the current level.

2. In the PPE object library at each higher level upwards in the hierarchy.

3. In the object libraries specified in the program library list on the Library Lists screen. The libraries are searched in the order listed.

4. In the object libraries listed in the job library list. The libraries are searched in the order listed.

The program loading process is described in more detail in the SCL Object Code Management manual.

Getting Started 2

Getting Started 2

This chapter describes how you can begin to use PPE. To do so, you must:

1. Define your terminal as a full-screen terminal.
2. Enter the SCL command to start PPE.
3. On initial entry to PPE, set up your own PPE level.

These topics are described in detail in the following sections.

Defining Your Terminal

Because PPE uses a full-screen interface, it can only be used from an interactive terminal that has full-screen capabilities and is defined for full-screen use.

If you have used a full-screen application from your terminal (such as editing in screen mode using the NOS/VE Full-Screen Editor), your terminal is a full-screen terminal and you can prepare your terminal for PPE the same way you prepare it for other screen mode applications.

However, if you have not used a screen mode application from your terminal before, you need to learn how to define your terminal as a full-screen terminal. You may be able to find out from someone else at your site. If not, continue reading. The following paragraphs list the characteristics of a full-screen terminal and describe how you define your terminal as a full-screen terminal.

Full-Screen Terminal Characteristics

A terminal can be defined as a full-screen terminal if it has the following minimum characteristics:

- Uses asynchronous communications.
- Has keys that move the cursor on the screen and transmit characters indicating that the cursor has moved.
- Supports direct cursor addressing.
- Provides a clear-screen operation.

The following terminal characteristics are also desirable:

- Provides a clear-to-end-of-line function.
- Has up to 32 definable function keys, each of which transmits a unique, identifying character sequence. Preferably, the sequence should end with the carriage-return character.
- Allows host definition of tab stops.
- Supports protected screen fields and tabbing between unprotected fields. The tab key must transmit a character sequence to the host indicating that the key was pressed.
- Has graphic characters for drawing lines.

Full-Screen Terminal Definition

A terminal is defined as a full-screen terminal when you provide NOS/VE with a full-screen terminal definition to use during the interactive session. You do this by specifying a TERMINAL_MODEL parameter value on the SCL command CHANGE_TERMINAL_ATTRIBUTE. You can put the CHANGE_TERMINAL_ATTRIBUTE command in your prolog file ($USER.PROLOG) so it is executed each time you log in to NOS/VE.

The TERMINAL_MODEL parameter value selects a predefined description of your terminal. Your site probably has a set of terminal definitions available on file $SYSTEM.TDU.TERMINAL_DEFINITIONS. To see the definitions available at your site, enter the following SCL command:

```
display_object_library $system.tdu.terminal_definitions
```

The command lists the names of the terminal definition modules. By convention, the names begin with the prefix CSM$ followed by the TERMINAL_MODEL value. For example, one of the modules may be a terminal definition for the Zenith Z29 terminal named CSM$Z29. To use the CSM$Z29 module, you enter the following command:

```
change_terminal_attribute terminal_model=z29
```

Many terminals provide a VT100 (or ANSI X3.64) mode. If your terminal does, you may not find its name in the list of terminal models supported, but using VT100 should work. If you cannot find a compiled terminal definition available at your site that is effective for your full-screen terminal, you or someone at your site must create a new terminal definition. The process of creating a new terminal definition is described in the Terminal Definition for NOS/VE manual.

Using Function Keys

One of the important features of the full-screen interface is the use of function keys. However, the actual function keys available to you depend on the terminal you use.

You may need to consult the manual for your terminal or the person who wrote the terminal definition to find the function keys on your keyboard. For some terminals, function keys are entered by a combination of keys entered at the same time or in sequence. For example, on the Z29 terminal, unshifted function keys are the top row of keys on the keyboard and shifted function keys are the SHIFT key and a numeric pad key entered at the same time. After entering a Z29 function key, you must press the RETURN key.

As listed under Full-Screen Terminal Characteristics, function keys are not required to use PPE. When the terminal definition does not define function keys, all PPE actions must be specified by commands entered on the home line.

In many cases, a terminal has some function keys, but not a full set of 32. In such cases, some PPE functions are available via function key, while others must be entered by commands on the home line. The function keys defined by the terminal definition are displayed at the bottom of the screen.

Function keys can be entered shifted or unshifted. On the display at the bottom of a PPE screen, the top label is for the shifted key and the bottom label is for the unshifted key.

The function key label is always valid as the corresponding home line command, unless the label is changed by the user. Thus, when a description refers to a function key label, the referenced function can be performed by pressing the function key, entering the command on the home line of the screen, or by entering the function key label on the home line of the screen.

User Breaks in a Full-Screen Application

A user break alows you to interrupt an interactive program. To activate user_break_1 and user_break_2 in a full-screen application such as PPE, you must enter one or two commands before entering the application.

User_break_1 (also known as pause_break) and user_break_2 (also known as terminate_break) are interactive conditions caused when the user enters a certain key sequence. A terminate_break condition terminates the currently executing command.

A pause_break condition discards typed-ahead input and suspends the executing command. While the command is suspended, you can interact with the system to get information such as the job's status, consult online manuals, and so forth. To resume the suspended command, enter RESUME_COMMAND; to terminate the suspended command, enter TERMINATE_COMMAND.

CDCNET Commands

If the communications network you are using is CDCNET, you must enter two commands. The first command defines the attention character value; the second command defines the attention character as the terminate_break key and the terminal's break key as the pause_break key. The following example assumes that the network control character is the default (%) and the attention character is to be defined as CTRL-T (ASCII DC4, character code 20):

```
%change_terminal_attribute attention_character=20
%change_connection_attribute attention_character_action=1 ..
break_character_action=2
```

or abbreviated,

```
%chata ac=20
%chaca aca=1 bka=2
```

NAM/CCP Command

If the communications network you are using is NAM/CCP, you enter only one command. The command changes the attention character to a non-null value. For example:

```
change_terminal_attribute attention_character=$char(20)
```

or abbreviated,

```
chata ac=$char(20)
```

Starting PPE

After preparing your terminal for full-screen use, you can start PPE by entering the SCL command ENTER_PPE, defined as follows:

Format: ENTER_PPE or ENTP
ENVIRONMENT_CATALOG=catalog_path
STATUS=status_variable

Parameters: ENVIRONMENT_CATALOG or EC
Path to the subcatalog for which PPE is executed. It becomes the lowest level of the your PPE hierarchy in the session.

PPE creates the subcatalog if it does not exist. If the subcatalog belongs to another user, the owner must grant you the following catalog permit:

```
Access_Modes=(all, cycle, control)
Application_Information="Il"
```

If you omit ENVIRONMENT_CATALOG, the default subcatalog used is $USER.PROFESSIONAL_ENVIRONMENT.

STATUS
Optional status variable in which the command returns its completion status.

Examples: The following command starts a PPE session in which the PPE level used is the default subcatalog, the user's $USER.PROFESSIONAL_ENVIRONMENT subcatalog.

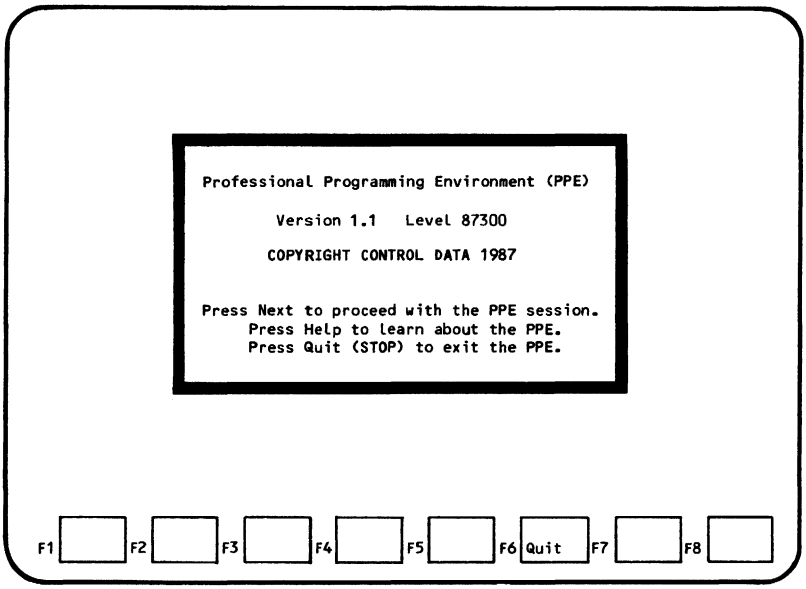
```
entp
```

The following command begins a PPE session using the subcatalog named $USER.XYZ.PPE_WORK:

```
entp ec=$user.xyz.ppe_work
```

Entering PPE for the First Time

The first screen you see each time you enter PPE is the PPE Banner screen. The Banner screen appears similar to the following display:



The Banner screen prompts you to make one of three choices:

NOTE

Commands cannot be entered on the home line of the Banner screen. Only commands executable by function keys are available.

1. Press Next to proceed with the PPE session.

   This choice displays the Environment Description screen allowing you to begin use of PPE. To select this choice, press the carriage-return key. (The carriage-return key transmits the carriage-return character; on various terminals, it is labeled RETURN, NEXT, CR, or a similar label.)

NOTE

When you select the first choice (Use Next to proceed with the PPE session), PPE creates the PPE catalog structure in the catalog specified on the ENTER_PPE command if the catalog has not been used as a PPE catalog before. The existing files and catalogs in the specified subcatalog are not affected. For more information on the PPE catalog structure, see appendix D.

2. Press Help to learn about the PPE.

   If you select this choice, you are shown the online manual introduction to PPE. To get help, press the Help key. Depending on your terminal definition, this could be the key labeled HELP on your terminal or a function key labeled Help at the bottom of the Banner screen.

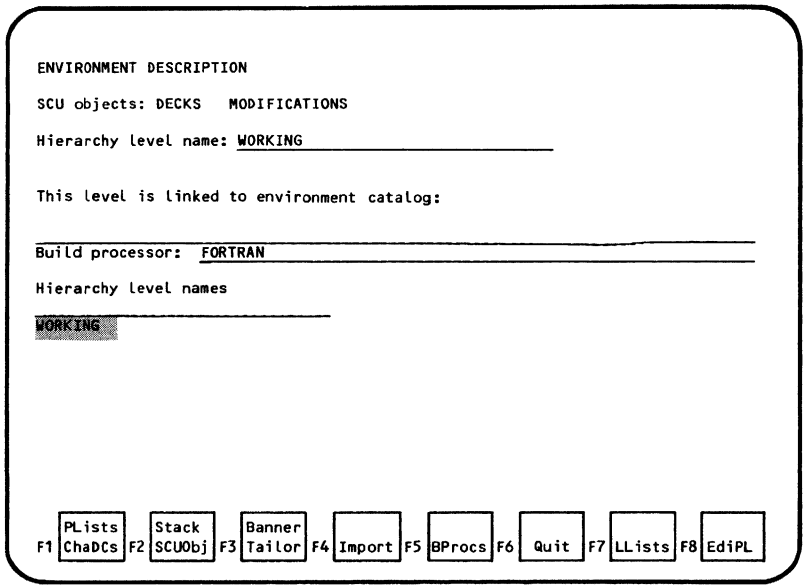
3. Press Quit (STOP) to exit the PPE.

   This choice ends the PPE session without creating the PPE subcatalog. To leave PPE, press the Stop or Quit key. Depending on your terminal definition, this could be the key labeled STOP on your terminal or a function key labeled Quit at the bottom of the Banner screen.

The Banner screen is always the first screen displayed when you enter PPE. However, the screen displayed when you proceed can differ. The screen displayed when you press the carriage-return key is the last screen displayed during your previous PPE session. This allows you to continue your work where you left your previous session.

Environment Description Screen

During your initial PPE session, the PPE screen you see after pressing the carriage-return key from the Banner screen is the Environment Description screen. At first, the Environment Description screen appears similar to the following (the function key labels at the bottom of your screen may differ):



You can begin by naming your PPE level. The name can be any descriptive sequence of 1 through 31 characters that follows NOS/VE naming rules. To name your level, type the level name after

Hierarchy level name:

You can type over any existing level name to to change your level name. However, be sure to strike over all characters of the old name. If the new name does not replace all characters of the old name, erase the remaining characters of the old name with blanks. Press the carriage-return key and your new name appears in the Hierarchy level names list at the bottom of the screen.

Linking to a Hierarchy

The Environment Description screen defines how your PPE level relates to other PPE levels in a hierarchy. Initially, your level is not linked to any hierarchy. You are not required to link to a hierarchy. However, if you intend to extract decks from and transmit decks to higher levels of a hierarchy, you must link your level to the hierarchy.

To link your PPE level to a hierarchy, specify the level to which your level is to be linked. Specify the hierarchy level by its catalog path. Type the catalog path on the line below the heading:

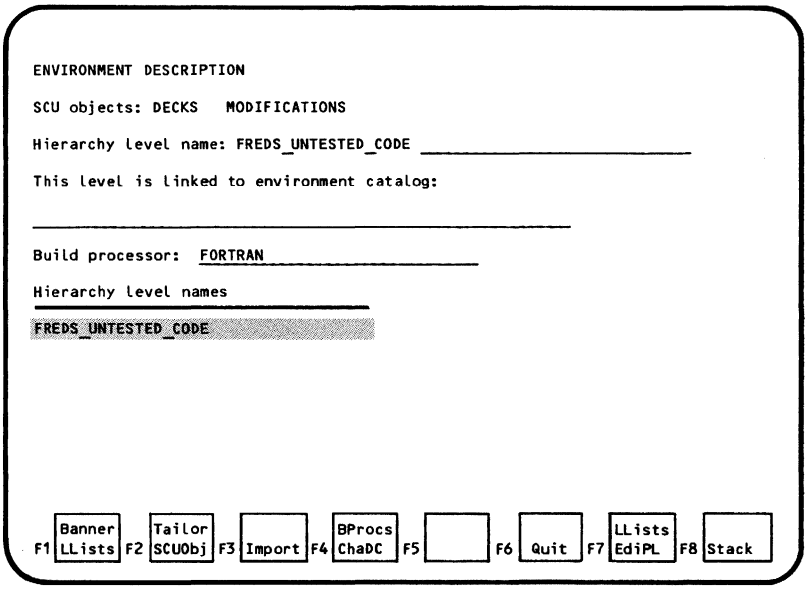
```
This level is linked to environment catalog:
```

Press the carriage-return key and the level names in the hierarchy will appear in the Hierarchy level names list. Your level name will be at the bottom of the list.

For example, to name an environment catalog level FREDS_UNTESTED_CODE, type the following level name in the Hierarchy level name: field:

```
freds_untested_code
```

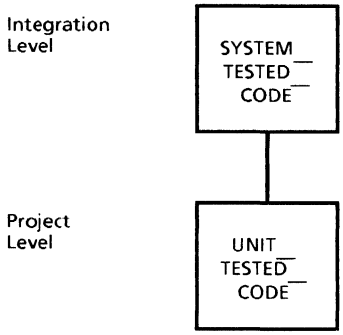
The Environment Description screen then appears similar to the following:



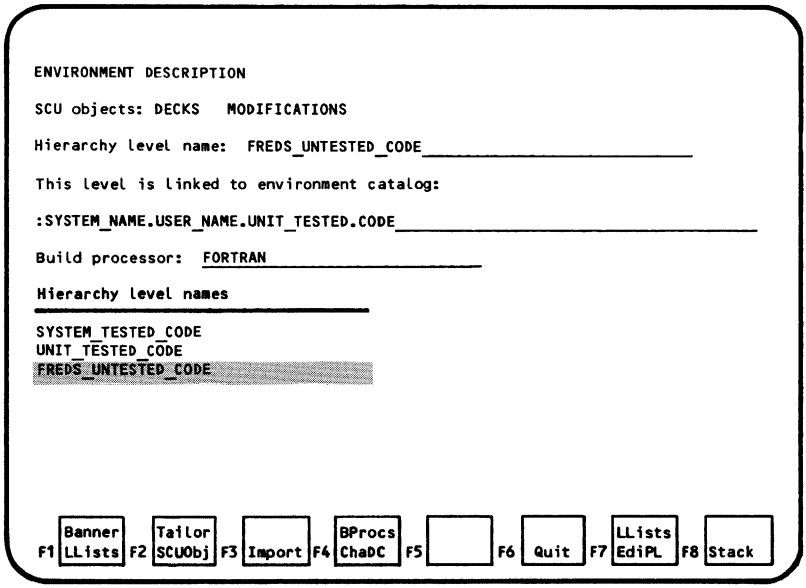
The environment catalog level is named FREDS_UNTESTED_CODE. The level name is added to the Hierarchy level names list. Since FREDS_UNTESTED_CODE is not linked to any other catalogs, it belongs to no environment catalog hierarchy and no other level names are listed in the Hierarchy level names list.

You can work on code using an environment catalog not connected to a hierarchy, but you cannot extract from or transmit to other PPE catalogs. To do so, your environment catalog must be linked to another environment catalog.

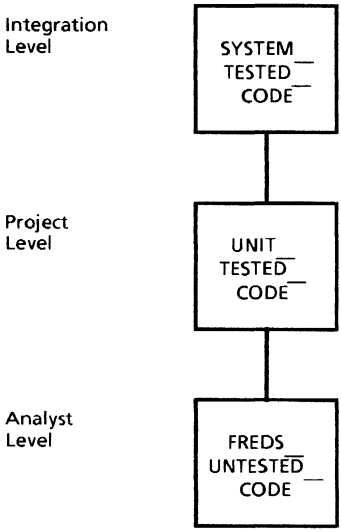
A typical environment catalog hierarchy might consist of an integration level, a project level, and an analyst level. An integration level and project level hierarchy might appear similar to the following:



If you want to link your level to another environment catalog, you must specify the file path of the catalog on the This level is linked to: field of your Environment Description screen. For example, to link the environment catalog level FREDS_UNTESTED_CODE to the UNIT_TESTED_CODE environment catalog level, specify the file path of the UNIT_TESTED_CODE environment catalog. After you specify the path, the Environment Description screen appears similar to the following:



The Hierarchy level names list contains the level names in Fred's environment catalog hierarchy. FREDS_UNTESTED_CODE is the bottom level of the hierarchy. If you could see the hierarchy, it would appear similar to the following:



Fred can now transmit decks to and extract decks from the environment catalogs in the environment catalog hierarchy.

Build Processor

Notice the Build processor: field on the Environment Description screen. This field specifies the processor used for all builds at your level of the hierarchy. Initially, the Build processor: field is set to FORTRAN, indicating that the NOS/VE FORTRAN Version 1 compiler is the build processor used.

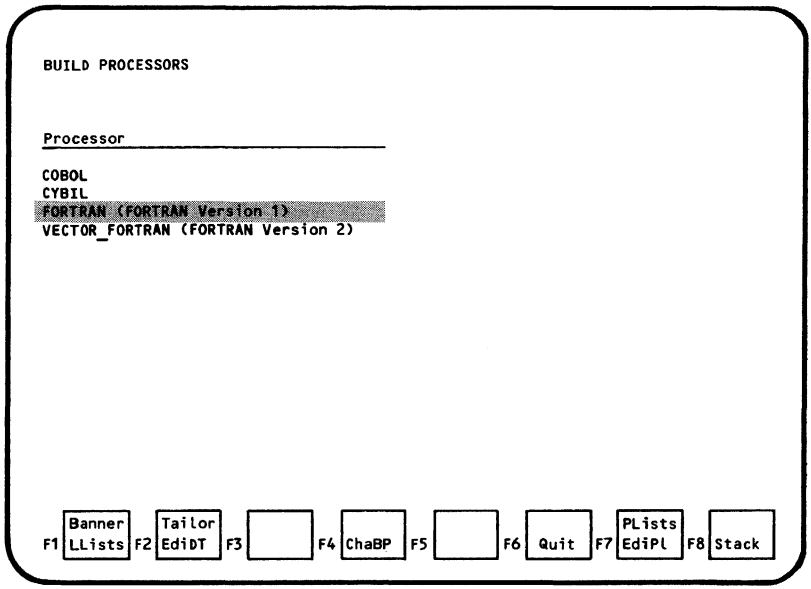
You can change the value in the Build processor: field by typing over the existing value. The specified build processor is used for all subsequent builds and so it must be an appropriate processor for the source code at your level. For example, it would not be appropriate to specify COBOL as the build processor when the source code to be compiled is FORTRAN source code.

Display Build Processors

To see a list of all build processors currently supported by PPE, request the DISPLAY_BUILD_PROCESSORS function by either of the following:

- Press the function key labeled BProcs, or
- Move the cursor to the home line, type the command DISPLAY_BUILD_PROCESSORS (or DISBP or BPROCS), and press the carriage-return key.

The Build Processors screen appears similar to the following display:



The build processor in use is highlighted on the Build Processors screen. To change the build processor, do one of the following:

- Position the cursor on the build processor name and then press the function key labeled ChaBP, or
- Position the cursor on the build processor name, press the HOME key, type the command CHANGE_BUILD_PROCESSOR (or CHABP), and press the carriage-return key, or
- Move the cursor to the home line, type the command CHANGE_BUILD_PROCESSOR (or CHABP) followed by the build processor name, and press the carriage-return key.

The other functions available from the Build Processors screen are described in the screen description in chapter 4. To return to the Environment Description screen, either:

- Press the BACK function key, or
- Move the cursor to the home line, type the command BACK_TO_PREVIOUS_CONTEXT (or BACTPC or BACK), and press the carriage-return key, or
- Use the Screen Stack screen.

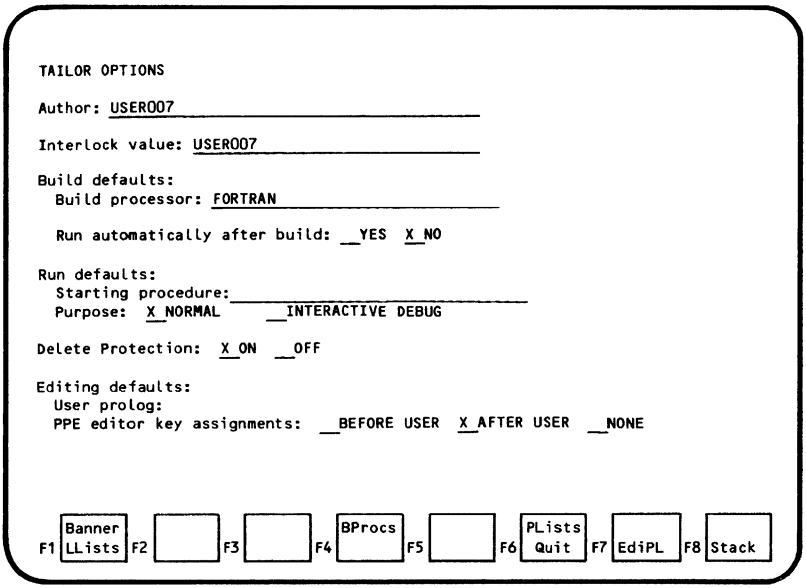
Tailor Options Screen

During your first PPE session, you might also look at the Tailor Options screen. It provides a number of options for tailoring your PPE level.

To display the Tailor Options screen, do either of the following:

- Press the function key labeled Tailor, or
- Move the cursor to the home line, type the command DISPLAY_TAILOR_OPTIONS (or DISTO or TAILOR), and press the carriage-return key.

The Tailor Options Screen appears similar to the following:



Initially, the values in the Author: and Interlock value: fields are the same; both are the NOS/VE user name of the job containing the PPE session. The Author: value on the Tailor Options screen is the default value PPE enters in the Author deck header field of created decks. The Interlock value: is the value PPE enters in the interlock fields when a deck is extracted.

If you like, you can change the Author and/or Interlock values to a value more descriptive, such as your name. To do so, type over the existing values and then press the carriage-return key.

The Build defaults are values used as defaults when the product is built. The Build processor value on the Tailor Options screen is the same as it is on the Environment Description screen. As described for the Environment Description screeen, you can change the Build processor value by typing over it and pressing the carriage-return key. Be sure to overstrike and blank out all characters of the old value, because any characters of the old value that remain on the screen become part of the new value. The value must be a keyword specifying one of the build processors supported by PPE.

The default value for the Run automatically after build field is NO, meaning that the product is not executed at the end of a successful build. If you like, you can change the value to YES, causing the product to execute after every successful build. To change the value, type a non-blank character in the field next to YES and press the carriage-return key.

The Run defaults are default values used when the product is executed. When the Starting procedure field is blank, execution begins at the last transfer symbol loaded, usually the main program. If you like, you can change the value to the name of a specific entry point where execution is to begin. To do so, type the name in the Starting procedure field and press the carriage-return key. (For a more detailed description of starting procedures, see the NOS/VE Object Code Management Usage manual).

The Purpose field sets the default run purpose for each execution of the product. Two run purposes are available:

- Normal execution (NORMAL)
- Execute under control of the Debug utility (INTERACTIVE DEBUG)

To change the default run purpose, type a non-blank character in the field you select and press the carriage-return key.

The Delete protection default allows you to enable or disable delete protection. Delete protection warns you when objects are about to be physically deleted. Options are:

ON PPE issues a warning before physically deleting an object. You can then veto or confirm the deletion of the object. This is the default selected by PPE.

OFF PPE physically deletes objects without allowing you to veto the action.

To select either option, type a non-blank character in the field to the left of the option.

When you use a DELETE command (for example, DELETE_DECK), the deleted object is logically deleted. This means that the name of the object is removed from the screen, but the object is still physically resident in PPE. Logically deleted objects can be recovered with the UNDO_LAST_DELETE command.

A logically deleted object becomes physically deleted if:

- Delete protection is ON and you confirm the deletion. The Delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.

- You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.

- Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

A physically deleted object cannot be recovered by invoking the UNDO_LAST_DELETE command. PPE can physically delete only objects that are logically deleted.

The Editing defaults: fields enable you to specify an editor prolog to be used when executing the PPE editor. The editing defaults let you define editor function keys using a combination of your function key definitions and PPE commands or using your function key definitions only. Therefore, when entering the PPE editor, you can use the PPE supplied editor prolog, an editor prolog of your choice, or a combination of both to define the editor function keys.

If you want to define the PPE editor function keys using your editor prolog, enter the file path of your editor prolog in the User prolog: field. For example, to use C721_EDITOR_PROLOG to define PPE editor function keys, type the following file path in the User prolog: field:

```
$user.C721_editor_prolog
```

If you do not specify an editor prolog in this field, PPE uses the editor prolog, SCU_EDITOR_PROLOG, to define the editor function keys. If the file, SCU_EDITOR_PROLOG, does not exist, PPE defines the editor function keys using the standard function key definitions.

The PPE editor key assignments: fields allow you to specify whether you want PPE editor key assignments made before or after your editor function key assignments are made, or whether you do not want any PPE editor key assignments made. The choices are:

BEFORE USER	Causes PPE to assign PPE commands to the editor function keys before executing your editor prolog. First PPE assigns PPE commands to the editor function keys, then the editor prolog assigns commands to the editor function keys.
AFTER USER	Causes PPE to assign PPE commands to the editor function keys after executing your editor prolog. First the editor prolog assigns commands to the the function keys, then PPE assigns PPE commands the keys.
NONE	Causes PPE to not assign any PPE commands to the editor function keys.

When you choose BEFORE USER or AFTER USER, both PPE defined and user defined function keys are specified for the PPE editor. Any function key defined by both your user prolog and by PPE commands assumes the definition assigned to the key last. Therefore, to use your editor prolog with the PPE editor prolog to maximum effect, do the following:

- Determine which function keys are assigned PPE commands.
- Define your editor prolog only for function keys not assigned a PPE command.
- Specify the file path of your editor prolog on the User prolog: field (the editor prolog can be any file containing EDIT_FILE function key definitions).
- Choose BEFORE USER or AFTER USER on the PPE editor key assignments: field.

From the Tailor Options screen, you may also want to go to the Library Lists, Parameter List, and Parameter List Library screens. To go to these screens, you can use the function keys labeled LLists, EdiPL, and PLists or the commands DISPLAY_LIBRARY_LISTS, EDIT_PARAMETER_LIST, and DISPLAY_PARAMETER_LIST_LIBRARY. These screens are described in chapter 4.

Leaving PPE

You can end your PPE session at any time. When you exit PPE from any screen other than the Deck Creation or Modification Creation screens, all of the files created and options selected are saved for your next PPE session. If you exit PPE from the Deck Creation or Modification Creation screens, any new information entered on the screen is lost.

To leave PPE, do one of the following:

- Press the STOP key if your terminal has one, or
- Press the function key labeled Quit, or
- Move the cursor to the home line, type the command QUIT_SAVE (or QUIT or QUIS or QUI), and press the carriage-return key.

The next time you enter PPE, the first screen you are shown after the Banner screen is the last screen at which you performed an action in the previous session unless you exited from the Screen Stack screen.

File Cycle Management Within the Environment Catalog

When you exit PPE, a new high cycle of the source library within your environment catalog is created. The low cycles allow you to recover from catastrophic events. However, you can easily use up a large amount of file space. To avoid using excess file space, you should frequently delete the low cycles of your source library. You can use EDIT_CATALOG from the home line of your environment catalog to do this.

The next chapter is an extended example of the primary PPE functions. The chapter following it is a reference chapter that describes all PPE screens and commands in detail.

PPE Operations 3

PPE Operations 3

This chapter contains procedures for performing the primary PPE operations. These include:

Moving from screen to screen

Importing a source library

Creating a modification

Creating a deck

Extracting a deck

Editing a deck

Building the product

Correcting errors

Transmitting code

This chapter does not describe all PPE commands or all PPE screens. The next chapter contains comprehensive descriptions of all PPE commands and screens.

See Appendix G, Usage Hints, for suggestions about using PPE with exiting SCU libraries and on new projects. Appendix G also shows you how to customize the PPE function key assignments.

NOTE

Changes made to the source library are not available to other tasks or PPE sessions at lower levels in the hierarchy until you exit the PPE session in which the changes occurred, or until the changed decks are transmitted to the next higher level in the hierarchy.

Moving from Screen to Screen

Moving from screen to screen within PPE does not actually accomplish any work, but it is an operation that is required for doing work. You can go to a screen by entering the specific command to display that screen or you can use the Screen Stack. (To find out which command displays a specific screen, look up the screen's description in this manual.)

Moving from Screen to Screen

As you move down the PPE screen hierarchy, each screen you display, except the Banner screen and the Screen Stack screen, is put on the stack. Each time you move back up the hierarchy, the current screen is removed from the stack. You can use the Screen Stack to go directly to a screen if the screen is currently on the stack.

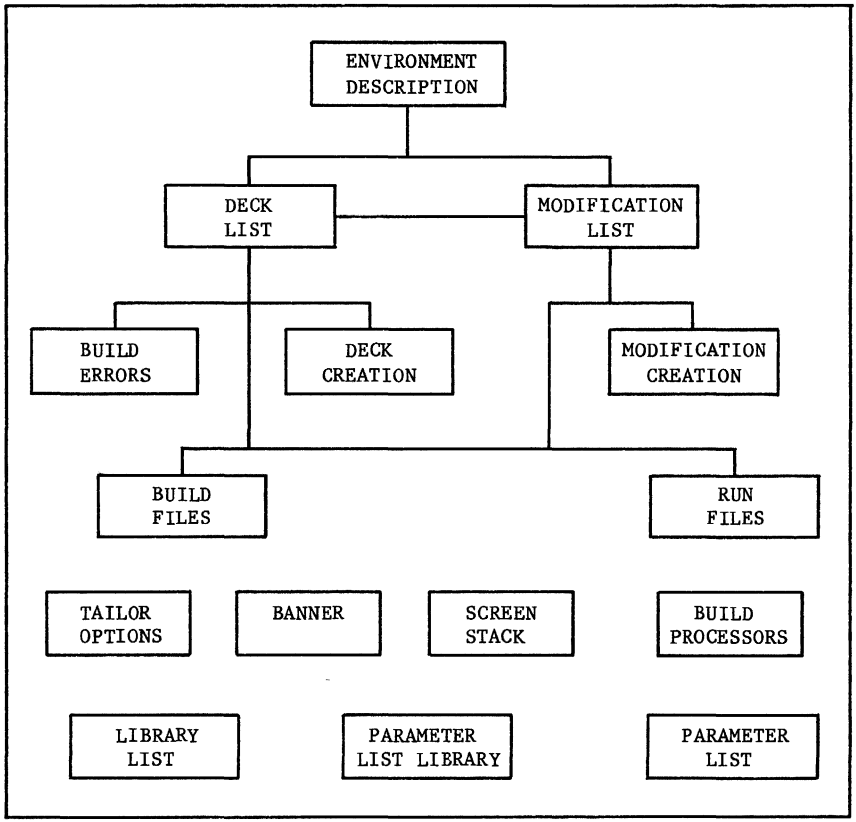
Thus, the PPE screens can appear on the stack only in hierarchical order. The hierarchy of PPE screens is as follows:

1. Environment Description

2. Deck List or Modification List

3. Build Errors, Build Files, Run Files, or the Deck Creation or Modification Creation screen from Deck List or Modification List, respectively.

The diagram below shows the PPE screen hierarchy. The screens not connected to any other screens can be reached from many screens. See the PPE Screens section of chapter 4 to determine which screen transitions are possible.

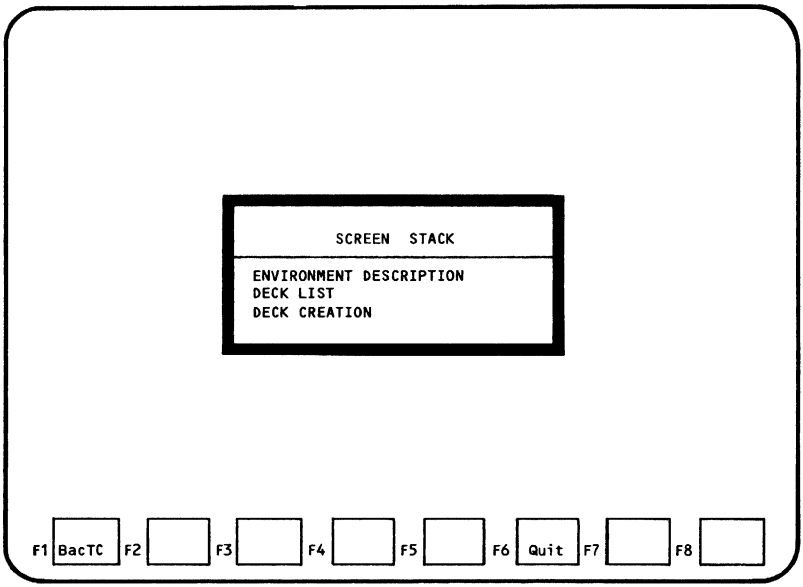


You can retrace your path down the PPE screen hierarchy by using BACK_TO_PREVIOUS_CONTEXT repeatedly. To do so, do either of the following:

Press the BACK function key, or

Move the cursor to the home line, type BACK_TO_PREVIOUS_CONTEXT (or, BACTPC or BACK) and press the carriage-return key.

You can jump directly to a screen in the stack using the Screen Stack screen. To go to the Screen Stack screen, press the function key labeled STACK or move the cursor to the home line, type the command DISPLAY_SCREEN_STACK (or DISSS or STACK), and press the carriage-return key. The Screen Stack screen then appears, similar to the following:



To go to a screen on the stack:

1. Position the cursor on the screen name (such as ENVIRONMENT DESCRIPTION).
2. Press the function key labeled BacTC.

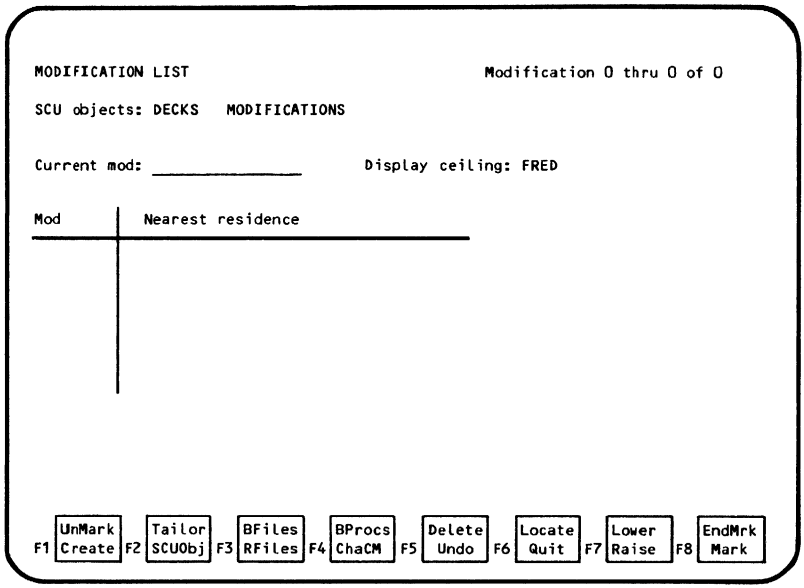
   PPE removes all entries below the selected screen from the stack and then displays the requested screen.

To create a modification:

1. Go to the Modification List screen. If the Modification List screen is on the screen stack, you can return there by using the Screen Stack screen or by using BACK one or more times. If not, you must go to the Environment Description screen or the Deck List screen and perform one of the following:

   - Position the cursor on the SCU object MODIFICATIONS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object MODIFICATIONS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST MODIFICATION (or DISSOL M or SCUOBJ M), and press the carriage-return key.

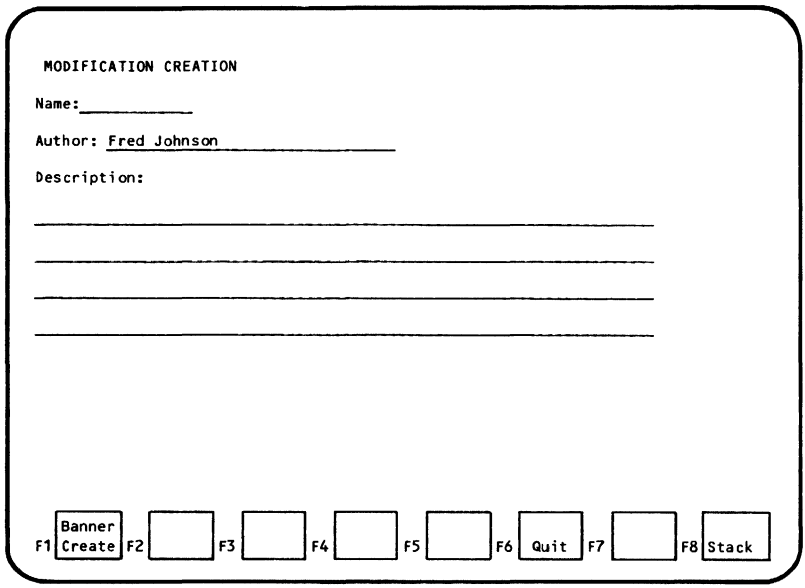
   The Modification List screen appears, similar to the following:



2. Go to the Modification Creation screen. To do so, do either of the following:

   - Press the function key labeled Create, or
   - Move the cursor to the home line, type CREATE_MODIFICATION (or CREM or CREATE), and press the carriage-return key.

   The Modification Creation screen appears, similar to the following:



3. Enter the information defining the modification.

   Type the modification name (1 through 9 characters) in the Name field.

   By default, the Author field contains the Author value from the Tailor Options screen. To change it, type over its current value.

   The modification description is optional. It usually describes the purpose of the modification. To enter it, type the information on the lines following the label:

   Description:

4. Cancel or create the modification.

   Canceling the modification discards the information entered on the Modification Creation screen and returns you to the previous screen displayed.

   To cancel a modification:

   - Press the function key labeled Back, or
   - Move the cursor to the home line, type BACK_TO_PREVIOUS_CONTEXT (or BACTPC or BACK) and press the carriage-return key.

   Creating the modification causes PPE to give the information on the Modification Creation screen to SCU which, in turn, creates the modification in the source library. PPE then returns you to the Modification List screen.

   If a modification with the same name already exists at any level in the PPE hierarchy, PPE does not create the the modification. Instead, it displays an explanatory message and leaves you at the Modification Creation screen. You can then change the modification name and request creation again or, you can leave the screen.

   To create a modification:

   - Press the function key labeled Create, or
   - Move the cursor to the home line, type CREATE_MODIFICATION (or CREM or CREATE) and press the carriage-return key.

Creating a Deck

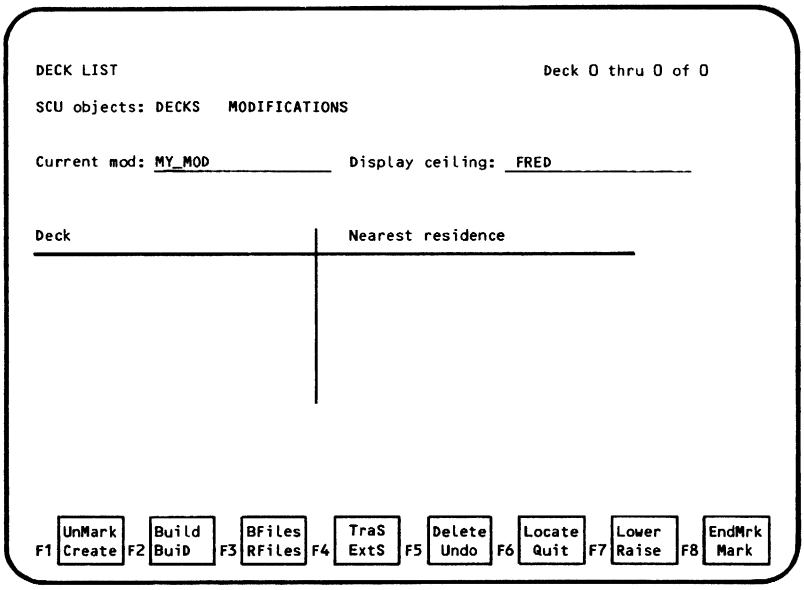
As described in chapter 1, an SCU deck contains a sequence of source lines. Often, the source text in a deck is for a compilation unit, such as a FORTRAN or COBOL subprogram. PPE provides a full-screen interface for deck creation.

To create a deck:

1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using the Screen Stack screen or pressing BACK one or more times. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object DECKS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

   The Deck List screen appears, similar to the following:



2. Go to the Deck Creation screen by doing any one of following:

   - Press the function key labeled Create, or
   - Move the cursor to the home line, type CREATE_DECK (or CRED or CREATE), and press the carriage-return key.

   The Deck Creation screen appears, similar to the following:

```
DECK CREATION

Creation modification: MY_MOD
Name: _____________________ Author: Fred Johnson

Deck is expandable: X TRUE  __FALSE

Initial source:
  X NONE  __DECK TEMPLATE

  File:_____________________________________________________

  Attributes: __CONTAINS DECK DIRECTIVES   __HAS MULTIPLE PARTITIONS

Description
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________

F1 Banner Create F2 [ ] F3 [ ] F4 [ ] F5 [ ] F6 Quit F7 [ ] F8 Stack
```

3. Enter the deck header information.

   Type the deck name (1 through 31 characters) in the Name field.

   The default author is the Author value from the Tailor Options screen. To change the author value, type over the value in the Author field.

   The creation modification is the modification to which the original set of lines in the deck are assigned. It is the current modification as shown Deck List screen. To change the modification value, type over the value in the Creation Modification field. If the modification does not exist at any level of the environment catalog hierarchy, PPE asks whether you want the modification created with SCU default values. Otherwise, if the modification does not exist at the lowest level of the hierarchy, PPE issues a message and returns the modification field to its previous state.

If you let PPE create the modification with SCU defaults, the author parameter of the modification is not set. Only values provided by the SCU are used to create the modification; information on the Deck Creation screen is not used.

The expand attribute determines whether the deck can be expanded directly or whether it must be copied by another deck. The default expand attribute is TRUE. To change the value, type a non-blank character by the FALSE field following the label:

```
Deck is expandable:
```

The Initial source: information describes the text stored in the deck when it is created. By default, no text is stored in the deck. You can specify that the initial source is the deck template or the text in a file.

The procedure for editing the deck template is described later under Editing the Deck Template.

If the initial text is in a separate file, type the file path in the File field. When the initial text is in a file, you can also specify one or both of the attributes CONTAINS DECK DIRECTIVES and HAS MULTIPLE PARTITIONS.

When you select the attribute CONTAINS DECK DIRECTIVES, SCU processes any *DECK directives embedded in the source text. Each *DECK directive marks the beginning of the text for another deck. The *DECK directive specifies the name of the deck. All other deck header information is identical for all decks created from the file.

When you select the attribute HAS MULTIPLE PARTITIONS, SCU continues copying text from the file even when it encounters end-of-partition delimiters. It converts each end-of-partition delimiter to a *WEOP embedded directive, which is converted back to an end-of-partition delimiter when the deck is expanded.

The final field on the screen is the deck description. A description is optional. It usually describes the content of the deck.

4. Cancel or create the new deck.

Canceling the deck discards the information entered on the Deck Creation screen and returns you to the previous screen displayed.

To cancel a deck:

- Press the function key labeled Back, or
- Move the cursor to the home line, type BACK_TO_PREVIOUS_CONTEXT (or BACTPC or BACK) and press the carriage-return key.

Creating the deck causes PPE to give the information on the Deck Creation screen to SCU which, in turn, creates the deck (or decks) in the source library. PPE then returns you to the previous screen displayed.

To create a deck, do either of the following:

- Press the function key labeled Create, or
- Move the cursor to the home line, type CREATE_DECK (or CRED or CREATE), and press the carriage-return key.

If a deck with the same name already exists at any level in the environment catalog hierarchy, PPE does not create the deck. Instead, it displays an explanatory message and leaves you at the Deck Creation screen. You can then change the deck name and create the deck, or you can leave the screen.

NOTE

PPE does not check to ensure that a deck is unique in the environment catalog hierarchy if you specify a file as the initial source of the deck and the file contains deck directives.

Editing the Deck Template

Each build processor (compiler) has a PPE deck template associated with it. The deck template is a generic framework for decks written in the compiler language; the template can be copied to a deck being created. You can edit the deck template to fit your requirements.

To edit the deck template, perform these steps:

1. Go to the Build Processors screen by doing either of the following:
   - Press the function key labeled BProcs, or,
   - Move the cursor to the home line, type DISPLAY_BUILD_PROCESSORS (or DISBP or BPROCS), and press the carriage-return key.

2. Do either of the following:

   - Position the cursor on the build processor whose template you want to see and press the function key labeled EdiDT, or

   - Move the cursor to the home line, type EDIT_DECK_TEMPLATE (or EDIDT) and the name of the build processor, and press the carriage-return key.

   The deck template is displayed for editing in a Full-Screen Editor session. The commands available are described under Editing a Deck.

Extracting a Deck

The third method of getting decks into your PPE source catalog (besides importing source libraries and creating decks) is extracting decks from higher levels in the PPE hierarchy. You extract a deck when you need to make changes to source code that already exists in a higher level.

As described in chapter 1, extracting a deck makes a copy of the deck and adds it to your PPE source library. It sets an interlock on the deck so nobody else can extract the deck until after you transmit it.

To extract a deck:

1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using BACK one or more times or by using the Screen Stack screen. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or

   - Position the cursor on the SCU object DECKS and press the carriage-return key, or

   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

2. Raise the display ceiling until the deck (or decks) to be extracted is in the list on the screen. (The procedure for raising the display ceiling is described later under Raising the Display Ceiling.)

   Often, higher levels have long deck lists, causing the deck list to be longer than can be displayed on a single screen. Therefore, you may need to move forward and backward through the list to find the decks to be extracted. (The procedures for repositioning a list are described later under Repositioning a List.)

3. Mark the deck (or decks) to be extracted. (The procedures for marking objects in a list are described later under Marking Objects in a List.)

   If no decks are marked, the deck under the cursor is extracted.

4. To extract the deck (or decks), either:

   - Press the function key labeled ExtS, or
   - Move the cursor to the home line, type EXTRACT_SOURCE (or EXTS) and press the carriage-return key.

PPE extracts as many of the marked decks as it can. If it cannot extract one or more of the decks, PPE displays an informative message listing the decks that were not extracted. A deck cannot be extracted if it is already interlocked or if it exists at your level of the hierarchy. (It may be interlocked because you have already extracted it. You can see that this is so if the level listed in the Nearest residence column beside the deck name is the name of your PPE level.)

Raising the Display Ceiling

On many screens, you can raise the display ceiling. The display ceiling is the highest level in the hierarchy whose contents are included in the list on the screen. Raising the display ceiling adds the contents of higher levels to the list.

To raise the ceiling one level, either:

- Press the function key labeled Raise, or
- Move the cursor to the home line, type RAISE_DISPLAY_CEILING (or RAIDC or RAISE), and press the carriage-return key.

To raise the ceiling more than one level, type the number of levels after the RAISE_DISPLAY_CEILING command. To raise the ceiling to the top of the hierarchy, type TOP after the RAISE_DISPLAY_CEILING command.

The level name of the display ceiling appears in the Display Ceiling field.

Repositioning a List

Many PPE screens display lists of objects. The lists are often too long to fit on the screen. The list header indicates how many entries are in the list and which entries are currently displayed. You can reposition a list display to find the objects you require; a list can be repositioned as follows:

- Forward or backward one page. (A page is the number of entries that can fit on the screen.)
- Forward or backward so that the object under the cursor is positioned at the top or bottom of the page, respectively.
- Forward to the end of the list or backward to the beginning of the list.

The following paragraphs describe how you perform these repositioning operations.

To move a list one page forward:

- Press the function key labeled Fwd, or
- Move the cursor to the home line, type PAGE_FORWARD (or PAGF or FWD), and press the carriage-return key.

To move a list one page backward:

- Press the function key labeled Bkw, or
- Move the cursor to the home line, type PAGE_BACKWARD (or PAGB or BKW), and press the carriage-return key.

To move a list so the object under the cursor is at the top of the page:

- Press the function key labeled Up, or
- Move the cursor to the home line, type MOVE_TO_TOP (or MOVTT or UP), and press the carriage-return key.

To move a list so the object under the cursor is at the bottom of the page:

- Press the function key labeled Down, or
- Move the cursor to the home line, type MOVE_TO_BOTTOM (or MOVTB or DOWN), and press the carriage-return key.

To display the beginning of a list and move the cursor to the first object:

- Press the function key labeled First, or
- Move the cursor to the home line, type MOVE_TO_FIRST (or MOVTF or FIRST), and press the carriage-return key.

To display the end of a list and move the cursor to the last object:

- Press the function key labeled Last, or
- Move the cursor to the home line, type MOVE_TO_LAST (or MOVTL or LAST), and press the carriage-return key.

Marking Objects in a List

Many PPE screens display lists from which one or more objects can be selected. Multiple objects are selected by marking them in the list. PPE highlights the marked objects (assuming the terminal definition provides a highlighting capability).

To mark one object:

1. Position the cursor on the object to be marked in the list.

2. Either:
   - Press the function key labeled Mark, or
   - Put the cursor on the object, then move the cursor to the home line, type BEGIN_MARK (or BEGM or MARK), and press the carriage-return key.

To mark a range of objects in the list:

1. Mark the first object in the range using the procedure for marking one object.

2. Mark the last object in the range by first positioning the cursor on the last object and then either:

   - Press the function key labeled EndMrk, or
   - Move the cursor to the home line, type END_MARK (or ENDM or ENDMRK), and press the carriage-return key.

You can mark more than one individual object or object range in the list.

If necessary, you can remove marks previously set.

To remove all marks, move the cursor to the home line, type REMOVE_MARK ALL (or REMM A), and press the carriage-return key.

To remove only one mark or range of marks:

- Position the cursor on the mark before pressing the UnMark function key, or
- Move the cursor to the home line, type REMOVE_MARK RANGE (or REMM R) or REMOVE_MARK INDIVIDUAL (or REMM I), and press the carriage-return key.

Editing a Deck

The actual work of changing source code is accomplished by editing decks. When you request deck editing, PPE puts you into a full-screen editing session with the text of the deck to be edited displayed.

You can edit a deck from the Deck List screen or the Deck Creation screen. The changes made during the editing operation are assigned to the modification in the Current Modification field of the Deck List screen or the Creation Modification field of the Deck Creation screen. Check that the correct modification is specified before doing the edit operation.

Editing a Deck

To edit a deck being created:

1. Go to the Deck Creation screen and enter the deck information (as described under Deck Creation).

2. Request editing:

   - Press the function key labeled Edit, or
   - Move the cursor to the home line, type EDIT_DECK (or EDID or EDIT), and press the carriage-return key.

To edit an existing deck:

1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using BACK one or more times. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object DECKS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

2. Position the cursor on the deck to be edited.

   When you request editing of a deck at a higher level in the hierarchy, PPE also extracts the deck. To display higher levels, raise the display ceiling as described under Raising the Display Ceiling. Paging forward and backward through a deck list is described under Repositioning a List.

3. Request editing:

   - Press the function key labeled Edit, or
   - Move the cursor to the home line, type EDIT_DECK (or EDID or EDIT), and press the carriage-return key.

   PPE starts a full-screen editing session to edit the text in the deck. To leave the editor and return to PPE with your changes saved, do one of the following:

   - Press the STOP key or,
   - Press the function key labeled Quit or,
   - Move the cursor to the home line, type QUIT, and press the carriage-return key.

   All standard full-screen editor commands are available in an editing session under PPE. (The editor commands are described in the Full-Screen Editor for NOS/VE manual.) PPE also provides some specialized capabilities including:

   - Topic lookup in the compiler online manual and,
   - Source text formatting.

These additional Full-Screen Editor capabilities are described next.

Online Manual Lookup

If, while editing, you need to see the online manual information about a keyword in your source text, do the following:

1. Position the cursor on the keyword.

2. Request lookup, by either of the following:

   - Press the function key labeled LookUp, or
   - Move the cursor to the home line, type LOOKUP_KEYWORD (or LOOK or LOOKUP), and press the carriage-return key.

LOOKUP_KEYWORD passes the keyword as a topic string to the EXPLAIN utility which searches for the topic in the index of the compiler online manual. If the topic is in the index, the corresponding screen of the online manual is displayed. You can read the topic and search the manual for other information as desired. (For more information on reading an online manual, press the HELP key while inside the manual.)

To return to PPE from the online manual, press the QUIT key.

Formatting Source Text

When you edit your source, you can use the PPE provided command FORMAT_SOURCE_TEXT to format the source text.

FORMAT_SOURCE_TEXT looks for a command with a name of the form FORMAT_processor_name_SOURCE where processor_name is the name of the processor for the source program or deck being edited (for example, FORMAT_COBOL_SOURCE). If FORMAT_SOURCE_TEXT finds a command with a name of this form, it executes the command. Two parameters are passed to the command: the name of the source file to be formatted, and the name of the file to contain the formatted source.

If a command of the form FORMAT_processor_name_SOURCE is not found and PPE provides a source code formatter for the processor, then the PPE provided formatter is used to format the source code. Currently, PPE only provides a source code formatter for FORTRAN Version 1.

If a command of the form FORMAT_processor_name_SOURCE is not found and PPE does not provide a source code formatter for the processor, PPE issues a message informing you that a formatter does not exist for the processor.

PPE provides a formatter for FORTRAN Version 1 source code. The formatter standardizes the appearance of your source code, including indenting block structures and positioning labels. However, the source text must already be in the correct zones (labels in columns 1 through 5, statements beginning in column 7).

To format the source text of the deck being edited, do either of the following:

- Press the function key labeled Format, or
- Move the cursor to the home line, type FORMAT_SOURCE_TEXT (FORST), and press the carriage-return key.

Building the Product

As described in chapter 1, to build the product means to expand the text source decks and compile the expanded text. The build process produces an updated object library containing the executable code for the product.

PPE provides two types of build operations: one to build changed decks, and the other to build selected decks.

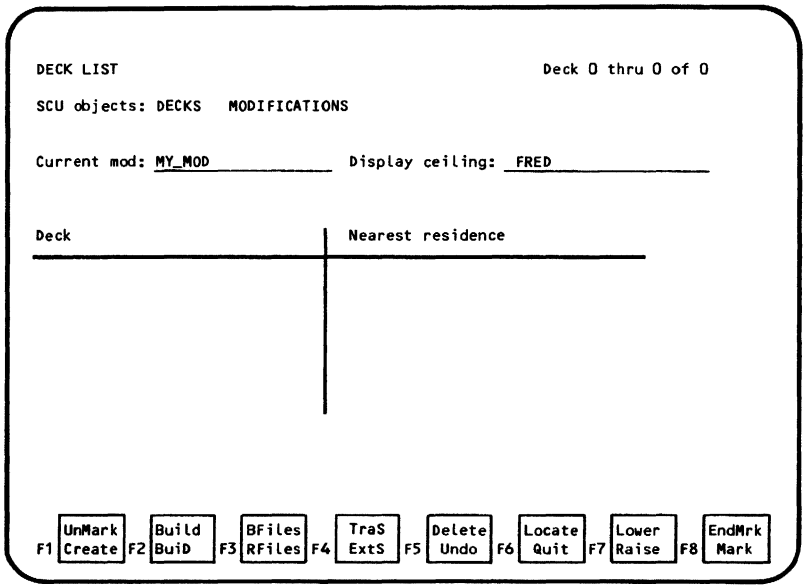
Building Selected Decks

To build selected expandable decks:

1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using the Screen Stack screen or pressing BACK one or more times. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object DECKS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

The Deck List screen appears, similar to the following:



2. Select the decks by marking individual decks or ranges of decks as described under Marking Objects in a List. If no decks are marked, only the deck under the cursor is selected.

3. Request the build by doing either of the following:

   - Press the function key labeled BuiD, or
   - Move the cursor to the home line, type BUILD_DECKS (or BUID), and press the carriage-return key.

The resulting object modules are combined with the existing object library at this level of the hierarchy. Existing modules with the same name are replaced and new modules are added to the hierarchy.

A deck can be expanded only if its expand attribute is YES (as specified when the deck was created). If non-expandable decks are selected for the build, PPE ignores the non-expandable selections unless all decks selected are non-expandable. In that case, it issues an informative message because no build is performed.

A failed build is one in which the compiler detects one or more errors. If the build fails, PPE executes a DISPLAY_BUILD_ERRORS command, which displays the Build Errors screen. Otherwise, if the build succeeds, PPE displays the Run Files screen. Also, if the build succeeds and the Run Automatically option is selected on the Tailor Options screen, the product is executed after the build.

Building Changed Decks

The build process for building changed decks is the same as described previously for building selected decks. The only differences are that no deck selection is performed and only changed decks are built.

PPE considers a deck to be changed if it or any deck it references resides at the lowest level of your environment catalog hierarchy and has changed since the last build. A deck is considered changed if it has been explicitly selected on an EDIT_DECK command from the Deck List screen. A deck edited by using SELECT_DECK or EDIT_DECK while editing another deck or while outside the PPE is not detected as having been changed by PPE.

To build all changed expandable decks, do either of the following:

- Press the function key labeled Build, or
- Move the cursor to the home line, type BUILD_CHANGED_DECKS (or BUICD or BUILD), and press the carriage-return key.

Correcting Errors

If a build detects one or more compilation errors, PPE displays the Build Errors screen at termination of the build operation. You can also display the Build Errors screen by using the DISPLAY_BUILD_ERRORS command.

To go to the Build Errors screen, perform these steps:

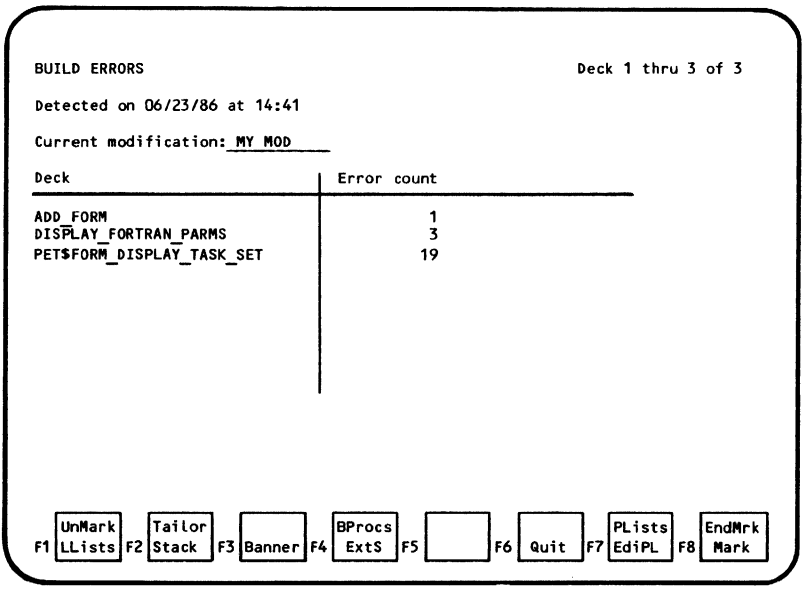
1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using BACK one or more times. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object DECKS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

2. Request the Build Errors screen, by doing either of the following:

   - Press the function key labeled Errors, or
   - Move the cursor to the home line, type DISPLAY_BUILD_ERRORS (or DISBE or ERRORS), and press the carriage-return key.

The Build Errors screen appears similar to the following:



The Build Errors screen shows the following information:

- The time and date of the build.
- The name of the modification to which any editing changes are assigned.
- A list of the decks in which compilation errors were detected and the number of errors in each deck.

To correct the errors in a deck, you must edit the deck text. To edit a deck, perform the following steps:

1. Position the cursor on the deck to be edited.
2. Request editing by doing either of the following:
   - Press the function key labeled Edit, or
   - Move the cursor to the home line, type EDIT_DECK (or EDID or EDIT), and press the carriage-return key.

When a deck is selected for editing, PPE checks that the deck has been extracted to the level. If it has not, PPE extracts the deck if possible. PPE then begins an editor session. The text displayed is the source text of the deck with the compiler diagnostics inserted at the points in the text where the errors were detected.

The first diagnostic is highlighted. As you correct errors in the source text, use the PPE the provided functions to delete the corresponding diagnostics and highlight the next diagnotics. The functions provided by PPE for deleting and highlighting diagnostics are DELETE_IDENTICAL_ERRORS, LOCATE_NEXT_ERROR, and SKIP_LINE_ERRORS. See chapter 4, PPE Quick Reference, for information about these functions.

The following specialized PPE operations are available when editing source text with interwoven diagnostics:

- Display the online manual explanation of a diagnostic.
- Locate a diagnostic:
- Locate the next diagnostic.
- Locate the next source line containing diagnostics associated with it.
- Remove all diagnostics identical to the highlighted diagnostic.
- Display the number of diagnostic messages remaining in the deck.

Use only the functions DELETE_IDENTICAL_ERRORS, LOCATE_NEXT_ERROR, and SKIP_LINE_ERRORS to change diagnostic lines. If you change a diagnostic line by any other means, PPE may not recognize the the changed diagnostic line, and may not remove the line from your source.

Displaying a Diagnostic Explanation

The compiler online manual contains information that can help you correct compiler diagnostics.

To display this information:

1. Position the cursor on the diagnostic message to be explained. (If the cursor is not on a diagnostic message, the highlighted diagnostic message is explained.)

2. Request the information by doing either of the following:

   - Press the function key labeled Assist, or

   - Move the cursor to the home line, type EXPLAIN_ERROR_MESSAGE (or EXPEM or ASSIST),and press the carriage-return key.

Locating Diagnostic Messages

When PPE locates a diagnostic message, the diagnostic message is highlighted, positioning the cursor as close as possible to the error in the source that caused the diagnostic message.

To locate and highlight the next diagnostic message and delete the currently highlighted diagnostic message, do either of the following:

- Press the function key labeled NxtErr, or

- Move the cursor to the home line, type LOCATE_NEXT_ERROR, (or LOCNE, or NXTERR), and press the carriage-return key.

To remove all diagnostic messages associated with the same source line as the currently highlighted diagnostic message, position the cursor at the location of the next error, and highlight the associated diagnostic message:

- Press the function key labeled Skip, or
- Move the cursor to the home line, type SKIP_LINE_ERRORS (or SKILE, or SKIP), and press the carriage-return key.

To delete all diagnostic messages identical to the highlighted diagnostic message:

- Press the function key labeled DelIE, or
- Move the cursor to the home line, type DELETE_IDENTICAL_ERRORS (or DELIE), and press the carriage-return key.

Displaying the Diagnostic Message Count

At any time during the editing session, you can display the number of diagnostic messages remaining in the deck.

To display the diagnostic message count for the deck:

- Move the cursor to the home line, type DISPLAY_REMAINING_ERROR_COUNT (or DISREC), and press the carriage-return key.

Executing the Product

After all compilation errors have been corrected and the product has been built successfully, the next step is to execute the product.

The product can be run with varying parameter lists. If execution errors are found, it can be executed under control of the Debug utility to find the cause of the errors. You would then edit the source decks and rebuild the product before testing product execution again.

If you have selected the Run Automatically option on the Tailor Options screen, the product is run after each successful build. Otherwise, you must initiate each run using the following procedure.

To run the product:

1. Go to the Run Files screen. (The Run Files screen is automatically displayed after a successful build or after executing the RFiles function from the Deck List screen.)

   The Run Files screen appears similar to the following:

```
RUN FILES                                          Run file 1 thru 2 of 2

Run purpose: X NORMAL     __INTERACTIVE DEBUG

Run parameters:
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________

Run file
_____________________

$LOADMAP
$TERMINAL_OUTPUT

F1 Banner Run  F2 Tailor Print  F3 Export Import  F4 BProcs LLists  F5 Delete Undo  F6 Quit  F7 PLists EdiPL  F8 Stacks
```

2. Select a Run purpose (NORMAL, INTERACTIVE DEBUG). Type a non-blank character before the selected purpose.

3. Type the desired parameter list in the field under the label:

   Run parameters:

   When execution begins, PPE passes the parameter list to the starting procedure of the product.

4. You may want to check that the correct program library list is specified on the Library Lists screen. (The DISPLAY_LIBRARY_LISTS command takes you to the Library Lists screen; using Back returns you to the Run Files screen.)

5. You may also want to check and possibly change the current set of program attributes. For example, the default load map options are none, in which case, no load information is written to the $LOADMAP file.

   To display the current program attribute set:

   a. Move the cursor to the home line, type DISPLAY_PROGRAM_ATTRIBUTES (or DISPA), and press the carriage-return key.

   b. After looking at the attribute display, press the carriage-return key to return to PPE.

   To display the program attribute parameters:

   a. Move the cursor to the home line, type DISPLAY_COMMAND_INFORMATION SET_PROGRAM_ATTRIBUTES (or DISCI SETPA), and press the carriage-return key.

   b. After looking at the parameter display, press the carriage-return key to return to PPE.

   To change the program attribute set:

   - Move the cursor to the home line, type SET_PROGRAM_ATTRIBUTES (or SETPA) followed by the desired attribute parameters, and press the carriage-return key.

   For example, to change the load map options to all, type:

   - set_program_attributes, load_map_options=all

6. Request the run, by doing either of the following:

   - Press the function key labeled Run, or
   - Move the cursor to the home line, type RUN, and press the carriage-return key.

PPE loads the product from the object library, satisfying external references in the following order:

- In the PPE object library at the current level.
- In the PPE object libraries at higher levels of the hierarchy
- In the object libraries specified in the program library list entries on the Library Lists screen in the order listed.
- In the object libraries listed in the job library list in the order listed.

If the LOAD_MAP_OPTIONS program attribute is not NONE, PPE writes a load map to file $LOADMAP, which can be viewed later.

After loading, product execution begins at the starting procedure specified on the Tailor Options screen. The product reads from its input files and writes to its output files, the same as it would outside of PPE. The only change is that anything written to $OUTPUT is also written to a file called $TERMINAL_OUTPUT for later viewing.

If the Run purpose is INTERACTIVE DEBUG, PPE executes the product under control of the Full-Screen Debug utility, using its full-screen interface.

When you execute the Run function, PPE searches every object library in your hierarchy. The loader produces a warning level error message if it encounters an empty object library. Since PPE sets the initial termination error level for a job to WARNING, an empty library might cause your run to terminate prematurely.

You can change the termination error level with the SCL SET_PROGRAM_ATTRIBUTES command. The recommended settings are WARNING and ERROR. The following lists the merits of each choice:

ERROR — An empty object library does not terminate a run. However, PPE does not inform you when any warning level error messages are produced, and the run might terminate unexpectedly. Also, you are not informed if an empty object library is encountered, even if none should be empty.

WARNING — Any WARNING level error message terminates a run. Therefore, an empty object library causes a run to terminate.

Regardless of the termination error level setting for your job, all errors are written to the $LOADMAP file.

At run completion, all PPE files generated by the run are listed under the heading Run file. To view the contents of any of these files, perform these steps:

1. Position the cursor on the file.
2. Display the file text by doing either of the following:
   - Press the function key labeled Edit, or
   - Move the cursor to the home line, type EDIT_RUN_FILE (or EDIRF or EDIT), and press the carriage-return key.

You can print any of the list files on the Run Files screen. To do so, perform the following steps:

1. Position the cursor on the file.
2. Request printing, by doing either of the following:
   - Press the function key labeled Print, or
   - Move the cursor to the home line, type PRINT_RUN_FILE (or PRIRF or PRINT), and press the carriage-return key.

When PPE executes the PRINT_RUN_FILE command, it first attempts to execute an SCL command of the following form:

```
PRINT lfn
```

where lfn is the file path of the file to be printed. If no PRINT command exits, PPE executes the standard NOS/VE PRINT_FILE command with default settings.

Transmitting Code

After you have completed the testing of code at the lowest level of the environment catalog hierarchy, you are ready to transmit it to the next higher level of the hierarchy.

Transmittal always moves the transmitted decks up one level in the hierarchy. It removes the transmitted decks from the source library at the lowest level of the environment catalog hierarchy and puts them in the source library at the next higher level. The code is then accessible to others whose PPE levels are linked to the higher level to which the code has been transmitted.

If all decks to which a modification applies are transmitted, the modification can no longer be used at the lower level. To continue using the modification, you must extract at least one of the decks to which the modification applies.

To transmit one or more decks:

1. Go to the Deck List screen. If the Deck List screen is on the screen stack, you can return there by using BACK one or more times. If not, you must go to the Environment Description screen or the Modification List screen and perform one of the following:

   - Position the cursor on the SCU object DECKS and press the function key labeled SCUObj, or
   - Position the cursor on the SCU object DECKS and press the carriage-return key, or
   - Move the cursor to the home line, type DISPLAY_SCU_OBJECT_LIST DECK (or DISSOL D or SCUOBJ D), and press the carriage-return key.

2. Mark the decks to be transmitted (as described earlier under Marking Objects in a List).

   If you mark no decks, PPE transmits the deck under the cursor.

3. Transmit the deck (or decks) by doing either of the following:

   - Press the function key labeled TraS, or
   - Move the cursor to the home line, type TRANSMIT_SOURCE (or TRAS), and press the carriage-return key.

To transmit a deck, PPE copies the deck and attempts to combine it with the source library at the next higher level of the hierarchy. If the deck is not already in the higher source library, PPE adds it to the library. If the deck is already in the higher source library, PPE checks whether the interlocks match.

If the interlocks in the deck headers do not match, the deck transmittal fails. If the interlocks match, the lower deck replaces the higher deck having the same name. After the deck has been transmitted to the higher library, PPE deletes it from the lower library.

File Cycle Management Within the Environment Catalog

When you transmit source, a new high cycle of the source library within your environment catalog is created. The low cycles allow you to recover from catastrophic events. However, you can easily use up a large amount of file space. To avoid using excess file space, you should frequently delete the low cycles of your source library. You can use EDIT_CATALOG from the home line of your environment catalog to do this.

When you transmit source, the object modules in the lowest level of your environment catalog hierarchy that are built by that source should be deleted from the highest cycle of $OBJECT_LIBRARY on the Build Files screen. The CREATE_OBJECT_LIBRARY utility can be used from the home line of the Build Files screen to do this.

Also, after you transmit, notify the owner of the next higher level in your hierarchy to do a build using the transmitted decks. This keeps the $OBJECT_LIBRARY at the higher level of the hierarchy up to date, incorporating your transmitted source code into the object code at that level.

Changes made to the source library are not available to other tasks or PPE sessions at lower levels in the hierarchy until you exit the PPE session in which the changes occurred, or until the changed decks are transmitted to the next higher level in the hierarchy.

PPE Quick Reference 4

This chapter contains concise descriptions of PPE screens and PPE commands presented in alphabetical order for quick reference.

PPE Screens

This section describes all PPE screens in alphabetical order by screen name. Each description lists the means of accessing the screen, the commands available from the screen, and the general appearance of the screen.

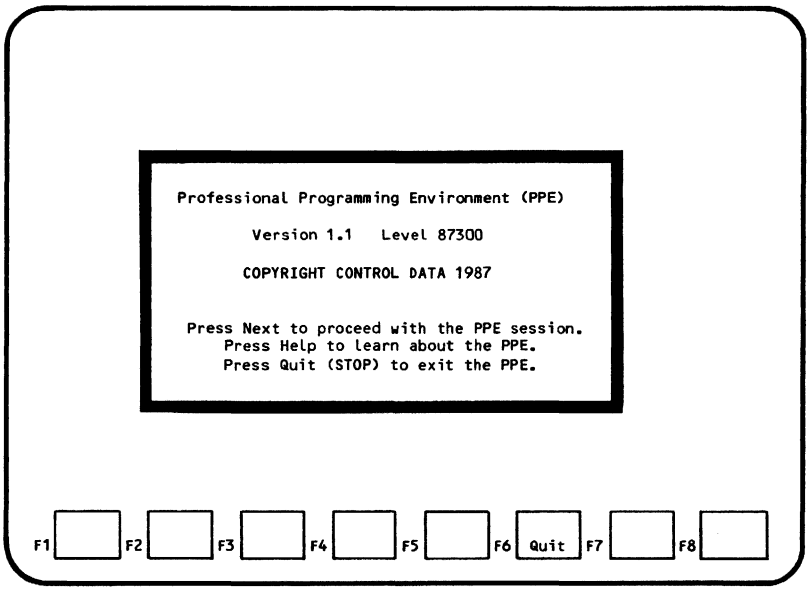
Banner

Purpose: Displays the name, version, and level of PPE with the copyright notice and instructions for use. Also, this screen provides access to the online manual introduction to PPE.

To access: Begin a PPE session, or
Use DISPLAY_BANNER (Banner or DISB) from any screen except the Screen Stack screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
QUIT_SAVE
REQUEST_HELP

Display:



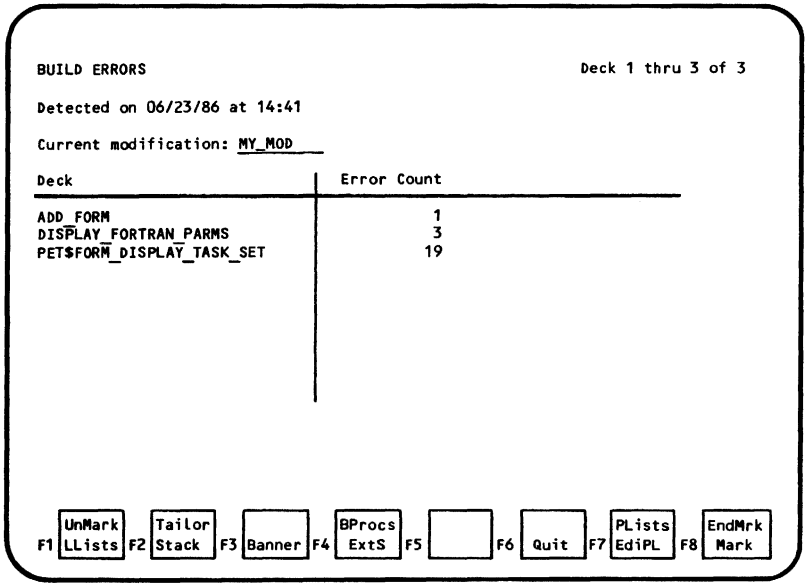
Build Errors

Purpose: Displays a list of decks which contain build errors and enables access to the decks for correction of their errors.

To access: Execute a build that fails, or
Use DISPLAY_BUILD_ERRORS (Errors or DISBE) from the Deck List screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
BEGIN_MARK
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_TAILOR_OPTIONS
EDIT_DECK
EDIT_PARAMETER_LIST
END_MARK
EXTRACT_SOURCE
HOME_CURSOR
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REMOVE_MARK
REQUEST_HELP

Display:



Current modification:
This field displays the name of the modification to which any editing changes made from this screen are assigned. You can change the field value by typing over the current value.

If the specified modification does not exist at the lowest level of the PPE environment catalog hierarchy, you cannot edit any decks. You must choose a modification that exists at the lowest level of the hierarchy to edit a deck.

Deck / Error Count:
Lists the names of the decks in which errors were detected during the last build and the number of errors in each. You select the deck to be edited by positioning the cursor on the deck name.

Build Files

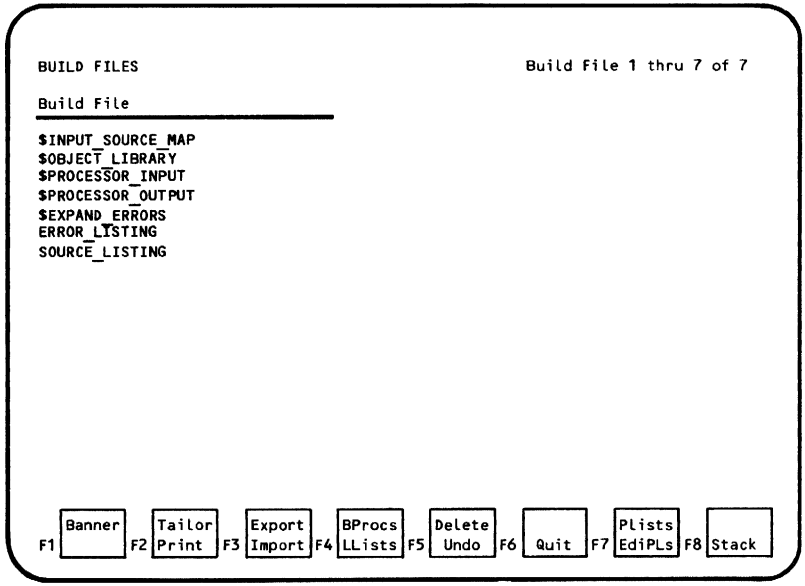
Purpose: Lists output files from the last build and allows viewing of their contents.

To access: Use DISPLAY_BUILD_FILES (BFiles or DISBF) from the Deck List screen.

Commands available:

BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
DELETE_BUILD_FILE
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_TAILOR_OPTIONS
EDIT_BUILD_FILE
EDIT_PARAMETER_LIST
EXPORT_BUILD_FILE
HOME_CURSOR
IMPORT_BUILD_FILE
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
PRINT_BUILD_FILE
QUIT_SAVE
REQUEST_HELP
UNDO_LAST_DELETE

Display:



Build File:
Lists the names of the build files produced by the last successful build. (The build files produced are those selected by the parameter list used by the build.) You can also import build files to this list.

You can delete, export, print, or view a build file in the list by positioning the cursor on the file name and executing the appropriate command.

File cycles are not supported. When you delete a file, all cycles of the file are deleted.

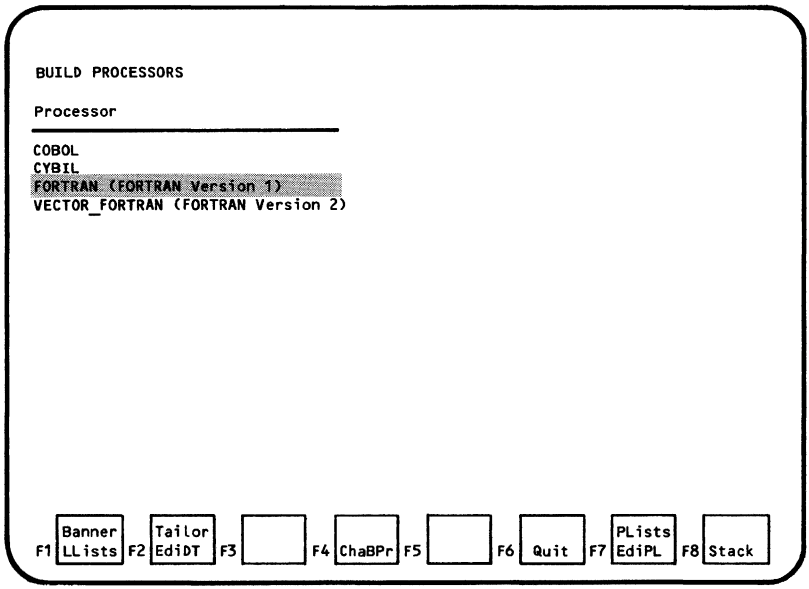
Build Processors

Purpose: Displays the list of build processors supported by PPE and enables editing of the deck template and parameter list for each processor.

To access: Use DISPLAY_BUILD_PROCESSORS (BProcs or DISBP) from any screen except the Banner and Screen Stack screens.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CHANGE_BUILD_PROCESSOR
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_TAILOR_OPTIONS
EDIT_DECK_TEMPLATE
EDIT_PARAMETER_LIST
HOME_CURSOR
QUIT_SAVE
REQUEST_HELP

Display:



Processor

Lists the names of the build processors supported by PPE. The currently selected build processor for the PPE level is highlighted. It is the build processor that executes all subsequent builds.

Each build processor has a deck template that can be used when creating decks. You can select the deck template to be edited by positioning the cursor on the build processor name.

NOTE

You can change the highlighted build processor using the CHANGE_BUILD_PROCESSOR command (or ChaBP key). However, it is not appropriate to change the build processor when the level already contains source code written for the previous build processor.

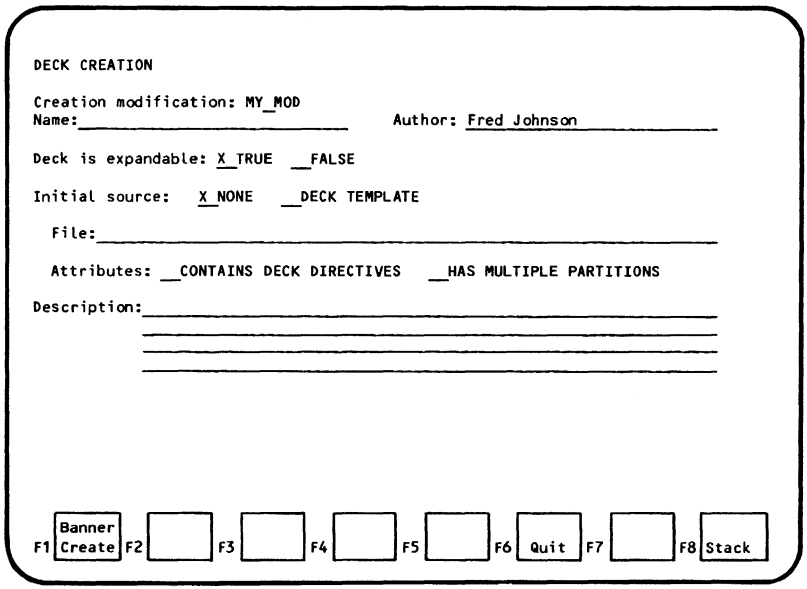
Deck Creation

Purpose: Enables creation of a new deck.

To access: Use CREATE_DECK (Create or CRED) from the Deck List screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
CREATE_DECK
DISPLAY_BANNER
DISPLAY_SCREEN_STACK
EDIT_DECK
HOME_CURSOR
QUIT_SAVE
REQUEST_HELP

Display:



Creation modification:
Field containing the name of the creation modification for the deck. Any source text inserted in the deck when it is created is assigned to the creation modification. The default value is the currently selected modification. You can change the value by typing over it.

If the specified modification does not exist at the lowest level of the PPE environment catalog hierarchy, the deck is not created. PPE issues an informative message and prompts you to acknowledge receipt of the message.

If the specified modification does not exist at any level of the PPE environment catalog hierarchy, PPE asks whether you want the modification created with SCU default values (which do not use the Author value from the Tailor Options screen). If you say yes to the prompt, PPE creates the modification; otherwise, PPE does not create the modification.

Name:
Field in which you type the name of the new deck. The name must follow the rules for SCL names and must not match the name of any existing deck in the PPE hierarchy. PPE notifies you if the name is invalid when you use CREATE_DECK or EDIT_DECK to create the deck.

Author:
Field containing the value stored in the Author field of the deck header. The default value is the Author value from the Tailor Options screen. You can change the value by typing over it.

Deck is expandable:
Selects the expand attribute for the deck: TRUE for a deck that can be expanded directly, FALSE for a deck that is expanded only when it is copied by another deck. You select a value by typing a non-blank character by TRUE or FALSE. The default is TRUE.

Initial source:
The information under this heading describes the source text inserted in the deck when it is created. It can be NONE or the text of the deck template for the build processor or the text copied from a file.

To select the deck template, type non-blank character in the field before DECK TEMPLATE.

To specify a file of text, enter its file path in the space after File:. If you specify a file, you can also specify that it CONTAINS DECK DIRECTIVES or HAS MULTIPLE PARTITIONS.

If you select CONTAINS DECK DIRECTIVES, SCU processes any *DECK directives embedded in the source text and thus, may create multiple decks. (The deck names are specified on the *DECK directives.)

If you select HAS MULTIPLE PARITIONS, SCU converts each end-of-partition delimiter to an embedded *WEOP directive and continues copying text until it reaches the end of the file. (Otherwise, it would stop copying text at the first end-of-partition delimiter.)

Description:
Field in which you can type an optional description of the deck contents to be stored in the deck header. The default for this field is BLANK.

Deck List

Purpose: Lists the decks available at this level so you can select decks for various commands.

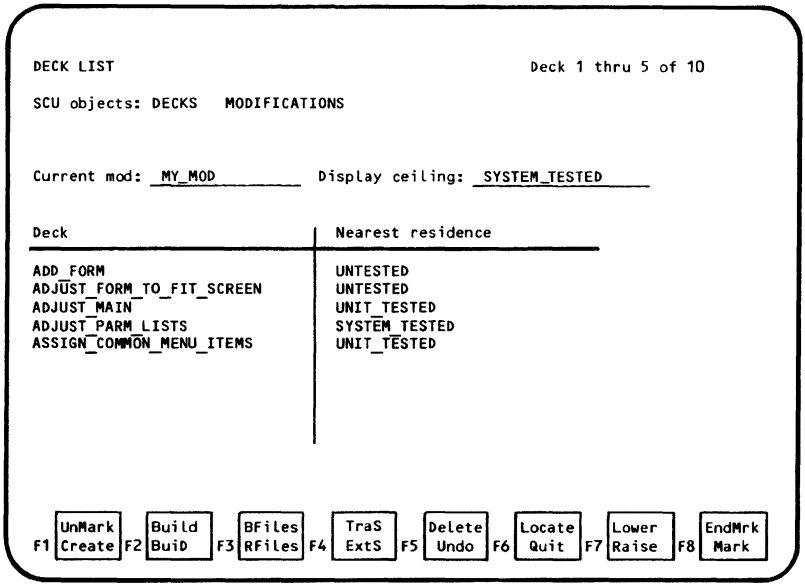
To access: Use DISPLAY_SCU_OBJECT_LIST DECK (SCUObj D or DISSOL D) from the Environment Description or Modification List screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
BEGIN_MARK
BUILD_CHANGED_DECKS
BUILD_DECKS
CLEAR_SCREEN
CREATE_DECK
DELETE_DECK
DISPLAY_BANNER
DISPLAY_BUILD_ERRORS
DISPLAY_BUILD_FILES
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_RUN_FILES
DISPLAY_SCREEN_STACK
DISPLAY_SCU_OBJECT_LIST
DISPLAY_TAILOR_OPTIONS
EDIT_DECK
EDIT_PARAMETER_LIST
END_MARK
EXTRACT_SOURCE
HOME_CURSOR
LOCATE_DECK
LOWER_DISPLAY_CEILING
MOVE_TO_BOTTOM

Commands available: (continued)

MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
RAISE_DISPLAY_CEILING
REMOVE_MARK
REQUEST_HELP
TRANSMIT_SOURCE
UNDO_LAST_DELETE

Display:



SCU objects:
Line listing the kinds of SCU objects that can be displayed.

To go to the Modification List screen, position the cursor on MODIFICATIONS and press the carriage-return key. (If the Modification List screen is on the screen stack, PPE executes BACK_TO_PREVIOUS_CONTEXT commands until the Modification List screen is displayed.)

Display ceiling:
Field displaying the level name of the current display ceiling. The display ceiling is the highest level of the PPE hierarchy whose decks are listed on the screen. You can change the display ceiling using the Raise and Lower functions.

Current mod:
This field displays the name of the modification to which any editing changes made from this screen are assigned. You can change the field value by typing over the current value.

If you change the Current mod: field value, you must choose a modification that exists at the lowest level of the hierarchy. PPE informs you if the specified modification does not exist at the lowest level of the environment catalog hierarchy, and you must choose another modification.

Deck / Nearest Residence:
List of decks in the PPE source libraries at or below the current display ceiling level. The Nearest Residence is the lowest PPE level that has a copy of the deck. You can select a deck from the list by positioning the cursor on the deck name or you can select decks by marking one or more ranges of decks.

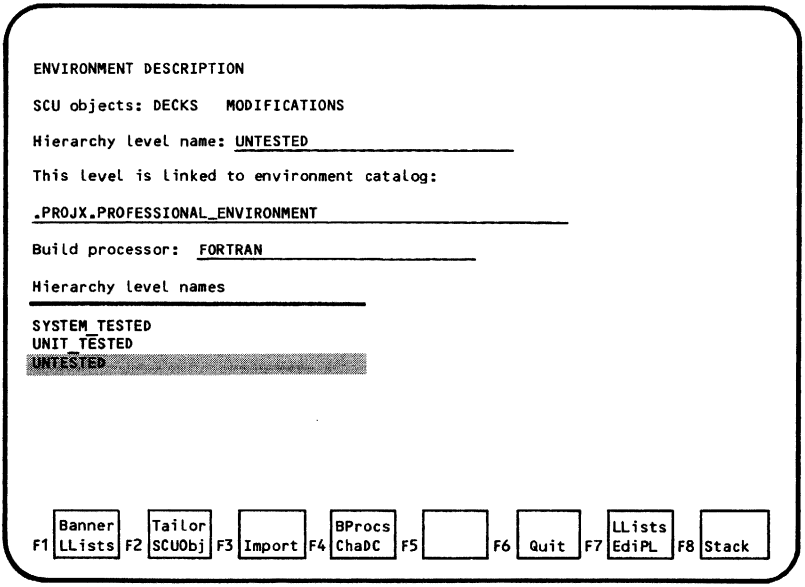
Environment Description

Purpose: Displays information about your PPE level within the PPE hierarchy.

To access: Press the carriage-return key at the Banner screen. Otherwise, use the Screen Stack or the BACK function key.

Commands available:
CHANGE_DISPLAY_CEILING
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_SCU_OBJECT_LIST
DISPLAY_TAILOR_OPTIONS
EDIT_PARAMETER_LIST
HOME_CURSOR
IMPORT_SOURCE_LIBRARY
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REQUEST_HELP

Display:



SCU objects:
Line listing the kinds of SCU objects that can be displayed. To go to the Deck List screen, position the cursor on DECKS and press the carriage-return key. To go to the Modification List screen, position the cursor on MODIFICATIONS and press the carriage-return key.

Hierarchy level name:
Field in which you assign a name to your PPE level. The name must follow SCL naming rules and be unique in the PPE hierarchy.

This level is linked to environment catalog:
Field in which you may type the catalog path of a PPE catalog to which your PPE level is to be linked. Your PPE level becomes the lowest level in a hierarchy consisting of your PPE level, the specified PPE catalog, the catalog to which the specified catalog is linked (if any), and so forth up an arbitrary number of levels.

Build processor:
Field displaying the name of the build processor used by your PPE level. This field defaults to FORTRAN. You can change the name in this field by typing over it.

Hierarchy level names:
List of the level names in the PPE hierarchy with your PPE level name at the bottom. The current display ceiling is highlighted.

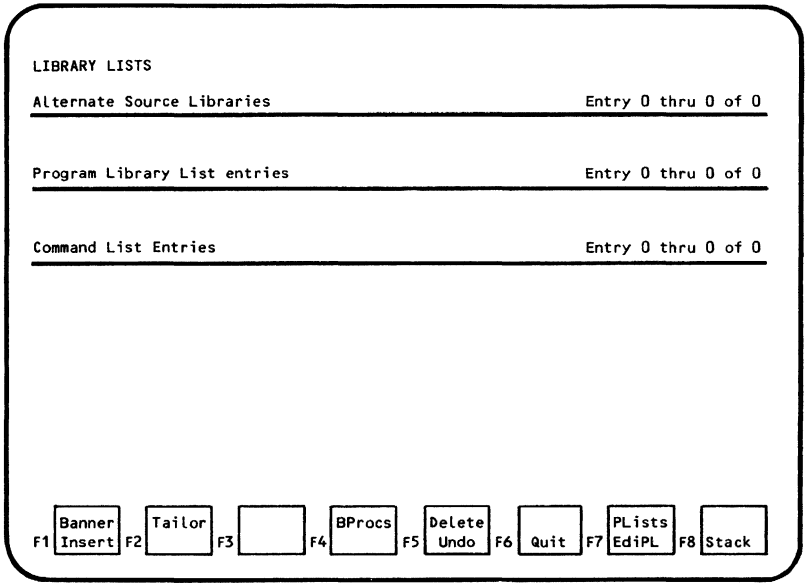
Library Lists

Purpose: Displays and enables editing of these lists: the alternate source library list, the program library list, and the command list.

To access: Use DISPLAY_LIBRARY_LISTS (LLists or DISLL) from the Build Errors, Build Files, Build Processors, Deck List, Environment Description, Modification List, Run Files, and Tailor Options screens.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
DELETE_LIST_ENTRY
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_TAILOR_OPTIONS
EDIT_PARAMETER_LIST
HOME_CURSOR
INSERT_LIST_ENTRY
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REQUEST_HELP
UNDO_LAST_DELETE

Display:



Alternate Source Libraries:
List of SCU source library files in the order the files are searched by a build. When a build expands a deck, it processes each copy directive in the deck by copying the specified deck into that location. The decks to be copied are searched for in this order:

1. The source libraries in the PPE hierarchy, from lowest level to highest level.

2. The source libraries in the Alternate Source Libraries list. These libraries are searched in the order presented in the list.

Program Library List entries:
List of object library files in the order that the files are searched by the loading process. When the product is loaded, its external references are satisfied by loading the modules containing the referenced entry points. The entry point search proceeds in this order:

1. The object libraries in the PPE hierarchy, from lowest level to highest level.

2. The program library list entries listed on this screen.

3. The job library list entries.

Command List Entries:
List of catalogs and object library files to be added as entries in the SCL command list. By adding the entries to the SCL command list, the entry points in the object libraries and the files in the catalogs can be executed as SCL commands during the PPE session. The entries are removed from the command list when the PPE session ends. For more information on the command list, see the SCL Language Definition manual.

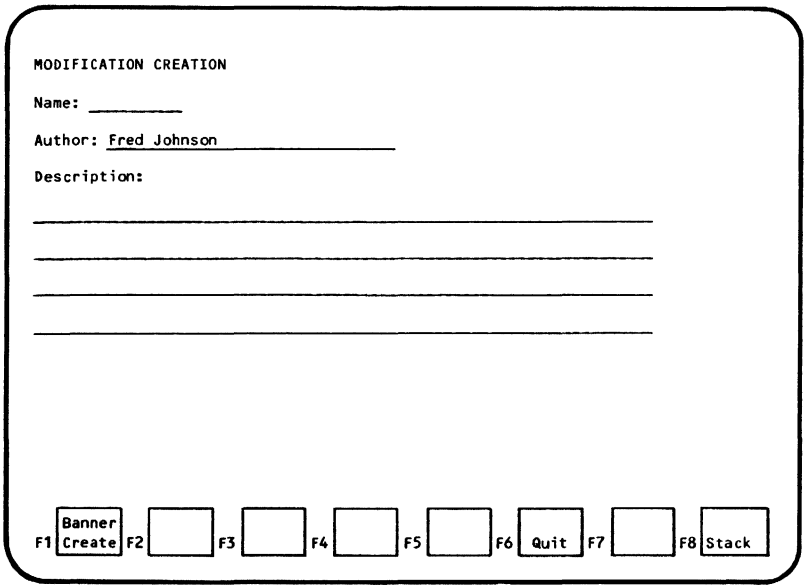
Modification Creation

Purpose: Enables creation of a new modification.

To access: Use CREATE_MODIFICATION (Create or CREM) from the Modification List screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
CREATE_MODIFICATION
DISPLAY_BANNER
DISPLAY_SCREEN_STACK
HOME_CURSOR
QUIT_SAVE
REQUEST_HELP

Display:



Name:
Field in which you type the name of the new modification. The name must follow SCL naming rules and be unique within the PPE hierarchy. If you select a modification name that already exits in your hierarchy when you use CREATE_MODIFICATION to create the modification, PPE so informs you. You must then select a new name to create the modification.

Author:
Field containing the value stored in the Author field of the modification header. The default value is the Author value from the Tailor Options screen. You can change the value by typing over it.

Description:
Field in which you can type an optional description of the purpose of the modification. The description is stored in the modification header.

Modification List

Purpose: Lists the available modifications.

To access: Use DISPLAY_SCU_OBJECT_LIST MODIFICATION (SCUObj M or DISSOL M) from the Environment Description or Deck List screen.

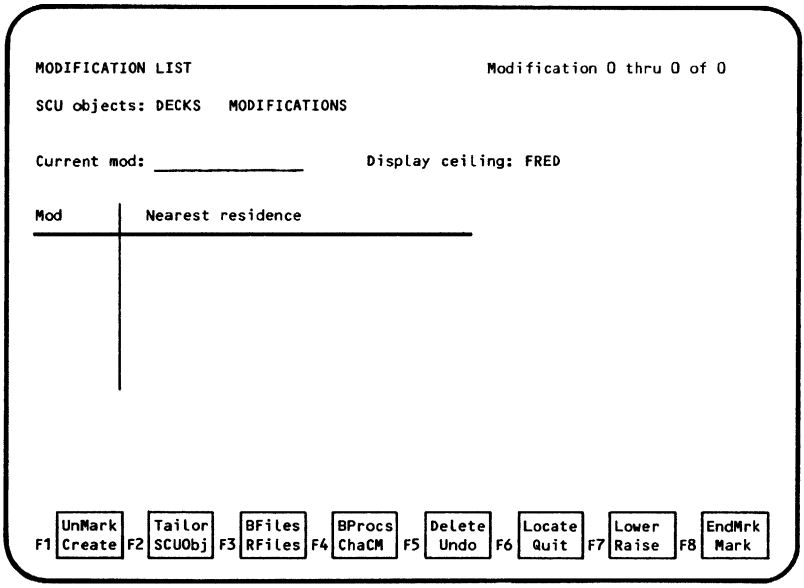
Use BACK_TO_PREVIOUS_CONTEXT from the Modification Creation screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
BEGIN_MARK
CHANGE_CURRENT_MODIFICATION
CLEAR_SCREEN
CREATE_MODIFICATION
DELETE_MODIFICATION
DISPLAY_BANNER
DISPLAY_BUILD_FILES
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_RUN_FILES
DISPLAY_SCREEN_STACK
DISPLAY_SCU_OBJECT_LIST
DISPLAY_TAILOR_OPTIONS
EDIT_PARAMETER_LIST
END_MARK

Commands available: (Continued)

HOME_CURSOR
LOCATE_MODIFICATION
LOWER_DISPLAY_CEILING
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
RAISE_DISPLAY_CEILING
REMOVE_MARK
REQUEST_HELP
UNDO_LAST_DELETE

Display:



SCU objects:
Line listing the kinds of SCU objects that can be displayed. To go to the Deck List screen, position the cursor on DECKS and press the carriage-return key.

Display ceiling:
Field displaying the level name of the display ceiling. The display ceiling is the highest level of the PPE hierarchy whose modifications are listed on the screen. You can change the display ceiling by typing over the value in this field.

Current mod:
Field displaying the name of the currently selected modification. You can change the field value by typing over the current value.

If the specified modification does not exist at the lowest level of the PPE environment catalog hierarchy, PPE issues a message informing you of this. You must specify a modification that resides at the lowest level of the hierarchy.

Mod / Nearest Residence:
List of modifications in the source libraries in the PPE hierarchy up through the display ceiling level. The Nearest Residence is the lowest PPE level in which the modification is defined. You can select a modification from the list by positioning the cursor on the modification name or you can can select more than one modification by marking ranges of names in the list.

Parameter List

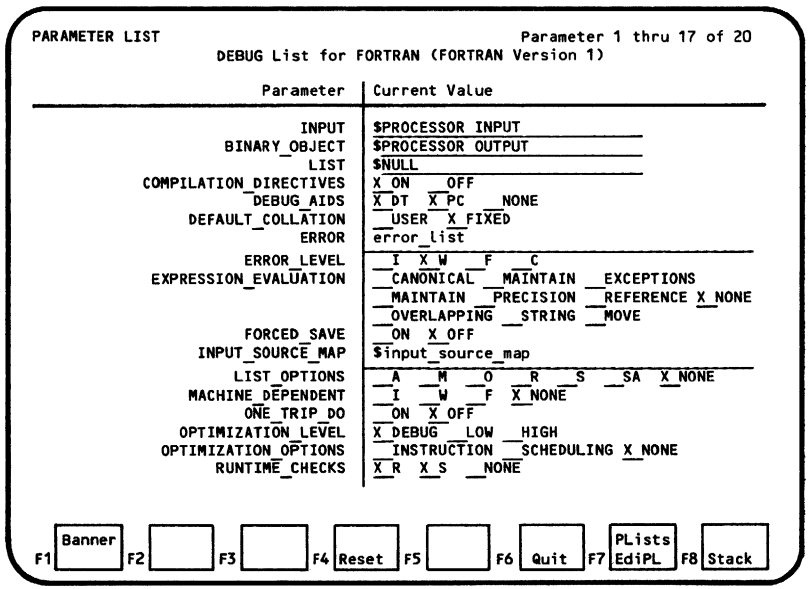
Purpose: Enables viewing and editing of a parameter list for the current build processor.

To access: Use CREATE_PARAMETER_LIST (Create or CREPL) from the Parameter List Library screen.

Use EDIT_PARAMETER_LIST (Edit or EDIPL) from the Parameter List Library, Tailor Options, Build Processor, or Environment Description screens.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CHANGE_TO_DEFAULT
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
EDIT_PARAMETER_LIST
HOME_CURSOR
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REQUEST_HELP

Display:



Parameter / Current Value:

Lists the parameter names and current parameter values. The file name is shown for file parameters; the available options are listed for option parameters. To select an option, type a non-blank character in the blank before the option.

To read about a parameter and its options, position the cursor on the parameter and use the REQUEST_HELP command.

File paths can be specified relative to the build files catalog of the PPE level or as the complete path. For example, the file path for a file in your master catalog would begin with $USER or the family name (such as :NVE).

Because PPE must control the input and output files generated during a build, it defines the following special files for its use:

$INPUT_SOURCE_MAP

PPE requires that each deck expansion write to this file the information required for mapping build errors to deck text and to provide for Full-Screen Debugging. Only specify $INPUT_SOURCE_MAP for the INPUT_SOURCE_MAP parameter.

$PROCESSOR_INPUT

PPE writes the source text expanded from decks to this file and then requires that the file be the input file to the build processor.

$PROCESSOR_OUTPUT

PPE requires that the build processor write its object modules to this file so that it can then combine the modules with the $OBJECT_LIBRARY file at this level.

The files created by the run are created relative to the run subcatalog. You can copy files to the run subcatalog and copy files from the run subcatalog using IMPORT_RUN_FILE and EXPORT_RUN_FILE, respectively.

PPE does not support cycles. When you delete a file, all cycles are deleted.

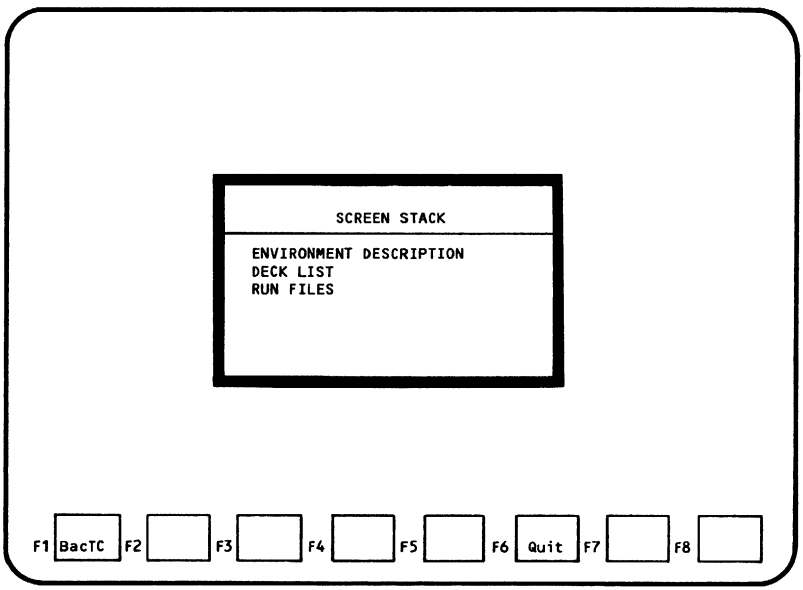
Screen Stack

Purpose: Enables direct access to any PPE screen in the stack.

To access: Use DISPLAY_SCREEN_STACK (Stack or DISSS) from any screen except the Banner screen.

Commands available:
BACK_TO_CONTEXT
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
MOVE_TO_FIRST
MOVE_TO_BOTTOM
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REQUEST_HELP

Display:



The list of screen names shows the path you have taken through the hierarchy of PPE screens. The first name is the screen at the top of the hierarchy; the last name is that of the screen displayed before the Screen Stack.

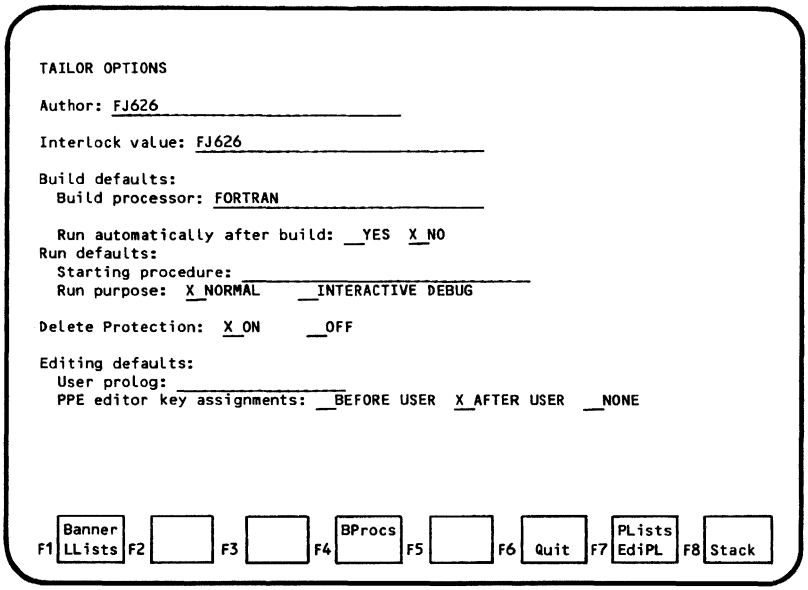
Tailor Options

Purpose: Enables viewing and selection of the default options for PPE operations.

To access: Use DISPLAY_TAILOR_OPTIONS (Tailor or DISTO) from any screen except the Banner screen and the Screen Stack screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
EDIT_PARAMETER_LIST
HOME_CURSOR
QUIT_SAVE
REQUEST_HELP

Display:



Author:
Field containing the default author value stored in deck and modification headers. The initial value is your user name. You can change the value by typing over it.

Interlock value:
Field containing the value stored as the interlock value when you extract a deck. The initial value is your user name. You can change the value by typing over it.

Build defaults:
These fields contain default values used when building the product.

Build processor:
The name of the build processor. It can be COBOL, CYBIL, FORTRAN, or VECTOR_FORTRAN. You can change the build processor by typing over the existing name, striking out all of its characters.

Run automatically after build:
Option that determines whether PPE executes the product immediately after it builds it. The initial value is NO. To select YES, type a non-blank character in the field before YES.

Run defaults:
These fields contain default values used when executing the product.

Starting procedure:
Name of the entry point at which PPE is to begin product execution. The default is the same as the default outside of PPE, that is, the last transfer symbol loaded. All characters, including leading, embedded, and trailing blanks, are significant in this field.

Run purpose:
Indicates the default value determining how the product should be executed. (It can be changed for each run.)

NORMAL	Execute without control by the Full Screen Debug utility.
INTERACTIVE DEBUG	Execute under control of the Debug utility. The product must have been built with the debug options in its parameter list. The debug options are:

```
OPTIMIZATION_LEVEL=DEBUG
DEBUG_AIDS=DT
INPUT_SOURCE_MAP=$INPUT_SOURCE_MAP
```

Delete protection:
Allows you to select delete protection. Delete protection warns you when objects are about to be physically deleted. Options are:

ON PPE issues a warning before physically deleting an object. You can then veto or confirm the deletion of the object. PPE defaults to ON.

OFF PPE physically deletes objects without allowing you to veto the action.

When you use a DELETE command (for example, DELETE_DECK), the deleted object is logically deleted. This means that the name of the object is removed from the screen, but the object is still physically resident in PPE. Logically deleted objects can be recovered with the UNDO_LAST_DELETE command.

A logically deleted object becomes physically deleted if:

- Delete protection is ON and you confirm the deletion. The Delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
- You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
- Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

A physically deleted object cannot be recovered by invoking the UNDO_LAST_DELETE command. PPE can physically delete only objects that are logically deleted.

Editing defaults:
Fields for specifying an editor prolog to define the editor function keys when you enter the PPE editor. In addition to specifying the editor prolog, you can tell PPE to assign PPE commands to editor function keys before or after executing the editor prolog or you can tell PPE not to assign commands to any function keys.

The Editing defaults: fields are the following:

User prolog:
To define PPE editor function keys using an editor prolog, enter the file path of the editor prolog in the User prolog: field. For example, to define PPE editor function keys using the editor prolog, C721_EDITOR_PROLOG, type the following file path in the User Prolog: field:

```
$user.c721_editor_prolog
```

If you do not specify an editor prolog in this field, PPE uses the editor prolog, SCU_EDITOR_PROLOG, to define the editor function keys. If the file, SCU_EDITOR_PROLOG, does not exist, PPE defines the editor function keys using the standard function key definitions.

PPE editor key assignments:
To tell PPE how to assign PPE commands to editor function keys place a non-blank character in one of the following fields:

BEFORE USER	Causes PPE to assign PPE commands to the editor function keys before executing your editor prolog. First PPE assigns PPE commands to the editor function keys, then the editor prolog assigns commands to the function keys.
AFTER USER	Causes PPE to assign PPE commands to the editor function keys after executing your editor prolog. First the editor prolog assigns commands to the the function keys, then PPE assigns PPE commands to the keys.
NONE	Causes PPE to not assign any PPE commands to the editor function keys.

When you choose BEFORE USER or AFTER USER, both PPE defined and user defined function keys are specified for the PPE editor. Any function key defined by both your user prolog and by PPE commands assumes the definition assigned to the key last. Therefore, to use your editor prolog with the PPE editor prolog to maximum effect, do the following:

- Determine which function keys are assigned PPE commands.
- Define your editor prolog only for function keys not assigned PPE commands.

- Specify the file path of your editor prolog on the User prolog: field (the editor prolog can be any file containing EDIT_FILE function key definitions).

- Choose BEFORE USER or AFTER USER on the PPE editor key assignments: field.

PPE Commands

This section contains individual descriptions of the PPE commands, presented in alphabetical order by command name. Each description gives the following information:

- The purpose of the command.

- The PPE screen from which the command is available.

- The function key label if your terminal has a function key for the command.

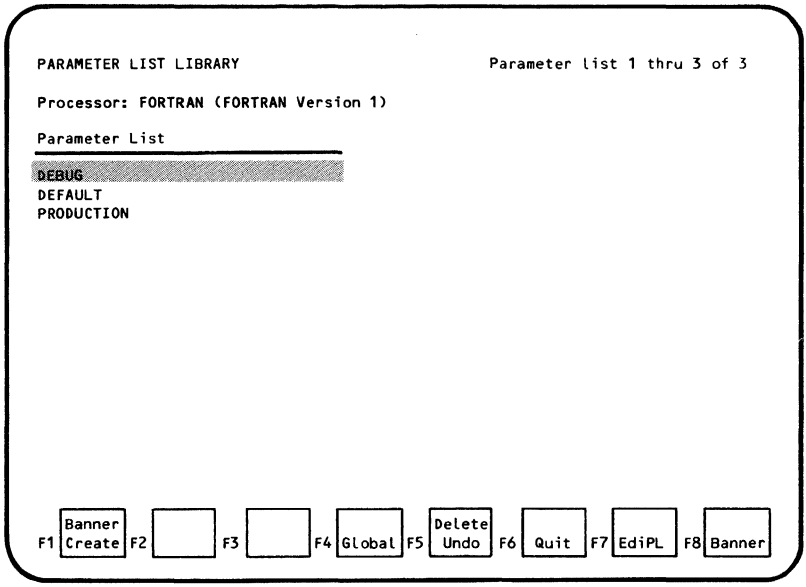
- The format of the command.

The command format lists the command name and parameters as they would be typed on the home line of a PPE screen. The format gives the full and abbreviated names for the command and lists the command parameters. The last parameter of every command is its STATUS parameter. Individual descriptions are given for each parameter except the STATUS parameter.

The STATUS parameter on PPE commands is the same standard STATUS parameter available on all NOS/VE SCL commands. Its use is optional. If you specify the name of an SCL status variable on the STATUS parameter of a command, the completion status of the command is stored in the status variable and execution continues with the next command received. To see the completion status stored in the status variable, you must display the variable. An SCL statement can also reference the contents of a status field. For more information on SCL status variables, see the SCL Language Definition manual.

Note that changes made to the source library are not available to other tasks or PPE seesions at lower levels in the hierarchy until you exit the PPE session in which the changes occurred, or until the changed decks are transmitted to the next higher level in the hierarchy.

Display:



Parameter List:

List of parameter lists stored in the parameter list library for the build processor. The global parameter list is highlighted. The global parameter list is used in subsequent builds.

PPE provides the following parameter lists:

DEBUG	Produces code for use with Debug.
DEFAULT	Uses processor default values for most parameters.
PRODUCTION	Produces code for production use.

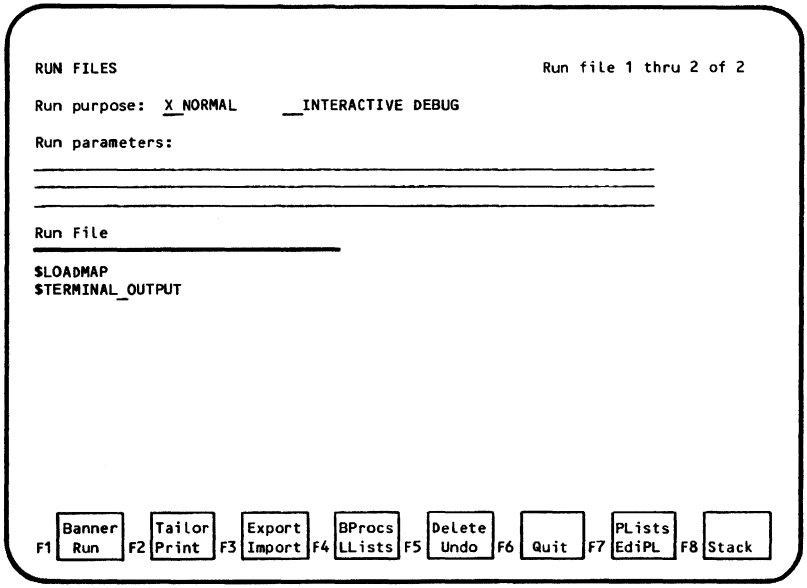
Run Files

Purpose: Enables execution of the product and viewing of the resulting run files.

To access: Execute a successful build, or
Use DISPLAY_RUN_FILES (RFiles or DISRF) from the Deck List or Modification List screens.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
DELETE_RUN_FILE
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
DISPLAY_TAILOR_OPTIONS
EDIT_PARAMETER_LIST
EDIT_RUN_FILE
EXPORT_RUN_FILE
HOME_CURSOR
IMPORT_RUN_FILE
MOVE_TO_BOTTOM
MOVE_TO_FIRST
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
PRINT_RUN_FILE
QUIT_SAVE
REQUEST_HELP
RUN
UNDO_LAST_DELETE

Display:



Run purpose:
Indicates how the product should be executed:

NORMAL	Execute without control by the Debug utility.
INTERACTIVE DEBUG	Execute under control of the Debug utility. The product must have been built with the debug options specified in its parameter list. The debug options are:

```
OPTIMIZATION_LEVEL=DEBUG
DEBUG_AIDS=DT
INPUT_SOURCE_MAP=$INPUT_SOURCE_MAP
```

Run parameters:
Lists the parameters PPE passes to the starting procedure of the product.

Run File:
Lists the files produced by the last run. It may include a load map file ($LOADMAP) and any output written to $OUTPUT ($TERMINAL_OUTPUT).

The files created by the run are created relative to the run subcatalog. You can copy files to the run subcatalog and copy files from the run subcatalog using IMPORT_RUN_FILE and EXPORT_RUN_FILE, respectively.

PPE does not support cycles. When you delete a file, all cycles are deleted.

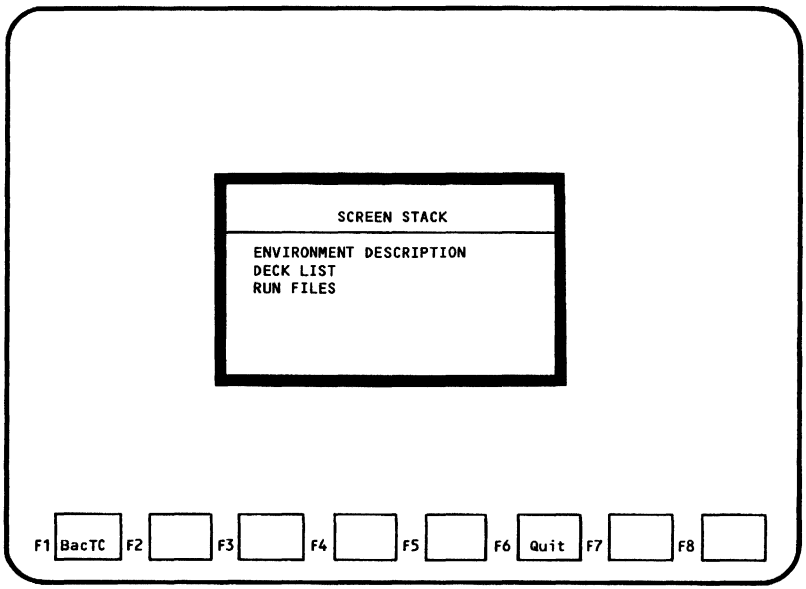
Screen Stack

Purpose: Enables direct access to any PPE screen in the stack.

To access: Use DISPLAY_SCREEN_STACK (Stack or DISSS) from any screen except the Banner screen.

Commands available:
BACK_TO_CONTEXT
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
MOVE_TO_FIRST
MOVE_TO_BOTTOM
MOVE_TO_LAST
MOVE_TO_TOP
PAGE_BACKWARD
PAGE_FORWARD
QUIT_SAVE
REQUEST_HELP

Display:



The list of screen names shows the path you have taken through the hierarchy of PPE screens. The first name is the screen at the top of the hierarchy; the last name is that of the screen displayed before the Screen Stack.

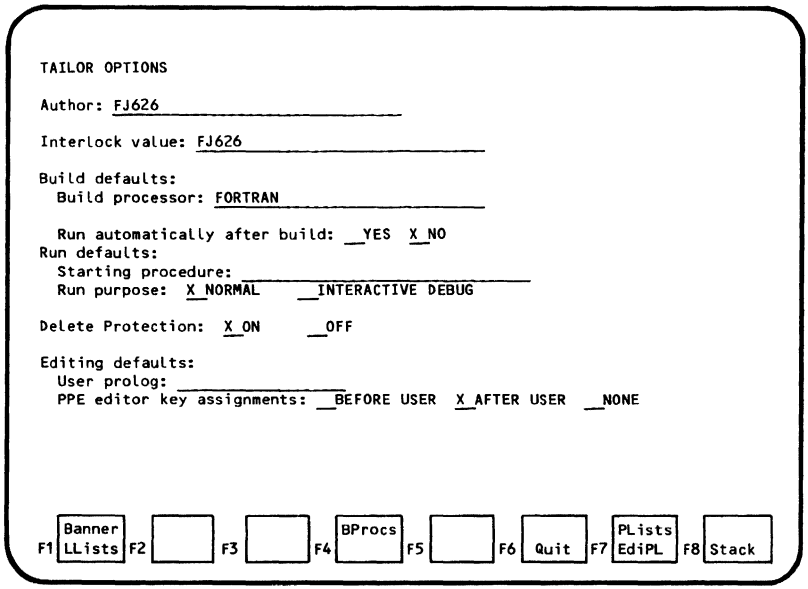
Tailor Options

Purpose: Enables viewing and selection of the default options for PPE operations.

To access: Use DISPLAY_TAILOR_OPTIONS (Tailor or DISTO) from any screen except the Banner screen and the Screen Stack screen.

Commands available:
BACK_TO_PREVIOUS_CONTEXT
CLEAR_SCREEN
DISPLAY_BANNER
DISPLAY_BUILD_PROCESSORS
DISPLAY_LIBRARY_LISTS
DISPLAY_PARAMETER_LIST_LIBRARY
DISPLAY_SCREEN_STACK
EDIT_PARAMETER_LIST
HOME_CURSOR
QUIT_SAVE
REQUEST_HELP

Display:



Author:
Field containing the default author value stored in deck and modification headers. The initial value is your user name. You can change the value by typing over it.

Interlock value:
Field containing the value stored as the interlock value when you extract a deck. The initial value is your user name. You can change the value by typing over it.

Build defaults:
These fields contain default values used when building the product.

Build processor:
The name of the build processor. It can be COBOL, CYBIL, FORTRAN, or VECTOR_FORTRAN. You can change the build processor by typing over the existing name, striking out all of its characters.

Run automatically after build:
Option that determines whether PPE executes the product immediately after it builds it. The initial value is NO. To select YES, type a non-blank character in the field before YES.

Run defaults:
These fields contain default values used when executing the product.

Starting procedure:
Name of the entry point at which PPE is to begin product execution. The default is the same as the default outside of PPE, that is, the last transfer symbol loaded. All characters, including leading, embedded, and trailing blanks, are significant in this field.

Run purpose:
Indicates the default value determining how the product should be executed. (It can be changed for each run.)

NORMAL	Execute without control by the Full Screen Debug utility.
INTERACTIVE DEBUG	Execute under control of the Debug utility. The product must have been built with the debug options in its parameter list. The debug options are:

```
OPTIMIZATION_LEVEL=DEBUG
DEBUG_AIDS=DT
INPUT_SOURCE_MAP=$INPUT_SOURCE_MAP
```

Delete protection:
Allows you to select delete protection. Delete protection warns you when objects are about to be physically deleted. Options are:

ON — PPE issues a warning before physically deleting an object. You can then veto or confirm the deletion of the object. PPE defaults to ON.

OFF — PPE physically deletes objects without allowing you to veto the action.

When you use a DELETE command (for example, DELETE_DECK), the deleted object is logically deleted. This means that the name of the object is removed from the screen, but the object is still physically resident in PPE. Logically deleted objects can be recovered with the UNDO_LAST_DELETE command.

A logically deleted object becomes physically deleted if:

- Delete protection is ON and you confirm the deletion. The Delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
- You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
- Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

A physically deleted object cannot be recovered by invoking the UNDO_LAST_DELETE command. PPE can physically delete only objects that are logically deleted.

Editing defaults:
Fields for specifying an editor prolog to define the editor function keys when you enter the PPE editor. In addition to specifying the editor prolog, you can tell PPE to assign PPE commands to editor function keys before or after executing the editor prolog or you can tell PPE not to assign commands to any function keys.

The Editing defaults: fields are the following:

User prolog:
To define PPE editor function keys using an editor prolog, enter the file path of the editor prolog in the User prolog: field. For example, to define PPE editor function keys using the editor prolog, C721_EDITOR_PROLOG, type the following file path in the User Prolog: field:

```
$user.c721_editor_prolog
```

If you do not specify an editor prolog in this field, PPE uses the editor prolog, SCU_EDITOR_PROLOG, to define the editor function keys. If the file, SCU_EDITOR_PROLOG, does not exist, PPE defines the editor function keys using the standard function key definitions.

PPE editor key assignments:
To tell PPE how to assign PPE commands to editor function keys place a non-blank character in one of the following fields:

BEFORE USER	Causes PPE to assign PPE commands to the editor function keys before executing your editor prolog. First PPE assigns PPE commands to the editor function keys, then the editor prolog assigns commands to the function keys.
AFTER USER	Causes PPE to assign PPE commands to the editor function keys after executing your editor prolog. First the editor prolog assigns commands to the the function keys, then PPE assigns PPE commands to the keys.
NONE	Causes PPE to not assign any PPE commands to the editor function keys.

When you choose BEFORE USER or AFTER USER, both PPE defined and user defined function keys are specified for the PPE editor. Any function key defined by both your user prolog and by PPE commands assumes the definition assigned to the key last. Therefore, to use your editor prolog with the PPE editor prolog to maximum effect, do the following:

- Determine which function keys are assigned PPE commands.
- Define your editor prolog only for function keys not assigned PPE commands.

- Specify the file path of your editor prolog on the User prolog: field (the editor prolog can be any file containing EDIT_FILE function key definitions).

- Choose BEFORE USER or AFTER USER on the PPE editor key assignments: field.

PPE Commands

This section contains individual descriptions of the PPE commands, presented in alphabetical order by command name. Each description gives the following information:

- The purpose of the command.

- The PPE screen from which the command is available.

- The function key label if your terminal has a function key for the command.

- The format of the command.

The command format lists the command name and parameters as they would be typed on the home line of a PPE screen. The format gives the full and abbreviated names for the command and lists the command parameters. The last parameter of every command is its STATUS parameter. Individual descriptions are given for each parameter except the STATUS parameter.

The STATUS parameter on PPE commands is the same standard STATUS parameter available on all NOS/VE SCL commands. Its use is optional. If you specify the name of an SCL status variable on the STATUS parameter of a command, the completion status of the command is stored in the status variable and execution continues with the next command received. To see the completion status stored in the status variable, you must display the variable. An SCL statement can also reference the contents of a status field. For more information on SCL status variables, see the SCL Language Definition manual.

Note that changes made to the source library are not available to other tasks or PPE seesions at lower levels in the hierarchy until you exit the PPE session in which the changes occurred, or until the changed decks are transmitted to the next higher level in the hierarchy.

BACK_TO_CONTEXT

Purpose: Pops the screen stack to display the PPE screen under the cursor.

Available from: Screen Stack screen.

Function key: BacTC

Format:
```
BACK_TO_CONTEXT or BACTC
    STATUS=status_variable
```

Remarks:

- All screens after the selected screen are removed from the stack. For example, if you select the second entry in the stack, the third and all subsequent entries are removed from the stack.

BACK_TO_PREVIOUS_CONTEXT

Purpose: Returns to the preceding screen on the screen stack.

Available from: Any screen except the Environment Description screen.

Function key: Back

Format:
```
BACK_TO_PREVIOUS_CONTEXT or BACTPC or BACK
    STATUS=status_variable
```

BEGIN_MARK

Purpose: Marks the first object in a range of selected objects. The last object in the range is marked using END_MARK. The marked range of objects is then referenced by subsequent commands (such as DELETE_DECK or EXTRACT_SOURCE).

Available from: The Build Errors, Deck List, and Modification List screens.

Function key: Mark

Format:
```
BEGIN_MARK or BEGM or MARK
    STATUS=status_variable
```

BUILD_CHANGED_DECKS

Purpose: Builds all expandable decks that have changed since the last build.

Available from: Deck List screen or a Full-Screen Editor session that originates from within the Deck List or Deck Creation screens.

Function key: Build

Format:
```
BUILD_CHANGED_DECKS or BUICD or BUILD
    STATUS=status_variable
```

Remarks:

- PPE considers a deck to have changed if it or any deck it references resides at the lowest level of your environment catalog hierarchy, and has been edited since the last build.
- When executed during an editing session, this command leaves the Full-Screen Editor, returns to the Deck List screen, and executes the build from there.
- For more information about builds, see the Remarks in the BUILD_DECKS command description.

BUILD_DECKS

Purpose: Builds each selected deck (if the deck is expandable).

Available from: Deck List screen.

Function key: BuiD

Format:
```
BUILD_DECKS or BUID
    STATUS=status_variable
```

Remarks:

- PPE sets the working catalog to the build subcatalog during the build.
- The input and output files for the build processor are specified by its parameter list. The parameter list used for the build is the global parameter list highlighted on the Parameter List Library screen.
- The default input file for for the build processor is $PROCESSOR_INPUT. PPE directs SCU to write the expanded source to the input file.
- The default output file for the build processor is $PROCESSOR_OUTPUT. PPE combines the object modules written to the output file with the object library at this level.
- The default source map file is $INPUT_SOURCE_MAP. The information that the deck expansion writes to the source map file allows for insertion of the build diagnostics into the source text at the points the errors were detected. This information is also used for displaying the source under control of the full-screen debugger.
- If the build fails due to processor errors, PPE displays the Build Errors screen.
- If the build fails due to SCU errors, PPE displays the Build Files screen. The file $EXPAND_ERRORS can be examined for the errors that caused the failure.
- If the build succeeds, PPE displays the Run Files screen; it also runs the product if you have selected the Run automatically after build option on the Tailor Options screen.

CHANGE_BUILD_PROCESSOR

Purpose: Selects the build processor used in all subsequent builds.

Available from: Build Processors screen.

Function key: ChaBP

Format:
```
CHANGE_BUILD_PROCESSOR or CHABP
    PROCESSOR=name
    STATUS=status_variable
```

Parameter:
PROCESSOR or P
Name of the selected build processor. The valid names are COBOL, CYBIL, FORTRAN, or VECTOR_FORTRAN.

If this parameter is omitted, the build processor under the cursor is selected.

Remarks:

- The currently selected build processor is highlighted on the screen.
- The selected build processor is used for all subsequent builds and so it must be an appropriate processor for the source code at the level for which PPE is executed. For example, it would not be appropriate to specify COBOL as the build processor when the source code to be compiled is FORTRAN source code.

CHANGE_CURRENT_MODIFICATION

Purpose: Selects the modification used by subsequent edit sessions.

Available from: Modification List screen.

Function key: ChaCM

Format:
```
CHANGE_CURRENT_MODIFICATION or CHACM
    MODIFICATION_NAME=name
    STATUS=status_variable
```

Parameter: MODIFICATION_NAME or MN
Name of the selected modification. The modification names are listed on the screen. If you omit this parameter, the modification under the cursor is selected.

Remarks:
- The modification must exist at the bottom level of the hierarchy. In other words, it must be a new modification or a modification that applies to a deck existing at this level. If necessary, extract a deck to which the modification applies using EXTRACT_SOURCE.

CHANGE_DISPLAY_CEILING

Purpose: Selects the highest PPE level whose content is included in displayed lists.

Available from: Environment Description screen.

Function key: ChaDC

Format:
```
CHANGE_DISPLAY_CEILING or CHADC
    LEVEL_NAME=name
    STATUS=status_variable
```

Parameter: LEVEL_NAME or LN
Name of the selected PPE level. The level names are listed on the screen. If you omit this parameter, the level under the cursor is selected. If the cursor is not positioned on a level name, PPE prompts for the level name.

Remarks:
- The display ceiling is highlighted in the list of Hierarchy Level Names.

CHANGE_GLOBAL_PARAMETER_LIST

Purpose: Selects the parameter list under the cursor as the global parameter list. The global parameter list is the parameter list used by all subsequent builds.

Available from: Parameter List Library screen.

Function key: Global

Format: CHANGE_GLOBAL_PARAMETER_LIST or CHAGPL or GLOBAL
STATUS=status_variable

Remarks:
- The currently selected parameter list is highlighted on the screen.

CHANGE_TO_DEFAULT

Purpose: Changes the parameter under the cursor to its default value.

Available from: Parameter List screen.

Function key: Reset

Format: CHANGE_TO_DEFAULT or CHATD or RESET
STATUS=status_variable

Remarks:
- For the processor's INPUT, OUTPUT, INPUT_SOURCE_MAP, and ERROR parameters the PPE default values are chosen. See the Parameter List screen for more information. For all other parameters, the processor's default values are chosen.

CLEAR_SCREEN

Purpose: Clears and replots the PPE screen.

Available from: Any screen.

Function key: Refrsh

Format: CLEAR_SCREEN or CLES or CLEAR or REFRSH
STATUS=status_variable

CREATE_DECK

Purpose: From the Deck List screen, displays the Deck Creation screen; from the Deck Creation screen, executes the deck creation operation.

Available from: Deck Creation or Deck List screen.

Function key: Create

Format:
```
CREATE_DECK or CRED or CREATE
    STATUS=status_variable
```

Remarks

- After the information for a new deck is entered on the Deck Creation screen, you use CREATE_DECK to actually create the new deck. PPE notifies you if a deck with the same name already exists in the hierarchy; if so, it does not create the deck and leaves the Deck Creation screen displayed.

  If the specified modification does not exist at any level of the PPE environment catalog hierarchy, PPE asks whether you want the modification created with SCU default values (SCU does not use the Author value from the Tailor Options screen). If you say yes to the prompt, PPE creates the modification; otherwise, PPE does not create the modification, leaves the Deck Creation screen displayed, and does not create the deck.

  If the specified modification does not exist at the lowest level of the environment catalog hierarchy, PPE issues a message and restores the Deck Creation screen without creating a new deck.

- If the deck creation succeeds, PPE executes a BACK_TO_PREVIOUS_CONTEXT command to return to the Deck List screen.

CREATE_MODIFICATION

Purpose: From the Modification List screen, displays the Modification Creation screen; from the Modification Creation screen, requests creation of the modification.

Available from: Modification Creation or Modification List screen.

Function key: Create

Format:
```
CREATE_MODIFICATION or CREM or CREATE
    STATUS=status_variable
```

Remarks:

- After the information for a new modification is entered on the Modification Creation screen, you use CREATE_MODIFICATION to actually create the new modification.
- From the Modification List screen, this functon takes you to the Modification Creation screen.

  PPE notifies you if a modification with the same name already exists in the hierarchy; if so, it does not create the modification and leaves the Modification Creation screen displayed.
- If the modification creation succeeds, PPE executes a BACK_TO_PREVIOUS_CONTEXT command to return to the Modification List screen.

CREATE_PARAMETER_LIST

Purpose: Starts a parameter list creation operation by prompting for the parameter list name and then displaying the Parameter List screen.

Available from: Parameter List Library screen.

Function key: Create

Format:
```
CREATE_PARAMETER_LIST or CREPL or CREATE
    STATUS=status_variable
```

Remarks:

- The initial values in the new parameter list are those of the global parameter list.
- The new parameter list is saved when you exit the Parameter List screen.

DELETE_BUILD_FILE

Purpose: Logically deletes the selected file from the build file list.

Available from: Build Files screen.

Function key: Delete

Format:
```
DELETE_BUILD_FILE or DELBF or DELETE
    STATUS=status_variable
```

Remarks:

- You can undo the deletion of a build file using UNDO_LAST_ DELETE. This is effective until you physically delete the file.

  A logically deleted file becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.

  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.

  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DELETE_DECK

Purpose: Logically deletes the selected decks from the deck list.

Available from: Deck List screen.

Function key: Delete

Format:
```
DELETE_DECK or DELD or DELETE
    STATUS=status_variable
```

Remarks:

- The selected decks are the decks that have been marked or, if none are marked, the deck under the cursor.
- Interlocked decks cannot be deleted. You can clear interlocks by using the CHANGE_DECK SCU subcommand on the home line. However, use caution when clearing interlocks; certify that the deck is no longer needed in the hierarchy. A deck with interlocks cleared cannot be transmitted, nor can you transmit to a deck with cleared interlocks.
- The command deletes the deck copy at the current PPE level only. If the deck exists at higher levels, the level name listed as the Nearest Residence of the deck changes.
- You can undo the deletion of a deck using UNDO_LAST_DELETE. This is effective until you physically delete the decks.

  A logically deleted deck becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DELETE_IDENTICAL_ERRORS

Purpose: Deletes all error messages identical to the highlighted error message. Locates the next error message, highlights it, and places the cursor as close as possible to the location where the error occurred.

Available from: Full Screen Editor session requested from the Build Errors screen.

Function key: Del IE

Format:
```
DELETE_IDENTICAL_ERRORS or DELIE
    STATUS=status_variable
```

Remarks:

- PPE decrements the error count for the deck by the number of messages deleted. It then finds the next error message, highlights it, and positions the cursor as near as possible to the point at which the error was detected.

DELETE_LIST_ENTRY

Purpose: Logically deletes the selected file from the library list.

Available from: Library Lists screen.

Function key: Delete

Format: DELETE_LIST_ENTRY or DELLE or DELETE
STATUS=status_variable

- You can undo the deletion of a file using UNDO_LAST_DELETE. This is effective until you physically delete the files.

  A logically deleted file becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.

  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.

  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DELETE_MODIFICATION

Purpose: Logically deletes the selected modifications from the modification list.

Available from: Modification List screen.

Function key: Delete

Format:
```
DELETE_MODIFICATION or DELM or DELETE
    STATUS=status_variable
```

Remarks:

- The selected modifications are those that are marked or, if none are marked, the modification under the cursor.
- The command deletes the modification at the current PPE level only. If the modification exists at higher levels, the level name listed as the Nearest Residence of the modification changes.
- You can undo the deletion of a modification using UNDO_LAST_DELETE. This is effective until you physically delete the modification.

  A logically deleted modification becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DELETE_PARAMETER_LIST

Purpose: Logically deletes the selected parameter lists from the parameter list library.

Available from: Parameter List Library screen.

Function key: Delete

Format:
```
DELETE_PARAMETER_LIST or DELPL or DELETE
    STATUS=status_variable
```

Remarks:

- Select a parameter list by placing the cursor on a parameter list name.
- You cannot delete the current global parameter list. However, you can change the global parameter list using CHANGE_GLOBAL_PARAMETER_LIST and then delete the former global parameter list.
- You can undo the deletion of a parameter list using UNDO_LAST_DELETE. This is effective until you physically delete the parameter list.

  A logically deleted parameter list becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DELETE_RUN_FILE

Purpose: Logically deletes the file under the cursor from the Run Files list.

Available from: Run Files screen.

Function key: Delete

Format: DELETE_RUN_FILE or DELRF or DELETE
STATUS=status_variable

Remarks:

- You can undo the deletion of a run file using UNDO_LAST_DELETE. This is effective until you physically delete the file.

  A logically deleted file becomes physically deleted if:

  - Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.

  - You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.

  - Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.

DISPLAY_BANNER

Purpose: Displays the Banner screen.

Available from: Any screen except the Banner and Screen Stack screens.

Function key: Banner

Format: DISPLAY_BANNER or DISB or BANNER
STATUS=status_variable

DISPLAY_BUILD_ERRORS

Purpose: Displays the Build Errors screen.

Available from: Deck List screen.

Function key: Errors

Format:
```
DISPLAY_BUILD_ERRORS or DISBE or ERRORS
    STATUS=status_variable
```

DISPLAY_BUILD_FILES

Purpose: Displays the Build Files screen.

Available from: Deck List screen.

Function key: BFiles

Format:
```
DISPLAY_BUILD_FILES or BFILES or DISBF
    STATUS=status_variable
```

DISPLAY_BUILD_PROCESSORS

Purpose: Displays the Build Processors screen.

Available from: The Build Errors, Build Files, Deck List, Environment Description, Library Lists, Modification List, Run Files, and Tailor Option screens.

Function key: BProcs

Format:
```
DISPLAY_BUILD_PROCESSORS or BPROCS or DISBP
    STATUS=status_variable
```

DISPLAY_LIBRARY_LISTS

Purpose: Displays the Library Lists screen.

Available from: The Build Errors, Build Files, Deck List, Environment Description, Modification List, Run Files, and Tailor Option screens.

Function key: LLists

Format:
```
DISPLAY_LIBRARY_LISTS or LLISTS or DISLL
    STATUS=status_variable
```

DISPLAY_PARAMETER_LIST_LIBRARY

Purpose: Displays the Parameter List Library screen.

Available from: The Build Errors, Build Files, Build Processors, Deck List, Environment Description, Library Lists, Modification List, Parameter List, Run Files, and Tailor Options screens.

Function key: PLists

Format:
```
DISPLAY_PARAMETER_LIST_LIBRARY or PLISTS or DISPLL
    PROCESSOR=name
    STATUS=status_variable
```

Parameter:

PROCESSOR or P
Optional name specifying the build processor whose parameter list library is to be displayed. The build processor names are COBOL, CYBIL, FORTRAN, or VECTOR_FORTRAN.

If this parameter is omitted, the currently selected build processor is used. The current build processor is shown on the Environment Description, Tailor Options, and the Build Processors screens.

On the Build Processor screen, the build processor selected is the processor under the cursor or the highlighted processor if no processor is under the cursor. For all other screens, the current build processor is selected.

DISPLAY_REMAINING_ERROR_COUNT

Purpose: Displays the number of error messages remaining in the deck being edited.

Available from: Full-Screen Editor session requested from the Build Errors screen.

Function key: DisREC

Format:
```
DISPLAY_REMAINING_ERROR_COUNT or DISREC
    STATUS=status_variable
```

DISPLAY_RUN_FILES

Purpose: Displays the Run Files screen.

Available from: Deck List or Modification List screens.

Function key: RFiles

Format:
```
DISPLAY_RUN_FILES or RFILES or DISRF
    STATUS=status_variable
```

DISPLAY_SCREEN_STACK

Purpose: Displays the Screen Stack screen.

Available from: Any screen except the Banner and Screen Stack screens.

Function key: Stack

Format:
```
DISPLAY_SCREEN_STACK or DISSS or STACK
    STATUS=status_variable
```

DISPLAY_SCU_OBJECT_LIST

Purpose: Displays either the Deck List screen or the Modification List screen.

Available from: Deck List, Environment Description, or Modification List screens.

Function key: SCUObj

Format:
```
DISPLAY_SCU_OBJECT_LIST or DISSOL or SCUOBJ
    OBJECT_KIND=keyword
    STATUS=status_variable
```

Parameter: OBJECT_KIND or OK
Keyword specifying the kind of SCU object to be displayed. Currently, the valid keywords are:

DECK or DECKS or D

MODIFICATION or MODIFICATIONS or M

If this parameter is omitted, the SCU object kind (DECKS or MODIFICATIONS) under the cursor is used. Otherwise, if the cursor is not positioned on DECKS or MODIFICATIONS, PPE prompts for the object kind.

Remarks:

- This operation can also be done by positioning the cursor on DECKS or MODIFICATIONS and pressing the carriage-return key.
- If the requested screen is already displayed, PPE redisplays the screen with updated data.
- If the requested screen is not displayed, but is already on the screen stack, PPE executes repeated BACK operations until it reaches the requested screen.

DISPLAY_TAILOR_OPTIONS

Purpose: Displays the Tailor Options screen.

Available from: The Build Errors, Build Files, Build Processors, Deck List, Environment Description, Library Lists, Modification List, and Run Files screens.

Function key: Tailor

Format: DISPLAY_TAILOR_OPTIONS or DISTO or TAILOR
STATUS=status_variable

EDIT_BUILD_FILE

Purpose: Starts a Full-Screen Editor session to display the contents of the selected build file.

Available from: Build Files screen.

Function key: Edit

Format:
```
EDIT_BUILD_FILE or EDIBF or EDIT
    STATUS=status_variable
```

Remarks:
- The file to be edited must be a listable file; a catalog or an object library cannot be edited.

EDIT_DECK

Purpose: Starts a Full-Screen Editor session to edit the selected deck. When entered from the Deck Creation screen, EDIT_DECK also creates the deck. When entered from the Build Errors or Deck List screens, it also extracts the deck if the deck does not reside at the lowest level.

Available from: Build Errors, Deck Creation, or Deck List screen.

Function key: Edit

Format:
```
EDIT_DECK or EDID or EDIT
    STATUS=status_variable
```

Remarks:
- The changes made during the editing session are assigned to the current modification.
- When this command is entered from the Deck Creation or Deck List screen, PPE displays only the deck source text for editing. However, when this command is entered from the Build Errors screen, the error diagnostics from the last build are inserted in the text where they occurred.

EDIT_DECK_TEMPLATE

Purpose: Starts a Full-Screen Editor session to edit the selected deck template.

Available from: Build Processors screen.

Function key: EdiDT

Format:
```
EDIT_DECK_TEMPLATE or EDIDT
    PROCESSOR=name
    STATUS=status_variable
```

Parameter: PROCESSOR or P

Optional name of the build processor whose deck template is to be edited. The build processor names are COBOL, CYBIL, FORTRAN, or VECTOR_FORTRAN.

If this parameter is omitted, the processor under the cursor is used. If the cursor is not positioned on the name of a build processor, the currently selected build processor is used. (This selected build processor is highlighted on the screen.)

EDIT_PARAMETER_LIST

Purpose: Enables editing of the selected parameter list.

Available from: Any screen except the Banner, Deck Creation, Modification Creation, and Screen Stack screens.

Function key: EdiPL

Format:
```
EDIT_PARAMETER_LIST or EDIPL
    PARAMETER_LIST=name
    PROCESSOR=name
    STATUS=status_variable
```

Parameter: PARAMETER_LIST or PL

Optional name of the parameter list to be edited. To see the names of the existing parameter lists, display the Parameter List Library screen for the selected build processor using DISPLAY_PARAMETER_LIST_LIBRARY.

If the specified parameter list does not exist, PPE displays a message which you must acknowledge to continue the PPE session.

If this parameter is omitted, the parameter list displayed depends upon the screen currently displayed, as follows:

Parameter List screen	- PPE prompts you for a parameter list name.
Parameter List Library screen	- PPE displays the parameter list under the cursor. If no parameter list is under the cursor, PPE displays the global parameter list of the current build processor.
All other screens	- PPE displays the global parameter list of the current build processor.

PROCESSOR or P
Optional name of the build processor for which the parameter list is specified. To see the names of the build processors available, display the Build Processors screen using DISPLAY_BUILD_PROCESSORS.

If this parameter is omitted, the build processor selected depends upon the screen currently displayed, as follows:

Build Processors screen	- PPE selects the processor under the cursor. If the cursor in not on a processor, the highlighted processor is chosen.
Parameter List Library screen	- PPE selects the processor named at the top of the screen.
Parameter List screen	- PPE selects the processor named at the top of the screen.
All other screens	- PPE selects the current build processor.

Remarks:
- When EDIT_PARAMETER_LIST is used on the Parameter List screen and specifies another parameter list, the currently displayed parameter list is saved before the next list is displayed.

EDIT_RUN_FILE

Purpose: Starts a Full-Screen Editor session to display the contents of the selected run file.

Available from: Run Files screen.

Function key: Edit

Format: EDIT_RUN_FILE or EDIRF or EDIT
STATUS=status_variable

Remarks:
- The file to be edited must be listable; a catalog or an object library cannot be edited.

END_MARK

Purpose: Marks the last object in a range of selected objects.

Available from: The Build Errors, Deck List, and Modification List screens.

Function key: EndMrk

Format:
```
END_MARK or ENDMRK or ENDM
    STATUS=status_variable
```

Remarks:
- A corresponding BEGIN_MARK command must precede each END_MARK command.

EXPLAIN_ERROR_MESSAGE

Purpose: Displays the online manual explanation of an error message. The error message is either the one under the cursor or, if the cursor is not positioned on a message, the highlighted message.

Available from: Full-Screen Editor session requested from the Build Errors screen.

Function key: Assist

Format:
```
EXPLAIN_ERROR_MESSAGE or EXPEM or ASSIST
    STATUS=status_variable
```

EXPORT_BUILD_FILE

Purpose: Copies the selected file from the Build File list to another file. PPE prompts you for the file to which it copies the build file.

Available from: Build Files screen.

Function key: Export

Format:
```
EXPORT_BUILD_FILE or EXPBF or EXPORT
    STATUS=status_variable
```

EXPORT_RUN_FILE

Purpose: Copies the selected Run File to another file. PPE prompts you for the file to which it copies the run file.

Available from: Run Files screen.

Function key: Export

Format:
```
EXPORT_RUN_FILE or EXPRF or EXPORT
    STATUS=status_variable
```

EXTRACT_SOURCE

Purpose: Copies the selected decks from their nearest residence in the higher levels of the PPE hierarchy to your PPE level and sets an interlock on each deck.

Available from: Build Errors or Deck List screen.

Function key: ExtS

Format:
```
EXTRACT_SOURCE or EXTS
    STATUS=status_variable
```

Remarks:

- A deck cannot be extracted if it is already interlocked or if it already exists at the lowest level of the environment catalog hierarchy. A deck is already extracted if its Nearest Residence is the current PPE level.

FORMAT_SOURCE_TEXT

Purpose: Executes the source text formatter for the build processor to format the text in the deck being edited.

Available from: Full-Screen Editor session.

Function key: ForST

Format:
```
FORMAT_SOURCE_TEXT or FORST
    STATUS=status_variable
```

Remarks:

- FORMAT_SOURCE_TEXT looks for a command with a name of the form FORMAT_processor_name_SOURCE where processor_name is the name of the processor for the source file being edited (for example, FORMAT_COBOL_SOURCE). If FORMAT_SOURCE_TEXT finds a command with a name of this form, it executes the command. Two parameters are passed to the command: the name of the source file to be formatted, and the name of the file to contain the formatted source.

  If a command of the form FORMAT_processor_name_SOURCE is not found and PPE provides a source code formatter for the processor, then the PPE provided formatter is used to format the source code. Currently, PPE only provides a source code formatter for FORTRAN Version 1.

  If a command of the form FORMAT_processor_name_SOURCE is not found and PPE does not provide a source code formatter for the processor, PPE issues a message informing you that a formatter does not exist for the processor.

- Currently, the only source text formatter supplied by PPE is for FORTRAN Version 1.

  The text to be formatted must be in the correct zones within each line (labels must be in columns 1 through 5, statements at column 7 and on). The formatter standardizes the format, indenting block structure and standardizes labels.

HOME_CURSOR

Purpose: Positions the cursor at the beginning of the home line.

Available from: Any screen except the Banner screen.

Function key: Home

Format:
```
HOME_CURSOR or HOMC or HOME
    STATUS=status_variable
```

IMPORT_BUILD_FILE

Purpose: Copies a file to a file in the PPE Build File subcatalog. You specify the file to be copied in response to a prompt. PPE also prompts you for the name to be given the new PPE file.

Available from: Build Files screen.

Function key: Import

Format:
```
IMPORT_BUILD_FILE or IMPBF or IMPORT
    STATUS=status_variable
```

IMPORT_RUN_FILE

Purpose: Copies a file to a file in the PPE Run File subcatalog. You specify the file to be copied in response to a prompt. PPE also promptes you for the name to be given the new PPE file.

Available from: Run Files screen.

Function key: Import

Format:
```
IMPORT_RUN_FILE or IMPRF or IMPORT
    STATUS=status_variable
```

IMPORT_SOURCE_LIBRARY

Purpose: Combines the specified SCU source library outside of PPE with the PPE source library at this level. Interlocks are not checked.

Available from: Environment Description screen.

Function key: Import

Format:
```
IMPORT_SOURCE_LIBRARY or IMPORT or IMPSL
    SOURCE_LIBRARY=file
    STATUS=status_variable
```

Parameter:

SOURCE_LIBRARY or SL
File path for the source library file to be imported. If you omit this parameter, PPE prompts you for the file path.

Remarks:

- Before you import a library, you should set the processor field in the header of each deck in the library to the name of the build processor for that deck. Use the SCU CHANGE_DECK command to change the deck headers.

INSERT_LIST_ENTRY

Purpose: Inserts a list entry before the list entry under the cursor. The subsequent list entries are moved down one entry. You can add a new name to the list by typing it at the prompt.

Available from: Library Lists screen.

Function key: Insert

Format:
```
INSERT_LIST_ENTRY or INSLE or INSERT
    STATUS=status_variable
```

Remarks:

- To insert an entry at the end of the list, place the cursor on the line following the last entry in the list prior to executing the command. Use PAGE_FORWARD (FWD) to reach the end of the list.

LOCATE_DECK

Purpose: Searches the deck list for the first name containing the specified string of characters and positions the cursor on that name.

Available from: Deck List screen.

Function key: Locate

Format:
```
LOCATE_DECK or LOCD or LOCATE
    MATCH_STRING=string
    STATUS=status_variable
```

Parameter:

MATCH_STRING or MS
Optional string of characters for which PPE is to search. The string must be enclosed in apostrophe (') characters. If the parameter is omitted, PPE prompts for the string. When you enter the string, do not enclose it in apostrophes. If you supply no string in response to the prompt, PPE uses the string from the last LOCATE_DECK command.

Remarks:

- LOCATE_DECK positions the cursor on the deck it finds which contains characters matching the string. It does not move the cursor if it finds no deck name containing the MATCH_STRING value.
- The search starts from the current cursor position in the deck list.

LOCATE_MODIFICATION

Purpose: Searches the modification list for the first name containing the specified string of characters and positions the cursor on the name.

Available from: Modification List screen.

Function key: Locate

Format:
```
LOCATE_MODIFICATION or LOCM or LOCATE
    MATCH_STRING=string
    STATUS=status_variable
```

Parameter:

MATCH_STRING or MS
Optional string of characters for which PPE is to search. The string must be enclosed in apostrophe (') characters.

If the parameter is omitted, PPE prompts for the string. When you enter the string, do not enclose it in apostrophes. If you supply no string in response to the prompt, PPE uses the string from the last LOCATE_MODIFICATION command.

Remarks:

- LOCATE_MODIFICATION positions the cursor on the modification it finds which contains characters matching the MATCH_STRING value.

  LOCATE_MODIFICATION does not move the cursor if it finds no modification name containing the MATCH_STRING value.

- The search starts from the current cursor position in the deck list.

LOCATE_NEXT_ERROR

Purpose: Moves the cursor to the next error, highlighting the corresponding error message. Deletes the currently highlighted error message.

Available from: Full-Screen Editor session requested from the Build Errors screen.

Function key: NxtErr

Format:
```
LOCATE_NEXT_ERROR or LOCNE or NXTERR
    STATUS=status_variable
```

Remarks:

- PPE decrements the error count for the deck. It then finds the next error message, highlights it, and positions the cursor as near as possible to the point at which the error was detected.

LOOKUP_KEYWORD

Purpose: Searches for the word under the cursor in the index of the online manual for the language and, if found, displays the associated online manual screen.

Available from: Full-Screen Editor session.

Function key: LookUp
key:

Format:
```
LOOKUP_KEYWORD or LOOK or LOOKUP
    STATUS=status_variable
```

Remarks:

- To return to PPE from the online manual, use the Quit function key or type QUIT and press the carriage-return key.
- The online manuals used for the build processors COBOL, CYBIL, FORTRAN, and VECTOR_FORTRAN are COBOL, CYBIL, FORTRAN, and VFORTRAN respectively.

LOWER_DISPLAY_CEILING

Purpose: Lowers the display ceiling the specified number of levels.

Available from: Deck List or Modification List screens.

Function key: Lower

Format:
```
LOWER_DISPLAY_CEILING or LOWDC or LOWER
    NUMBER_OF_LEVELS=integer or key
    STATUS=status_variable
```

Parameter: NUMBER_OF_LEVELS or NL

Number of levels the display ceiling is to be lowered. The value must be an integer greater than 0 or the keyword BOTTOM, which lowers the display ceiling to the lowest level in the hierarchy.

If the number specified is greater than the number of levels that the ceiling can be lowered, BOTTOM is used instead.

The default value of this parameter is 1.

MOVE_TO_BOTTOM

Purpose: Repositions the list so that the object under the cursor is the last object displayed on the page.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: Down

Format:
```
MOVE_TO_BOTTOM or MOVTB or DOWN
    STATUS=status_variable
```

MOVE_TO_FIRST

Purpose: Repositions the list so the beginning of the list is displayed and the cursor is positioned on the first object in the list.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: First

Format:
```
MOVE_TO_FIRST or MOVTF or FIRST
    STATUS=status_variable
```

MOVE_TO_LAST

Purpose: Repositions the list so the end of the list is displayed and the cursor is positioned on the last object in the list.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: Last

Format:
```
MOVE_TO_LAST or MOVTL or LAST
    STATUS=status_variable
```

MOVE_TO_TOP

Purpose: Repositions the list so that the object under the cursor is the first object displayed.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: Up

Format:
```
MOVE_TO_TOP or MOVTT or UP
    STATUS=status_variable
```

PAGE_BACKWARD

Purpose: Repositions the list so that the first object displayed becomes the last object displayed.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: Bkw

Format:
```
PAGE_BACKWARD or PAGB or BKW
    STATUS=status_variable
```

PAGE_FORWARD

Purpose: Repositions the list so that the last object displayed becomes the first object displayed.

Available from: Build Errors, Build Files, Deck List, Environment Description, Library List, Modification List, Parameter List, Parameter List Library, Run Files, and Screen Stack screens.

Function key: Fwd

Format:
```
PAGE_FORWARD or PAGF or FWD
    STATUS=status_variable
```

PRINT_BUILD_FILE

Purpose: Prints the selected build file using the site-provided PRINT command if available; otherwise, it uses the standard NOS/VE PRINT_FILE command.

Available from: Build Files screen.

Function key: Print

Format:
```
PRINT_BUILD_FILE or PRIBF or PRINT
    STATUS=status_variable
```

Remarks:

- The file to be printed must be a list file; it cannot be the object library.
- PPE passes only one parameter to the print command: the file path.
- When PPE executes the PRINT_BUILD_FILE command, it first executes an SCL command with the following form:

  ```
  PRINT lfn
  ```

  where lfn is the file path of the file to be printed. If no PRINT command exits, PPE executes the standard NOS/VE PRINT_FILE command with default settings.

PRINT_RUN_FILE

Purpose: Prints the selected run file using the site-provided PRINT command if available; otherwise, it uses the standard PRINT_FILE command.

Available from: Run Files screen.

Function key: Print

Format:
```
PRINT_RUN_FILE or PRIRF or PRINT
    STATUS=status_variable
```

Remarks:

- The file to be printed must be a list file; it cannot be the object library.
- PPE passes only one parameter to the print command: the file path.
- When PPE executes the PRINT_RUN_FILE command, it first executes an SCL command with the following form:

  ```
  PRINT lfn
  ```

  where lfn is the file path of the file to be printed. If no PRINT command exits, PPE executes the standard NOS/VE PRINT_FILE command with default settings.

QUIT_SAVE

Purpose: Ends the PPE session, saving all information required for your next session to begin where this one ended.

Available from: Any screen.

Function key: Quit

Format:
```
QUIT_SAVE or QUIS or QUIT or QUI
    STATUS=status_variable
```

Remarks: When you exit PPE, a new high cycle of the source library within your environment catalog is created. The low cycles allow you to recover freom catastrophic events. However, you can easily use up a large amount of file space. To avoid using excess file space, you should frequently delete the low cycles of your source library. You can use EDIT_CATALOG from the home line of your environment catalog to do this.

RAISE_DISPLAY_CEILING

Purpose: Raises the display ceiling the specified number of levels.

Available from: Deck List or Modification List screen.

Function key: Raise

Format:
```
RAISE_DISPLAY_CEILING or RAIDC or RAISE
    NUMBER_OF_LEVELS=integer or key
    STATUS=status_variable
```

Parameter: NUMBER_OF_LEVELS or NL

Number of levels the display ceiling is to be raised. The value can be an integer greater than 0 or the keyword TOP, which raises the display ceiling to the highest level in the hierarchy.

If the number specified is greater than the number of levels that the ceiling can be raised, TOP is used instead.

The default value of this parameter is 1.

REMOVE_MARK

Purpose: Removes the selected marks from the list.

Available from: Build Errors, Deck List, and Modification List screens.

Function key: UnMark

Format:
```
REMOVE_MARK or REMM or UNMARK
    MARK_SCOPE=key
    STATUS=status_variable
```

Parameter: MARK_SCOPE or MS

Keyword specifying the marks that are removed.

ALL or A	All marks in the list.
INDIVIDUAL or I	Individual mark under the cursor (default).
RANGE or R	Marked range under the cursor.

If you omit the MARK_SCOPE parameter, the default value is INDIVIDUAL.

REQUEST_HELP

Purpose: Displays help information. When help is requested on the Banner Screen, PPE displays the PPE online manual. Otherwise, this command displays help information about the object or keyword under the cursor or the current message on the message line.

Available from: Any screen.

Function key: Help

Format: REQUEST_HELP or REQH or HELP
STATUS=status_variable

Remarks:

- If a help message is available for the object, REQUEST_HELP displays a window containing brief help information. The window also tells you how to display further information in the PPE online manual and how to return to PPE.
- If no window help is available for the object, the PPE online manual index is searched for the object. If the object is found in the index, the corresponding online manual screen is displayed.
- To leave the online manual, use the Quit function key or type QUIT and press the carriage-return key. You are returned to PPE.
- After receiving help, PPE always returns you to the screen where you requested help.

RUN

Purpose: Loads and executes the product.

Available from: Run Files screen.

Function key: Run

Format:

```
RUN
    RUN_PURPOSE=keyword
    STATUS=status_variable
```

Parameter:

RUN_PURPOSE or RP
Keyword indicating how the product should be executed:

NORMAL or N	Execute without control by the Full-Screen Debug utility.
INTERACTIVE_ DEBUG or ID	Execute under control of the Full-Screen Debug utility. The product must have been built using the debug options in its parameter list. The debug options are:

```
OPTIMIZATION_LEVEL=DEBUG
DEBUG_AIDS=DT
INPUT_SOURCE_MAP=$INPUT_SOUR
CE_MAP
```

If you omit the RUN_PURPOSE parameter, the Run Purpose selected on the Run Files screen is used.

Remarks:

- PPE loads the object library existing at this level. It then satisfies the external references in the library by loading modules from other object libraries. When searching for the entry point to satisfy an external reference or starting procedure, it searches in the following order:

  1. In the PPE object library at the current level.
  2. In the PPE object library at each higher level upwards in the hierarchy.
  3. In the object libraries specified in the program library list on the Library lists screen in the order listed.
  4. In the object libraries listed in the job library list in the order the library files are listed.

Program execution begins with the starting procedure as specified on the Tailor Options screen. (The default starting procedure is the last transfer symbol loaded, the same default in effect outside of PPE.)

The program reads its input files and writes its output files. It may also write the following run files:

$LOADMAP
: A load map is written to this run file if the LOAD_MAP_OPTIONS program attribute has been changed from its default value, NONE. To change the program attribute, you must enter the SCL command SET_PROGRAM_ATTRIBUTES.

$TERMINAL_OUTPUT
: All output data sent to $OUTPUT is written to this run file.

- When you execute the Run function, PPE searches every object library in your hierarchy. The loader produces a warning level error message if it encounters an empty object library. Since PPE sets the initial termination error level for a job to WARNING, an empty library might cause your run to terminate prematurely.

  You can change the termination error level with the SCL SET_PROGRAM_ATTRIBUTES command. The recommended settings are WARNING and ERROR. The following lists the merits of each choice:

  ERROR
  : An empty object library does not terminate a run. However, PPE does not inform you when any warning level error messages are produced, and the run might terminate unexpectedly. Also, you are not informed if an empty object library is encountered, even if none should be empty.

  WARNING
  : Any WARNING level error message terminates a run. Therefore, an empty object library causes a run to terminate.

  Regardless of the termination error level setting for your job, all errors are written to the $LOADMAP file.

SKIP_LINE_ERRORS

Purpose: Deletes all error messages associated with the same source line as the highlighted message. Highlights the first remaining error message, moving the cursor as close as possible to the location where it occurred.

Available From: Full-Screen Editor session requested from the Build Errors screen.

Function key: Skip

Format:
```
SKIP_LINE_ERRORS or SKILE or SKIP
    STATUS=status_variable
```

TRANSMIT_SOURCE

Purpose: Copies the selected decks to the source library at the next higher PPE level, clears the deck interlocks, and deletes the decks from the lower PPE level.

Available from: Deck List screen.

Function key: TraS

Format:
```
TRANSMIT_SOURCE or TRAS
    STATUS=status_variable
```

Remarks:

- PPE attempts to transmit all decks marked or, if none are marked, it transmits the deck under the cursor.
- TRANSMIT_SOURCE always transmits decks from the lowest level in the environment catalog hierarchy to the source library at the next higher level.
- Only decks residing in the lowest level of the environment catalog hierarchy can be transmitted.

- When you transmit source, a new high cycle of the source library within your environment catalog is created. The low cycles allow you to recover from catastrophic events. However, you can easily use up a large amount of file space. To avoid using excess file space, you should frequently delete the low cycles of you source library. You can use EDIT_CATALOG from the home line of your environment catalog to do this.

  When you transmit source, the object modules in the lowest level of your environment catalog hierarchy that are built by that source should be deleted from the highest cycle of $OBJECT_LIBRARY on the Build Files screen. The CREATE_OBJECT_LIBRARY utility can be used from the home line of the Build Files screen to do this.

  Also, after you transmit, notify the owner of the next higher level in your hierarchy to do a build using the transmitted decks. This keeps the $OBJECT_LIBRARY at the higher level of the hierarchy up to date, incorporating your transmitted source code into the object code at that level.

  Changes made to the source library are not available to other tasks or PPE sessions at lower levels in the hierarchy until you exit the PPE session in which the changes occurred, or until the changed decks are transmitted to the next higher level in the hierarchy.

UNDO_LAST_DELETE

Purpose: Removes the effect of the last logical deletion operation, restoring the deleted objects. However, physically deleted objects cannot be restored.

Available from: Build Files, Deck List, Library Lists, Modification List, Parameter List Library, or Run Files screen.

Function key: Undo

Format: UNDO_LAST_DELETE or UNDLD or UNDO
STATUS=status_variable

Remarks: A logically deleted object becomes physically deleted if:

- Delete protection is ON and you confirm the deletion. The delete protection warning is displayed when you attempt to leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object.
- You exit PPE. When you exit PPE, all logically deleted objects become physically deleted. The delete protection warning is not displayed even if you selected Delete protection from the Tailor Options screen.
- Delete protection is OFF and you leave the current screen, execute a command on the home line, or execute a command that might reference a logically deleted object. No deletion warning is displayed and you cannot veto the deletion.
- A physically deleted object cannot be recovered by using UNDO_LAST_DELETE.

Appendixes

Glossary A

This appendix lists terms with their definitions as used in this manual.

A

Author

Identification of the original creator of a deck or modification as stored in its header.

B

Build

Process by which PPE constructs the executable object library from decks on a source library. To do so, it calls SCU to expand the decks, then calls the build processor to compile the expanded text, and finally calls the object library generator to combine the object modules with the existing object library.

Build Processor

The software that PPE calls to transform the expanded deck text into object modules. The supported build processors are the NOS/VE COBOL and FORTRAN Version 1 compilers.

C

Catalog

A system-maintained directory whose entries describe NOS/VE files and subcatalogs. The NOS/VE file system is organized as a hierarchy of catalogs. Each user has a master catalog, which is the top of the user's catalog hierarchy.

Compilation

The process of transforming source text into object code.

Context

Within PPE, the screen currently displayed, including the cursor position and any currently displayed messages. It determines the commands currently available and the context in which they are executed.

Creation Modification

The modification specified when a deck is created. The initial source text in the deck belongs to the creation modification. A creation modification cannot be deleted until all decks for which the modification is a creation modification are deleted.

Cursor

The pointer displayed by your terminal that you can position to specify a screen location.

D

Deck

An SCU object consisting of a sequence of lines with a descriptive header. Under PPE, the lines in a deck are a unit of source text that can be expanded by SCU and then compiled by the build processor.

Default

The assumed value for a parameter when the parameter is not specified by the user.

Diagnostic Message

Text issued to describe an error that has been detected. PPE weaves the diagnostic messages issued for a build by the build processor into the source text so that the message is as near as possible to the point at which the error was detected.

Display Ceiling

Highest level in the PPE hierarchy whose contents are included in displayed lists.

E

Entry Point

A location in a module at which execution of the module can begin. During the loading process, an entry point is found to match each external reference in the product.

Environment

The data and operations available to the programmer.

Execution

The carrying out of a program's instructions by the computer.

Expand Attribute

A deck header field that determines whether the deck can be expanded directly or whether it can be expanded only when it is copied by another deck.

Expansion

The process by which SCU reformats the compressed text stored in a deck so that it can be used as input text to a build processor. As it expands the deck, it processes all SCU directives embedded in the deck text.

External Reference

A call to an entry point in another module. The loading process satisfies external references by matching each external reference to its entry point.

Extraction

The process by which an existing deck at a higher level is made available for editing under PPE. PPE copies the deck from the nearest higher level of the PPE hierarchy, sets an interlock on the original deck and its copy, and combines the deck copy with the source library at the bottom of the hierarchy.

F

File Path

The means of identifying a NOS/VE file. It specifies the location of the file entry by the sequence of catalog names that lead to the file entry in the hierarchy of catalogs. The last name in the file path is the name of the file itself. For more information, see the SCL Language Definition manual.

FSE

Full-Screen Editor; the NOS/VE editor that allows you to edit text either in line mode or screen mode.

Full-Screen Application

A program whose user interface sends and receives screens of information, not just individual lines. Such a program can be used only from a terminal having both the minimal capabilities required by the program and a terminal definition defining those capabilities.

Function Key

A key on the terminal keyboard that, when pressed, transmits a unique, identifying character sequence. The terminal definition for a full-screen terminal defines the function keys available. As a full-screen application, PPE uses the terminal definition so that when you press a function key PPE executes the command associated with that key.

G

Global Parameter List

The currently selected parameter list for the build processor.

H

Hierarchy

A specified rank or order of items. PPE supports a hierarchy of levels; a user can change data only at the lowest level in the hierarchy, but has access to data at higher levels.

Home Line

The line on the terminal screen to which the cursor returns when the HOME_CURSOR operation is performed or when the terminal's hardwired HOME key is pressed. The user can enter commands only on the home line.

I

Interlock

A field set in a deck header indicating that the deck has been extracted. The original interlock field is set on header of the the original deck in the source library from which the deck was extracted; the subinterlock field is set in the header of the deck copy.

L

Level

Rank within a hierarchy. A PPE level is a catalog containing required PPE files and catalogs, including a source library, an object library, and other data files.

M

Modification

An SCU object consisting of a set of inserted, replaced, and deleted lines with a descriptive header.

N

Name

SCL identifier consisting of a sequence of 1 through 31 characters. The characters can be letters, digits, and the following special characters:

```
# @ $ _ [ ]   ^ ` { } | ~
```

The first character cannot be a digit.

O

Object Library

A file containing one or more executable modules and a directory to each module and its entry points. PPE maintains an object library at each level of the hierarchy.

Object Module

A compiler-generated unit containing object code and instructions for loading the object code. Under PPE, all object modules are stored in object libraries.

P

Parameter List

A string of specifications passed to a program when its execution is begun. PPE passes a parameter list to the build processor for each build. It also passes a parameter list to the product for each run.

Parameter List Library

A collection of parameter lists stored for a build processor. The PPE user can create, edit, and select parameter lists in the library.

Program Library List

A list of object libraries searched by the loader for entry points when the entry points do not exist in the PPE object libraries. After searching the program library list provided by PPE, the loader then searches the job library list, the job Debug library list (if the run purpose is INTERACTIVE_DEBUG), and, finally, the NOS/VE task services library.

R

Run

An instance of execution of the software product being developed using PPE.

S

Screen

A formatted collection of information that can be displayed at one time on a terminal screen. The currently displayed PPE screen determines the context in which PPE operates and the commands available to the user.

SCU

Source Code Utility; the NOS/VE command utility that PPE uses to manage decks and modifications on source libraries.

SCU Object

Entity generated and maintained by SCU. Currently, the only SCU objects that PPE supports are decks and modifications.

Source Library

A collection of decks on a file with a descriptive header, generated and maintained by SCU. PPE maintains a source library at each level of a PPE hierarchy.

Source Text

The sequence of lines provided as input to a compiler or other processor.

Stack

Data structure in which data items are retrieved in reverse order from that in which the items were stored. An item stored in the stack is said to be pushed on the stack; an item retrieved from the stack is said to be popped off the stack. PPE maintains a stack showing the path the user has taken down the PPE screen hierarchy; each BACK command pops a screen from the stack.

T

Terminal Definition

A module that defines the capabilities of an interactive terminal. An interactive session must define the terminal it is using before executing any full-screen application.

Text-Embedded Directive

An SCU directive inserted at the point in the deck text at which the directive is to be processed when the deck is expanded. The most commonly used directives are the copy directives that expand the text of another deck and insert it into the deck containing the copy directive.

Transfer symbol

An entry point within a module at which program execution can begin.

Transmittal

The process by which decks in the lowest level of the environment catalog hierarchy are moved to the source library at the next higher level of the PPE hierarchy. Deck interlocks are enforced and the decks are deleted from the lower level of the hierarchy.

Related Manuals B

This appendix lists the manuals related to this manual and describes how you can acquire them.

A complete list of NOS/VE manuals is provided as appendix B in the SCL Language Definition manual.

The following list includes all manuals that are either referenced in this manual or contain additional information that may be of use to the reader of this manual. It lists the publication number of printed manuals and the manual name for online manuals. (The manual name is the name specified on the MANUAL parameter of the EXPLAIN command; for example, EXPLAIN,M=DEBUG.)

Manual Title	Publication Number	Online Manual Name
SCL Manuals:		
Debug for NOS/VE Usage	60488213	
Debug for NOS/VE Quick Reference		DEBUG
Diagnostic Messages for NOS/VE Quick Reference	60464613	MESSAGES
Full Screen Editor for NOS/VE Tutorial/Usage	60464015	
NOS/VE System Usage	60464014	
NOS/VE Object Code Management Usage	60464413	
NOS/VE Source Code Management Usage	60464313	
Terminal Definition Usage	60464016	

(Continued)

Related Manuals

(Continued)

Manual Title	Publication Number	Online Manual Name
Language Manuals:		
COBOL for NOS/VE Usage	60486013	COBOL
FORTRAN for NOS/VE Language Definition Usage	60485913	
FORTRAN for NOS/VE Topics for FORTRAN Programmers Usage	60485916	
FORTRAN for NOS/VE Quick Reference		FORTRAN
Math Library Usage	60486513	

Accessing Online Manuals

If your site has installed the NOS/VE online manual set, you can find an abstract for each NOS/VE manual in the NOS_VE online manual. To access the NOS_VE online manual, enter the SCL command:

```
explain, manual=nos_ve
```

Ordering Printed Manuals

You can order printed Control Data manuals from your local Control Data sales office. Sites in the U.S. can also order manuals directly from the following address:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

When ordering manuals, please indicate whether you require the entire manual or just the latest revision packet.

ASCII Character Set C

This appendix lists the ASCII character set (table C-1).

NOS/VE supports the American National Standards Institute (ANSI) standard ASCII character set (ANSI X3.4-1977). NOS/VE represents each 7-bit ASCII code in an 8-bit byte. The 7 bits are right-justified in each byte. For ASCII characters, the leftmost bit is always zero.

In addition to the 128 ASCII characters, NOS/VE allows use of the leftmost bit in an 8-bit byte for 256 characters. The use and interpretaion of the additional 128 characters is user-defined.

Table C-1. ASCII Character Set (Continued)

Decimal	ASCII Code (Hexadecimal)	Octal	Graphic or Mnemonic	Name or Meaning
000	00	000	NULL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal	ASCII Code (Hexadecimal)	Octal	Graphic or Mnemonic	Name or Meaning
036	24	044	$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe
040	28	050	(	Opening parenthesis
041	29	051	)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus
044	2C	054	,	Comma
045	2D	055	-	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equal to
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal	ASCII Code (Hexadecimal)	Octal	Graphic or Mnemonic	Name or Meaning
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	[	Opening bracket
092	5C	134	\	Reverse slant
093	5D	135	]	Closing bracket
094	5E	136	^	Circumflex
095	5F	137	_	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g

(Continued)

Table C-1. ASCII Character Set (Continued)

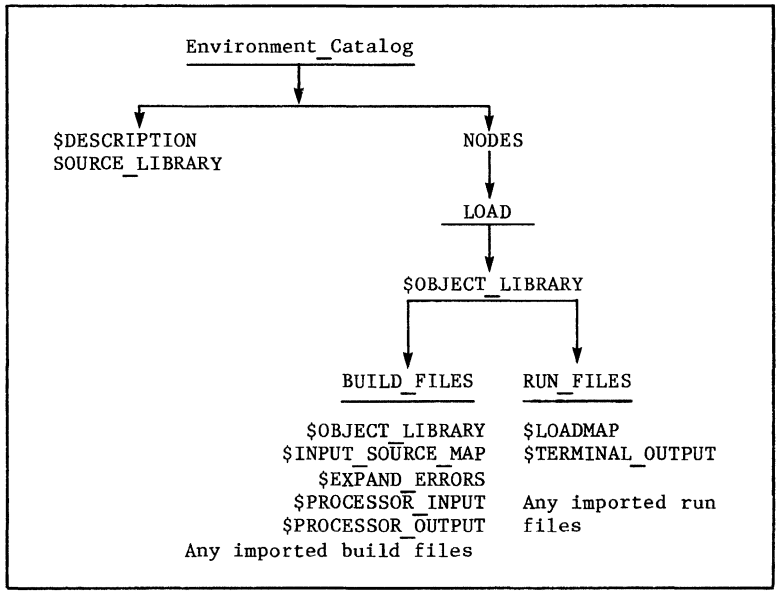
Decimal	ASCII Code (Hexadecimal)	Octal	Graphic or Mnemonic	Name or Meaning
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174	|	Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

PPE Catalog Structure D

Each level in a PPE hierarchy is a structure consisting of a hierarchy of subcatalogs and files. PPE creates the subcatalog structure within an environment catalog when it creates the environment catalog. PPE creates files in the catalog only as they are needed.

The PPE catalog structure is a hierarchy of catalogs with the catalog specified on the ENTER_PPE command at the top of the structure. The default PPE catalog is $USER.PROFESSIONAL_ENVIRONMENT.

Diagrammed, the PPE catalog structure appears as follows:



PPE Catalog Structure

PPE creates entries for two files and two subcatalogs in the PPE catalog. The files are:

$DESCRIPTION	File in which PPE maintains the information that describes the state of the PPE session when a QUIT_SAVE command is entered. PPE uses the information when it is restarted.
SOURCE_LIBRARY	SCU source library file for this PPE level.

The subcatalogs in the PPE catalog are:

```
.NODES.LOAD.$OBJECT_LIBRARY.BUILD_FILES
.NODES.LOAD.$OBJECT_LIBRARY.RUN_FILES
```

The .NODES.LOAD.$OBJECT_LIBRARY.BUILD_FILES subcatalog contains any imported build files and the following PPE-created files:

$INPUT_SOURCE_MAP	The deck expansion writes the source map to this file; it must be specified as the INPUT_SOURCE_MAP parameter value in the build processor parameter list. The source map is the information required for the mapping of build diagnostics into the deck text where the errors were detected and for the full-screen debugger to find the source it displays during debugging.
$EXPAND_ERRORS	The deck expansion process writes any errors found to this file.
$PROCESSOR_INPUT	The deck expansion writes the expanded source text to this file; it must be specified as the INPUT parameter value in the build processor parameter list.
$PROCESSOR_OUTPUT	The build processor writes the object modules it generates to this file and the object library generator uses it as input when combining the modules with the object library.

The .NODES.LOAD.$OBJECT_LIBRARY.RUN_FILES subcatalog contains any imported run files and the following PPE-created files:

$LOADMAP	Default load map file. If no other file is specified by the LOAD_MAP program attribute, the load map generated by the run (if any) is written to this file. (A load map is generated only if the appropriate options are specified by the LOAD_MAP_OPTIONS program attribute.)
$TERMINAL_OUTPUT	File on which output written to $OUTPUT is captured. The file can be viewed after the run using EDIT_RUN_FILE.

To allow another PPE level to link to your PPE catalog, you must permit the following file access to the owner of the other level.

Access	File
Read	$DESCRIPTION
Read, modify, and cycle (plus APPLICATION_INFORMATION="I1")	SOURCE_LIBRARY
Read and execute	.LOAD.$OBJECT_LIBRARY.BUILD_FILES. $OBJECT_LIBRARY

Control permission is required if the user wants PPE to delete low cycles of source_libraries during transmits.

Assume that your environment catalog is the default catalog, $USER.PROFESSIONAL_ENVIRONMENT, and that the user name to which you want to give access is FRED. You can create various file and catalog permits that allow FRED to link to your level. Two examples follow:

1. You can give FRED access to your level with the catalog permit:

```
create_catalog_permit, ..
    catalog=$user.professional_environment, group=user, ..
    user=fred, ..
    access_modes=(read, modify, cycle, control, execute), ..
    share_mode=none
```

   With this permission, FRED can access all files at your level. You can restrict access to files at your level by setting more restrictive file permits on these files. For example, if you want to allow only read access to the file $DESCRIPTION create the following file permit:

```
create_file_permit, ..
    file=$user.professional_environment.$description, ..
    group=user, user=fred, access_modes=read, share_mode=none
```

2. You can give FRED access to individual files at your level with the following SCL commands:

```
set_working_catalog, $user.professional_environment

create_file_permit, file=$description, group=user, ..
    user=fred, access_modes=read, share_mode=none


create_file_permit, file=source_library, group=-user, ..
    user=fred, ..
    access_modes=(read, modify, cycle, control), ..
    share_mode=none, application_information="I1"

create_file_permit, ..
f=nodes.load.$object_library.build_files.$object_library, ..
    group=user, user=fred, access_modes=(read, execute) ..
    share_mode=none
```

Note that the last file permit can be made only after a build has occurred at your level. Otherwise, only the file permits for the files $DESCRIPTION and SOURCE_LIBRARY can be created, and user name FRED can access only these files when linked to your level.

PPE Function Key Summary E

The following is an alphabetical listing of the function key labels and the operation performed by the corresponding function key.

Function Key Label	Screen on Which the Label Appears	PPE Operation
Assist	FSE session requested from the Build Errors	Displays the online manual explanation of an error message.
Back	All screens except Environment Description	Returns to the preceding screen on the screen stack.
BacTC	Screen Stack	Pops the screen stack to display the PPE screen under the cursor.
Banner	All except Screen Stack	Displays the Banner screen.
BFiles	Deck List Modification List	Displays the Build Files screen.
Bkw	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the first object displayed becomes the last object displayed.

(Continued)

PPE Function Key Summary

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Bprocs	Build Errors Build Files Deck List Environment Description Library Lists Modification List Run Files Tailor Options	Displays the Build Processors screen.
BuiD	Deck List	Builds each selected deck (if the deck is expandable).
Build	Deck List FSE session	Builds all expandable decks that have changed since the last build.
ChaBP	Build Processors	Selects the build processor used by subsequent builds.
ChaCM	Modification List	Selects the modification used by subsequent edit sessions.
ChaDC	Environment Description	Selects the highest PPE level whose contents are included in displayed lists.
Create	Deck Creation Modification Creation	Executes the creation request.
	Deck List Modification List Parameter List Library	Displays the Deck Creation, Modification Creation, or Parameter List screen, respectively.

(Continued)

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Delete	Build Files Deck List Library Lists Modification List Run Files Parameter List Library	Deletes the selected objects from the list.
DisREC	FSE session requested from the Build Errors screen	Displays the number of error messages remaining in the deck being edited.
Down	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the object under the cursor is the last object displayed.
EdiDT	Build Processors	Starts an FSE session to edit the selected deck template.
EdiPL	All screens except the Banner or Screen Stack screens	Displays the Parameter List screen to edit the selected parameter list for the selected build processor.
Edit	Build Errors Deck Creation Deck List	Starts an FSE session to edit the selected deck.
	Build Files Run Files	Displays the information in the selected file.
EndM	Build Errors Deck List Modification List	Marks the last object in a range of selected decks.
Errors	Deck List	Displays the Build Errors screen.

(Continued)

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Export	Build Files Run Files	Copies the file selected from the list to another file outside of PPE.
ExtS	Build Errors Deck List	Copies the selected decks from their nearest residence in the higher levels of the PPE hierarchy to your PPE level and sets an interlock on each deck.
First	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the beginning of the list is displayed and the cursor is on the first object in the list.
ForST	FSE session	Executes the source text formatter.
Fwd	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the last object displayed becomes the first object displayed.
Global	Parameter List Library	Selects the parameter list under the cursor as the parameter list used by subsequent builds.
Help	All screens	Displays information about the keyword under the cursor.

(Continued)

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Home	All screens except the Banner screen	Positions the cursor at the beginning of the home line.
Import	Build Files Run Files	Copies a file from outside of PPE to a file in the Build Files or Run Files subcatalog.
	Environment Description	Combines the specified SCU source library outside of PPE with the PPE source library at this level. No interlocks are set.
Insert	Library Lists	Inserts a blank list entry before the list entry under the cursor.
Last	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the end of the list is displayed and the cursor is on the last object in the list.
LLists	Build Errors Build Files Build Processors Deck List Environment Description Modification List Run Files Tailor Options	Displays the Library Lists screen.
Locate	Deck List Modification List	Searches the list for the first name containing the string of characters and positions the cursor on that name.

(Continued)

PPE Function Key Summary

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Lookup	FSE session	Searches for the word under the cursor in the index of the online manual for the language and, if found, displays the associated online manual screen.
Lower	Deck List Environment Description Modification List	Lowers the display ceiling one level.
Mark	Build Errors Deck List Modification List	Marks the first object in a range of selected decks.
NxtErr	FSE session	Locates the next highlighted error message; deletes the current highlighted error message.
PLists	All screens except Banner, Screen Stack, and Parameter List	Displays the Parameter List Library screen.
Print	Build Files Run Files	Prints the selected file using the site-provided PRINT command; otherwise, the standard SCL PRINT_FILE command is used.
Quit	All screens	Ends the PPE session. Your next PPE session will begin where this one ends.
Raise	Deck List Environment Description Modification List	Raises the display ceiling one level.
Refrsh	All screens	Clears and replots the PPE screen.
Reset	Parameter List	Sets the parameter under the cursor to its default value.

(Continued)

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
RFiles	Deck List Modification List	Displays the Run Files screen.
Run	Run Files	Loads and executes the product.
SCUObj	Deck List Environment Description Modification List	Displays the Deck List or Modification List screen.
Skip	FSE session	Locates the next line with an error. Deletes all error messages associated with the same source line as the currently highlighted message.
Stack	All screens except Banner and Screen Stack	Displays the Screen Stack screen.
Tailor	Build Errors Build Files Build Processors Deck List Environment Description Library Lists Modification List Run Files	Displays the Tailor Options screen.
TraS	Deck List	Transmits the selected decks, copying them to the next higher PPE level and deleting them from the lowest level in which it resides.
UnMark	Build Errors Deck List Modification List	Removes all marks from the list.

(Continued)

PPE Function Key Summary

(Continued)

Function Key Label	Screen on Which the Label Appears	PPE Operation
Undo	Build Files Deck List Library Lists Modification List Run Files Parameter List Library	Removes the effect of the last logical deletion operation, restoring the logically deleted objects.
Up	Build Errors Build Files Deck List Environment Description Library Lists Modification List Parameter List Parameter List Library Run Files Screen Stack	Repositions the list so that the object under the cursor is the first object displayed.

Using PPE F

This chapter describes some procedures that can help you use PPE. These procedures include:

- Moving existing SCU based software to a PPE hierarchy
- Using PPE to begin a new project
- Using NOS/VE from within PPE
- Direct use of SCU
- Customizing PPE function key assignments

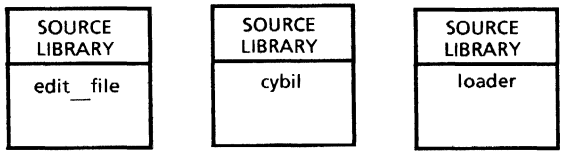
This chapter does not describe how to perform the PPE operations. See chapter 3, PPE Operations, for descriptions of how to perform the primary PPE operations. See chapter 4, PPE Quick Reference, for comprehensive descriptions of all PPE screens and commands. See appendix E, PPE Function Key Summary, for a description of all default function key labels and their use.

Moving Existing SCU Based Software to a PPE Hierarchy

The following steps show how to move existing SCU based software into PPE environment catalog hierarchies.

1. Divide your software into categories. Make sure that each category is a managable size. If a category is too large to manage, subdivide it until all subdivisions are managable. Once you have decided how to categorize the software, clear all deck interlocks and combine the source text for each category into an SCU library reserved for that category.

   The most common method of division is by product. For example, if your site offers an editor, a compiler, and a loader, combine the source for each onto separate libraries as illustrated below.



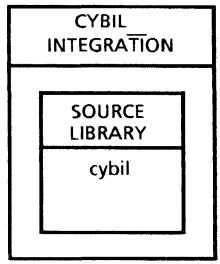
2. Use PPE to create a unique environment catalog for each product category. Each environment catalog is the topmost level of the product category environment catalog hierarchy. Import the source library for the product category into its environment catalog using the IMPORT_SOURCE_LIBRARY command from the Environment Description screen. See Chapter 4, PPE Quick Reference, for information about the Environment Description screen.

3. After importing a source library, specify information identifying the level and the owner of its environment catalog on the Environment Description and Tailor Options screens. For example, specify the following information for each environment catalog created:

   - A descriptive name for the level, like MASTER, TOP, or INTEGRATION. These names indicate that the catalog is the topmost level of its hierarchy.
   - The build processor. Options are NOS/VE COBOL, CYBIL, FORTRAN, and VECTOR_FORTRAN.
   - The owner of the level.
   - An interlock value that uniquely identifies the owner.
   - The build defaults.
   - The run defaults.

   Since each SCU library imported into an environment catalog from existing software belongs at the topmost level of its PPE environment catalog hierarchy, the This level is linked to: field of the Environment Description screen should be left empty.

   For example, create an environment catalog for the CYBIL compiler called CYBIL_INTEGRATION. Then import the CYBIL source library into CYBIL_ INTEGRATION. The environment catalog structure for the CYBIL_INTEGRATION environment catalog hierarchy appears similar to the following:



Initially, no other environment catalogs are linked to CYBIL_INTEGRATION.

4. Use the NOS/VE commands CREATE_FILE_PERMIT and CREATE_CATALOG_PERMIT to create the permits for each environment catalog that needs to extract source from the topmost catalog. These permits allow the members of the project to link their environment catalogs to the topmost catalog, thus forming a PPE environment catalog hierarchy. See appendix D, Environment Catalog Structure, for more information about environment catalog hierarchies.

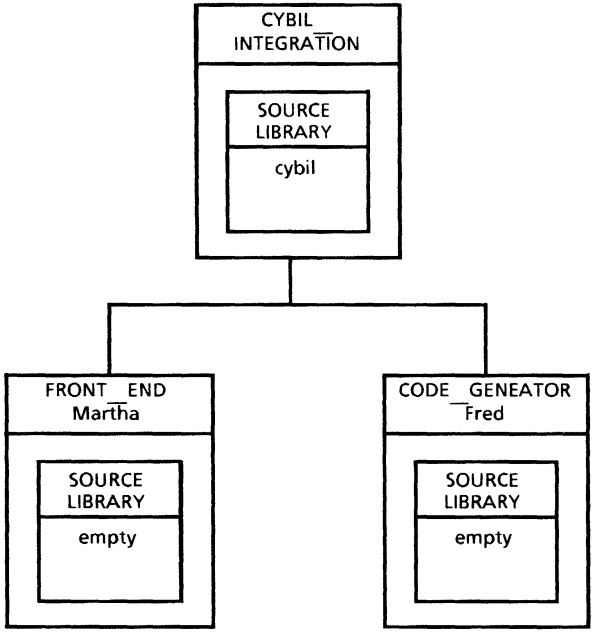
   For example, if Fred and Martha belong to the CYBIL project, create file permissions allowing them to link their environment catalogs to the CYBIL_INTEGRATION catalog by executing the following commands:

```
set_working_catalog $user.cybil_integration

    create_file_permit file=$description group=user ..
    user=(fred,martha) ..
    access_modes=read share_mode=none

    create_file_permit file=source_library group=user ..
    user=(fred,martha) ..
    access_modes=(read, modify, cycle, control) ..
    share_mode=none application_information='I1'

    create_file_permit ..
    file=nodes.load.$object_library.build_files.$object_library ..
    group=user user=(fred,martha) access_modes=(read, execute) ..
    share_mode=none
```

   Each analyst creates an environment catalog for each product area maintained. Then each each analyst's catalog is linked to the corresponding project level environment catalog or to the highest level of the hierarchy. Catalog and file permits are not required for the analyst's environment catalog since it is the lowest level of the environment catalog hierarchy.

   For example, if Martha is responsible for the front end of the CYBIL compiler, she would create the environment catalog FRONT_END. If Fred is responsible for the code generator for CYBIL, he would create the environment catalog CODE_GENERATOR. Using the permissions established above, they would link their catalogs to the CYBIL_INTEGRATION catalog, creating an environment catalog hierarchy. Since Martha and Fred are at the bottom of the hierarchy, they do not need to establish permits to their environment catalogs.

The CYBIL_INTEGRATION environment catalog hierarchy appears similar to the following:



Fred and Martha can now extract decks from the CYBIL_INTEGRATION catalog, update the decks in their environment catalogs, and transmit the changed source back to the CYBIL_INTEGRATION catalog.

Project leaders desiring control over which modifications to a product are transmitted to the highest level can create an intermediate environment catalog to which all project members must transmit their code. This catalog can be linked to the project level catalog or the top most catalog in the hierarchy. Then the project leader can select which modifications to send to the highest level of the hierarchy. File and catalog permits must be created for the intermediate catalog to allow project members to link their catalogs to it.

More than one project level environment catalog can be created. When this is done, each level typically represents some logical division of the work done on the product. For example, a project responsible for a compiler might divide the work into the subareas "front end" and "code generator". Thus modifications to these areas are controlled separately.

Using PPE to Begin a New Project

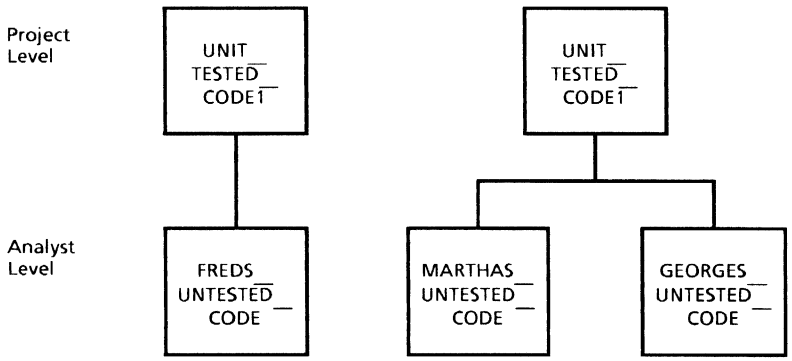
There are many ways to begin a new project using PPE. The method you use is largely dependent on the number of analysts initially involved at the start of the project and how quickly they will need to share source and object code.

On a project starting with one or two analysts, you might elect to create an environment catalog for each analyst. These analysts then work independently of each other (avoiding duplication of deck names, variable names, and so forth) until they need to share resources. This approach requires analysts who work well together and who do not need to share resources.

For example, if Fred and Martha are beginning a new project, each would create an environment catalog from which to begin the project. The catalogs might appear similar to the following:

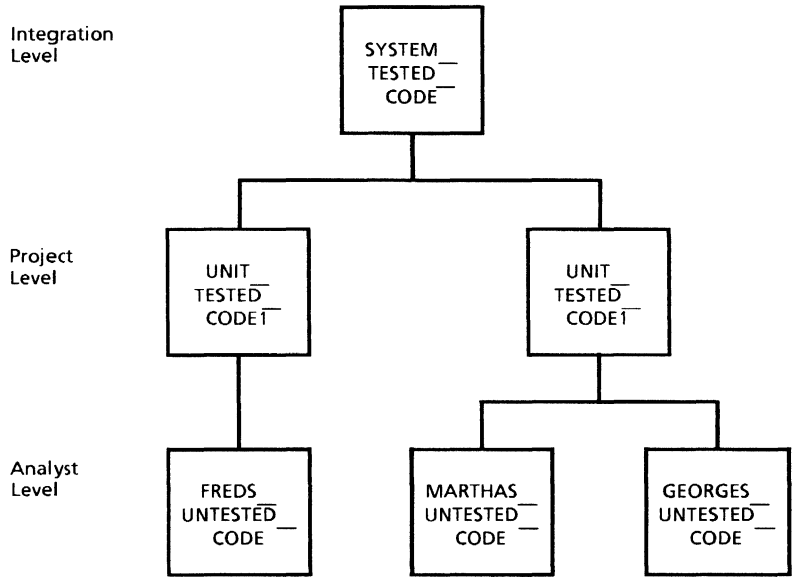
When the project requires more coordination among analysts, one or more project level environment catalogs can be created. Then each analyst's catalog can be linked to a project level catalog, the source from which is transmitted to the project level catalog. All further work on the project occurs within the project environment catalog hierarchy.

For example, two project level catalogs and an additional member may be required to continue Fred and Martha's project. The environment catalog hierarchy that results from linking these environment catalogs might appear similar to the following:



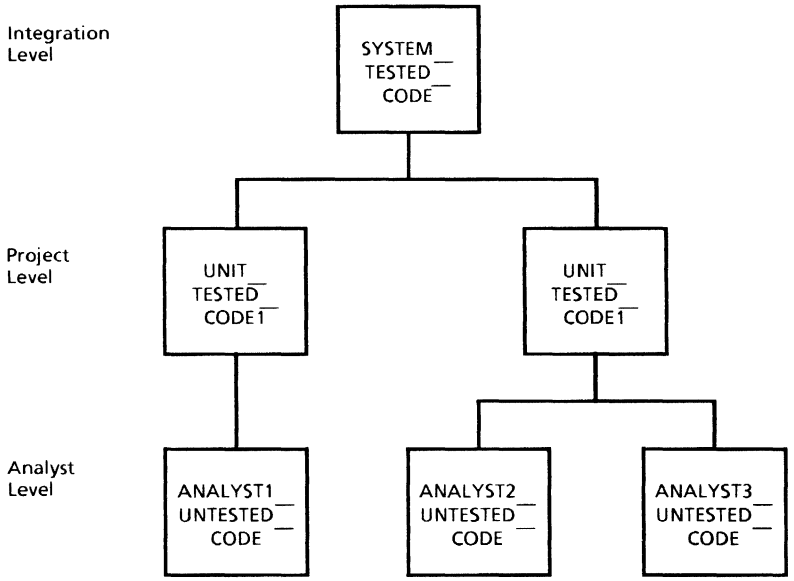
Development continues at this stage of the project until enough of the product is created and tested to bring in system integrators. Then create a top most integration level environment catalog and link the project level catalogs to it. Transmit the project level source to the new top level of the environment catalog hierarchy. At this point, the structure of the new project is indistinguishable from that of a project that began using PPE after the maintainence phase. Developers can extract, edit, and transmit decks as required to finish the project. If more project levels or more analysts are required, additional environment catalogs can be created and linked to the hierarchy.

The hierarchy that results from linking the project level catalogs to a top level integration catalog is similar to the following:



A project starting with many analysts who need to share resources in the near future should elect to create environment catalogs from the top down as described in the discussion on maintaining existing SCU based software. The source libraries in this case would all be initially empty. They become filled as source is created and transmitted up the hierarchy. As analysts join and leave the project, new environment catalogs are linked to and separated from the environment catalog hierarchy.

A project starting with a full environment catalog hierarchy might start with a hierarchy that appears similar to the following:



Using NOS/VE From Within PPE

All NOS/VE commands available prior to starting a PPE session are also available during the PPE session. To execute a NOS/VE command from within a PPE session, move the cursor to the home line of the screen, type the command, and press the carriage-return key. Command utilities such as CREATE_OBJECT_LIBRARY can be executed from within PPE. While you are using a command utility, it controls the terminal; on exit from the utility, PPE regains control of the terminal.

A NOS/VE command executed from the home line performs in the style communicated to NOS/VE through the CHANGE_INTERACTION_STYLE command. If you set the interaction style to SCREEN and execute EDIF $user.xyz on the home line, the NOS/VE File Editor comes up in full-screen mode. If you execute a command that is not a full-screen command and the command causes NOS/VE to write data to the terminal, the data is displayed. A prompt then appears requesting you to press NEXT. Press the carriage-return key to return to PPE.

Regardless of interaction style, if the status parameter of a NOS/VE command executed from the home line is not specified and the command terminates abnormally, PPE displays the abnormal status in a window after it regains control of the terminal.

To facilitate the creation and maintenance of files contained in the environment catalog, PPE manipulates the working catalog. The working catalog is set to the environment catalog when you view most of the screens in the PPE screen hierarchy. The exceptions are the Build Files and Build Errors screens, which are set to the BUILD_FILES subcatalog and the Run Files screen which is set to the RUN_FILES subcatalog. Therefore, if the environment catalog is $USER.XYZ, the Build Files screen is displayed, and you execute the command DISPLAY_OBJECT_LIBRARY $OBJECT_LIBRARY on the home line of the screen, the contents of the object library $USER.XYZ.NODES.LOAD.$OBJECT_LIBRARY.BUILD_FILES.$OBJECT_LIBRARY are displayed. See Appendix D, PPE Catalog Structure, for information about the PPE catalog structure.

Direct Use of SCU

PPE uses the NOS/VE Source Code Utility to provide many of its features. As a result of this usage, an SCU session is active the entire time PPE runs. All commands issued by you or PPE interact with the SCU source library contained in the environment catalog.

Although all SCU commands are available to you from the PPE home line, the END_LIBRARY and USE_LIBRARY SCU commands should not be executed because they may interfere with PPE. Direct use of these commands can cause a source code library PPE is interacting with to differ from that which the SCU session is interacting with. Differing source libraries for PPE and SCU can cause errors to occur.

Much of the value of PPE lies in its ability to automatically interact with the proper source code libraries. Avoid using END_LIBRARY and USE_LIBRARY to guarantee that this value is preserved.

Customizing Function Key Assignments

PPE uses automatic menu assignment to associate commands with particular terminal key sequences. A terminal key sequence is called a function key, and the function key labels at the bottom of a PPE screen are called the menu items. The collection of menu items is called the screen menu.

For any terminal, the primary goals of the menu assignment are to associate as many PPE commands as possible with function keys and to keep the function key assignments as consistent as possible from screen to screen. Unfortunately, these goals can conflict, resulting in one or more commands with function key assignments that vary from screen to screen.

The PPE command to function key assignment variation is controlled by customizing the PPE function key assignments. You can assign commands to particular function keys and display menu items with labels different from those provided by PPE. The information about function key assignments is stored in a message module. All message modules are stored in the object library $SYSTEM.OSF$COMMAND_LIBRARY and the name of a message module is given by piM$natural_language, where pi represents the product identifier and natural_language is the name of your natural language. You must supply the product identifier and natural language to select a message module. For the PPE function key assignment menu, the product identifier is SW and the natural language is US_ENGLISH.

To obtain a file containing the English PPE message module, execute the following commands:

```
create_object_library
add_module m=swm$us_english l=$system.osf$command_library
generate_library l=mm_file f=message_module
edit_file mm_file
```

MM_FILE is a file that contains CREATE_OBJECT_LIBRARY subcommands specifying the PPE menu item labels and function key assignments. You edit MM_FILE to change function key assignments and menu item labels. After editing MM_FILE, use it to generate an object module which you add to the command list. The changed function key assignments and menu labels become effective on entry to PPE.

For example, to force the EDIT_DECK command on the Deck List screen to appear on function key F1 with the label Modify, edit MM_FILE as follows:

1. Change the following lines in MM_FILE from:

```
.
.
.

CREATE_APPLICATION_MENU NAME=DECK_LIST
.
.
.

CREATE_MENU_ITEM CLASS='Actions' SHORT_LABEL=' Edit '
LONG_LABEL='Edit Deck'
.
.
.
```

.. to the following lines assigning the EDIT_DECK command to function key F1 and changing the short label to Modify from Edit:

```
.
.
.

CREATE_APPLICATION_MENU NAME=DECK_LIST
.
.
.

CREATE_MENU_ITEM CLASS='Actions' SHORT_LABEL='Modify'
   LONG_LABEL='Edit Deck' KEY=F1
.
.
.
```

2. After modifying the menu item, exit the editor and execute the commands:

```
include_file  mm_file
generate_library l=my_ppe_menus
quit
set_command_list a=my_ppe_menus
```

Upon entering PPE, the change you made assigns the EDIT_DECK function to key F1 with the label Modify on the Deck List screen.

Although the function key assignments and menu labels can be changed, the order of the menu items must remain unchanged. The order of the menu items in the file determines which commands are affected by your label and key assignment changes. If you change the order of the CREATE_MENU_ITEM commands, the association of commands to function keys is also changed.

The automatic menu assignment operates on menu items for which no key assignment is forced. This means that forcing some but not all key assignments might result in undesired variations of function key assignments from screen to screen.

Index

C

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

Professional Programming Environment for NOS/VE Usage 60486613 B

We would like your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who Are You?

___ Manager
___ Systems Analyst or Programmer
___ Applications Programmer
___ Operator
___ Other _____

How Do You Use This Manual?

___ As an Overview
___ To learn the Product/System
___ For Comprehensive Reference
___ For Quick Look-up

Which Are Helpful to You?

___ Command and Function Summaries
___ Related Manuals Appendix
___ Online Quick Reference
___ Other _____

What programming languages do you use? _____

Which are helpful to you? ___ Parameter Summary (inside cover) ___ Related Manuals Page
___ Character Set ___ Other _____

How Do You Like This Manual? Check those that apply.

Yes	Somewhat	No	
___	___	___	Is the manual easy to read (print size, page layout, and so on)?
___	___	___	Is it easy to understand?
___	___	___	Is the order of topics logical?
___	___	___	Are there enough examples?
___	___	___	Are the examples helpful? (___ Too simple ___ Too complex)
___	___	___	Is the technical information accurate?
___	___	___	Can you easily find what you want?
___	___	___	Do the illustrations help you?
___	___	___	Does the manual tell you what you need to know about the topic?

Comments? If applicable, note page number and paragraph.

___ **Check here if you want a reply**

Continue on other side

From:

Name _____ Company _____

Address _____ Date _____

_____ Phone No. _____

_____

Please send program listing and output if applicable to your comment.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 8241 MINNEAPOLIS, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Technology and Publications Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD FOLD

Comments (continued from other side)