HP NonStop Pathway/iTS SCUP Reference Manual

Abstract

This manual describes the SCREEN COBOL Utility Program (SCUP), a utility that enables SCREEN COBOL application programmers to control and manipulate SCREEN COBOL object files and to convert SCREEN COBOL object programs to web clients.

Product Version

Pathway/iTS 1.0 and 1.1

Supported Release Version Updates (RVUs)

This publication supports J06.03 and all subsequent J-series RVUs, H06.03 and all subsequent H-series RVUs, and G05.00 and all subsequent G-series RVUs, until otherwise indicated by its replacement publications.

Part Number	Published
520269-002	November 2010

Document History

Part Number	Product Version	Published
127342	Pathway/TS D42	August 1996
426747-001	Pathway/iTS 1.0	October 2000
520269-001	Pathway/iTS 1.0 Update	June 2001
520269-002	Pathway/iTS 1.0 and 1.1	November 2010

Legal Notices

Printed in the US

HP NonStop Pathway/iTS SCUP Reference Manual

Index Figures Tables

2. SCUP Commands (continued)

What’s New in This Manual

Manual Information

Abstract

This manual describes the SCREEN COBOL Utility Program (SCUP), a utility that enables SCREEN COBOL application programmers to control and manipulate SCREEN COBOL object files and to convert SCREEN COBOL object programs to web clients.

Product Version

Pathway/iTS 1.0 and 1.1

Supported Release Version Updates (RVUs)

This publication supports J06.03 and all subsequent J-series RVUs, H06.03 and all subsequent H-series RVUs, and G05.00 and all subsequent G-series RVUs, until otherwise indicated by its replacement publications.

Part Number	Published
520269-002	November 2010

Document History

Part Number	Product Version	Published
127342	Pathway/TS D42	August 1996
426747-001	Pathway/iTS 1.0	October 2000
520269-001	Pathway/iTS 1.0 Update	June 2001
520269-002	Pathway/iTS 1.0 and 1.1	November 2010

New and Changed Information

Changes to the 520269-002 manual:

- Added supported release statements.
- Added the Pathway/iTS 1.1 version.
- Updated Operational Details information for CONVERT command on page 2-20.
- Added troubleshooting information for CONVERT command on page 2-22.
- Changed all instances of “Compaq” to “HP”.
- Changed all instances of “NonStop Himalaya system” to “NonStop system”.

- Changed “Compaq NonStop Kernel Operating System” to “HP NonStop Operating System”.

Changes to the 520269-001 Manual

The Compaq *NonStop*™ Pathway/iTS product was formerly called Pathway/TS. For the Pathway/iTS 1.0 independent product release, the product was renamed to conform to current Compaq product naming standards and to reflect the internet (web client) capabilities of the product. After the first reference to the product name in each section of this manual, subsequent references use the shortened form of the name, Pathway/iTS.

Product Changes

All technical changes are marked with change bars.

- Removed the requirement that SCOBOL source files be in the same volume and subvolume as the object files. Added a disclaimer notice to the CONVERT command regarding the user’s authorization to convert SCOBOL files. See Converting SCREEN COBOL Programs to Web Clients on page 1-3.
- Updated the CONVERT command.
  - Updated text throughout the command description.
  - Added new subsections:

    Limits Section Parameters on page 2-17
    Summary Information on page 2-19
    Example Summary on page 2-19
    Restrictions on page 2-21
    Troubleshooting on page 2-21
- Converter messages: Added new messages and updated the cause, effect, and recovery descriptions for some messages.

Corrections and Enhancements to the Manual

The following enhancements have been made to the material in this manual:

- References to Compaq trademarks have been updated.
- Miscellaneous terminology changes and editorial corrections have been made.

About This Manual

This manual describes the SCREEN COBOL Utility Program (SCUP), which enables SCREEN COBOL application programmers to control and manipulate SCREEN COBOL object files. The manual includes SCREEN COBOL object file handling concepts, SCUP commands available for object file manipulation, and examples that demonstrate general SCUP usage.

Related Documentation

The Pathway environment is supported by three products:

HP *NonStop*™ Pathway/iTS
HP *NonStop*™ Transaction Services/MP (NonStop TS/MP)
HP *NonStop*™ Pathway/XM

The following manuals may be useful:

Pathway/iTS SCREEN COBOL Reference Manual	Describes the SCREEN COBOL programming language, which is used for writing programs that define and control terminal displays or intelligent devices for online transaction processing applications running in a PATHMON environment.
Pathway/iTS Web Client Programming Manual	Describes how to convert SCREEN COBOL requesters to web clients, explains how to build and deploy those clients, and also provides the information Java developers and web designers need to to modify and enhance the Java and HTML portions of the converted clients.
Pathway/iTS TCP and Terminal Programming Guide	A guide for programmers who are writing SCREEN COBOL requesters to be used in Pathway applications.
Pathway/iTS System Management Manual	Describes the interactive management interface to the Pathway/iTS product and describes how to configure and manage Pathway/iTS objects.
NonStop™ TS/MP System Management Manual	Describes the interactive management interface to the NonStop TS/MP product and describes how to configure and manage NonStop TS/MP objects. This interface is used together with the Pathway/iTS management interface to configure and manage a Pathway environment.
Pathway/XM System Management Manual	Describes the higher-level management interface provided by the Pathway/XM product, which you might want to use to configure and manage your Pathway environment rather than use the management interfaces provided by Pathway/iTS and NonStop TS/MP.

Pathway/iTS Management Programming Manual	Describes the management programming interface for Pathway/iTS objects in the PATHMON environment.
Pathway Products Glossary	Defines technical terms used in this manual and in other manuals for the Pathway products: Pathway/iTS, NonStop TS/MP, and Pathway/XM.
Operator Messages Manual	Describes all messages that are distributed by the Event Management Service (EMS), including those generated by NonStop TS/MP and Pathway/iTS processes.

Notation Conventions

General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *`Italic computer type`* letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

```
pathname
```

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name

INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
LIGHTS [ ON             ]
       [ OFF            ]
       [ SMOOTH [ num ] ]

K [ X | D ] address-1
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list may be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address-1 [ , new-value ]...
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] CONTROLLER

   [ , attribute-spec ]...
```

Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

`p-register`

`process-name`

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

`Event number =` *`number`* `[ Subject =` *`first-subject-value`* `]`

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

`LDEV` *`ldev`* `[ CU %`*`ccu`* `| CU %... ] UP [ (`*`cpu`*`,`*`chan`*`,%`*`ctlr`*`,%`*`unit`*`) ]`

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list might be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LBU { X | Y } POWER FAIL
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The %þnotation precedes an octal number. The %Bþnotation precedes a binary number. The %Hþnotation precedes a hexadecimal number. For example:

```
%005400
```

`P=%`*`p-register`* `E=%`*`e-register`*

Change Bar Notation

Change bars are used to indicate substantive differences between this manual and its preceding version. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

> The message types specified in the REPORT clause are different in the COBOL85 environment and the Common Run-Time Environment (CRE).
>
> The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to docsfeedback@hp.com.

Include the document title, part number, and any comment, error found, or suggestion for improvement concerning this document.

1 Introduction to SCUP

Introduction to SCUP

The SCREEN COBOL Utility Program (SCUP) simplifies your task of maintaining SCREEN COBOL object library files. With SCUP, you can control the SCREEN COBOL object files by issuing commands to do the following:

- Convert a group of programs in a SCREEN COBOL library into a web client consisting of Java classes and HTML pages
- Display information about the library files or about programs in a SCREEN COBOL library
- Control access by a terminal control process (TCP) to programs in a SCREEN COBOL library
- Copy programs from one SCREEN COBOL library to another
- Delete programs from a SCREEN COBOL library
- Reclaim file space by compressing SCREEN COBOL library files

SCUP operates on the SCREEN COBOL object library without recompiling the source program.

A SCREEN COBOL object library consists of up to three files generated by the SCREEN COBOL compiler.

COD	Code file
DIR	Directory to the code file
SYM	Symbols table (generated if the SYMBOLS option is selected for compilation)

To compile a SCREEN COBOL program, specify a file-name root that does not exceed five characters. The compiler generates the actual file names by appending COD, DIR, and SYM to the file-name root. For example, executing the command

```
SCOBOLX /IN TESTFILE/ MANUF
```

causes SCREEN COBOL to compile the program in TESTFILE, and to generate directory entry MANUFDIR and object code file MANUFCOD.

Optionally, you can specify the SYMBOLS compiler command that causes the SCREEN COBOL compiler to build a symbol table into the object code. This symbol table is used during program execution by the HP *Inspect* symbolic program debugging tool, and is also required when converting SCREEN COBOL programs to web clients. If you request a program symbol table, the SCREEN COBOL compiler generates an additional file. The symbol table file is given the file-name root specified when the program is compiled, with SYM appended.

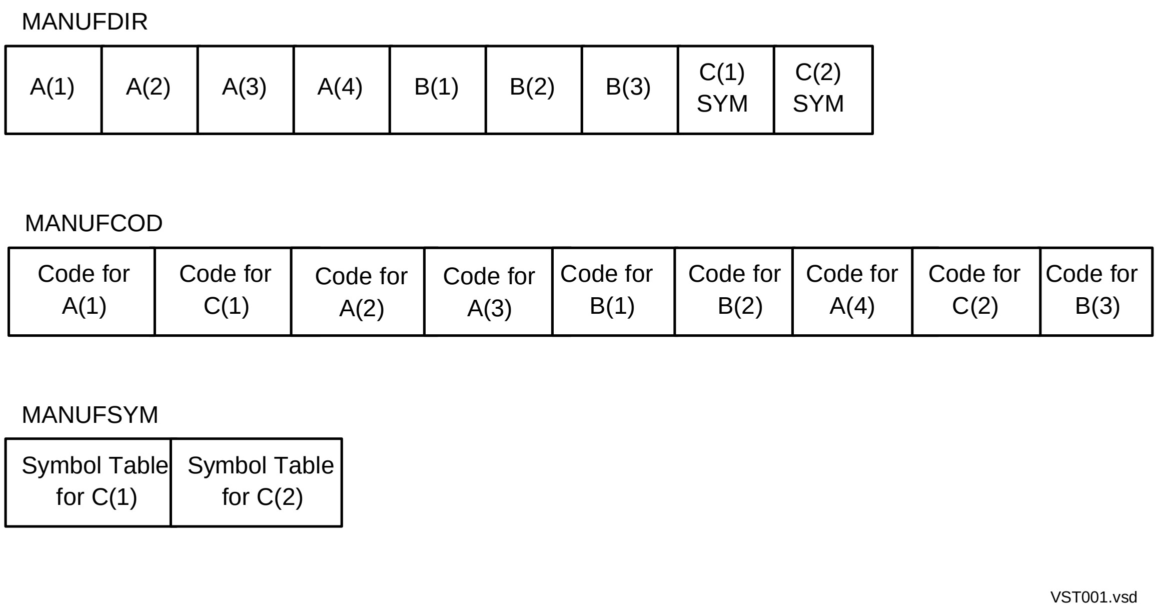
For example, executing the command

```
SCOBOLX/IN TESTFILE/ MANUF;
```

causes SCREEN COBOL to compile SYMBOLS the program in TESTFILE, and to generate directory entry MANUFDIR, object code file MANUFCOD, and symbol table MANUFSYM.

SCUP automatically processes all three associated object files. A diagram of SCREEN COBOL object files is shown in Figure 1-1.

Figure 1-1. Representation of SCREEN COBOL Object Files



The SCREEN COBOL object files can contain several different programs, several versions of a program, and optionally, several versions of a symbol table. The compiler does not overwrite the previous contents of the files when it adds a program and the associated symbol table to the object files. Instead, the compiler appends the code to the code file, adds the symbol table to the symbol table file, and inserts the program name and version number in the directory file.

The version of the program and its symbol table is 1 if previous versions do not exist in the specified SCREEN COBOL object file. For example, TEST-PROG(2) refers to the second version of the program called TEST-PROG and the second version of the symbol table for that file, that is, the second time the program called TEST-PROG is added to the specified object file either through compilation or through the SCUP copy function.

The highest version number always represents the last version added to the object file.

Converting SCREEN COBOL Programs to Web Clients

SCUP lets you convert a SCREEN COBOL application (a group of related program units within an object library) to a web client without making any changes to the SCREEN COBOL source code. Requester-based business logic is converted to Java classes; screen components are converted to HTML pages with embedded JavaScript code. The converter also generates command files to help you build and deploy the converted web client. The compiled Java objects and HTML pages can be deployed on any web server hosted on the HP NonStop operating system.

DISCLAIMER NOTICE. While this SCUP utility is a tool, you must have the right to modify third-party files. In no way does the licensing or use of our tool grant you rights, which you may not have vis-a-vis a third party.

Any valid block-mode requester programs for supported terminal types (6520, 6530, 6540, and IBM 3270) can be converted. (Intelligent device support (IDS) requesters and requesters with message sections cannot be converted.) The specified program units are converted in such a way that most of the functionality of the original SCREEN COBOL program remains unchanged.

This manual describes the syntax and semantics of the CONVERT command that performs the conversion and the error messages that can be issued by the converter. Further information about the resulting web clients and how to deploy them is given in the *Pathway/iTS Web Client Programming Manual*.

You can configure and manage Pathway applications containing converted web clients by using either HP *NonStop*™ Pathway/XM or PATHCOM, but it is easier to do so with Pathway/XM. Configuration and management information needed for Pathway/iTS web clients is provided in the *Pathway/iTS System Management Manual* and the *Pathway/XM System Management Manual*.

Displaying Information

SCUP operations display information about the object files themselves and about the program units.

The information displayed about the object files is:

- Library access setting (ON or OFF) that applies to programs added to the object files
- Extents for the object files in the following form:

  file(primary-extent,secondary-extent)

  File is DIR, COD or SYM. Extents are in 2048-byte pages.

The information displayed about program units includes:

- Name of the program (taken from the PROGRAM-ID paragraph in the Identification Division)
- Access flag for the program unit
- Version of the program
- Location and size in the code file where the program object code is stored
- Flag that indicates whether a symbol table exists for the SCREEN COBOL program

When displaying information about the contents of a SCREEN COBOL object file, SCUP retrieves some of the information from the directory file and retrieves the date the program was compiled from the code file.

Controlling TCP Access

The TCP needs to locate programs in the following two situations. In each case, the TCP examines the directory file to find the entry for the named program:

- When starting a terminal, the TCP must locate the INITIAL program for that terminal.
- When a SCREEN COBOL program CALL statement is executed, the TCP must locate the called program.

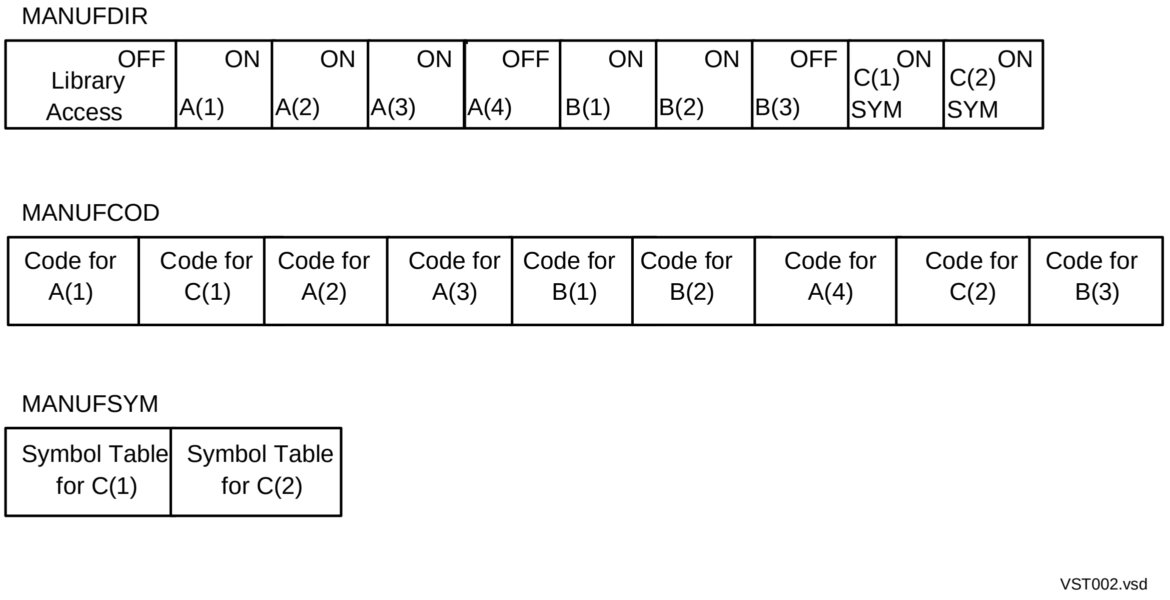
SCUP lets you control which programs and versions of programs the TCP can execute by setting an access flag for each program. Each entry in the directory file has an access flag that you can turn ON or OFF through SCUP. The TCP recognizes the flag settings and ignores those entries for which access has been denied. The TCP looks for the highest version number of the program with the access flag ON.

In addition to the access flag for each entry, the library has an access flag (ON or OFF) that applies to new programs added to the object files. If the library access is set OFF, any program added to the library has access OFF. The library access setting does not affect entries already in the library.

SCUP allows you to alter the access flag for program units and the access flag for the library.

An example of a directory file with access flags shown is illustrated in Figure 1-2.

Figure 1-2. Directory File with Access Flags Shown



Copying Programs

You can copy programs from one object file to another. SCUP copies the programs and assigns version numbers in the order you specify, thus giving you control over the priority assigned to versions of a program.

If the program to be copied has an associated symbol table, SCUP copies the symbol table to the destination symbol table file.

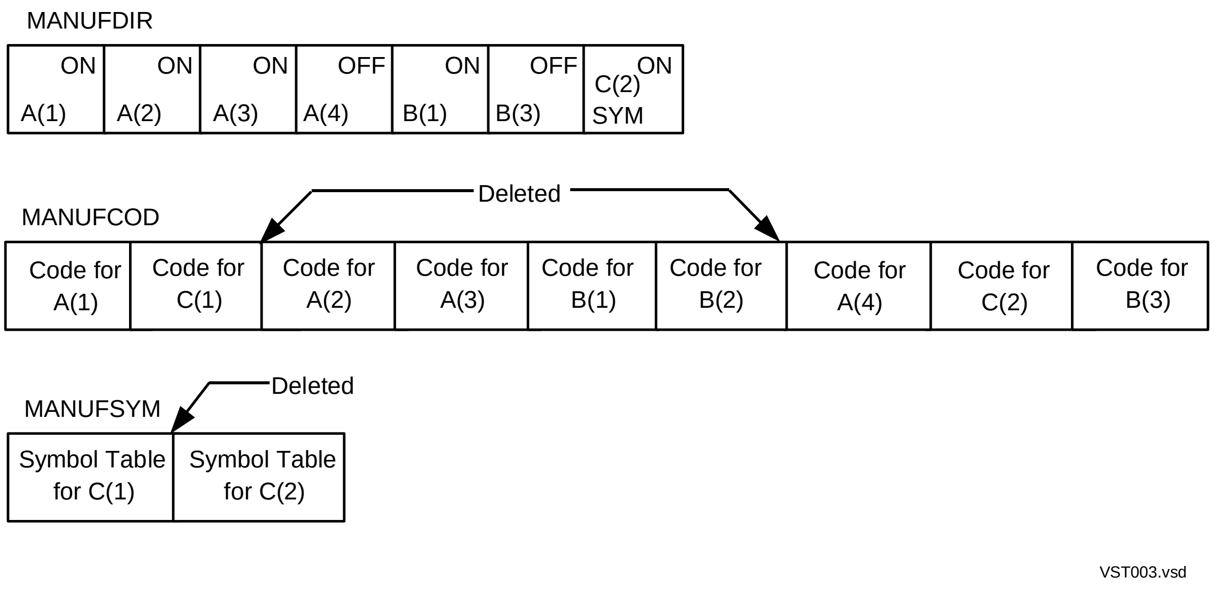
Deleting Programs

You can delete programs from the object files. SCUP deletes programs by removing the entry for the program in the directory file.

When you delete a program, SCUP does not alter the code file until the file is compressed; thus, a program that is currently in use by a TCP can be deleted while the program processing continues. Any subsequent reference to the deleted program either accesses another version of the program or fails if the deleted version was the last one. If the deleted program has an associated symbol table, the entry for the table is deleted from the directory file, but the symbol table file is not altered until the file is compressed.

An example of code and directory files with deleted programs is shown in Figure 1-3.

Figure 1-3. Code and Directory Files With Deleted Programs

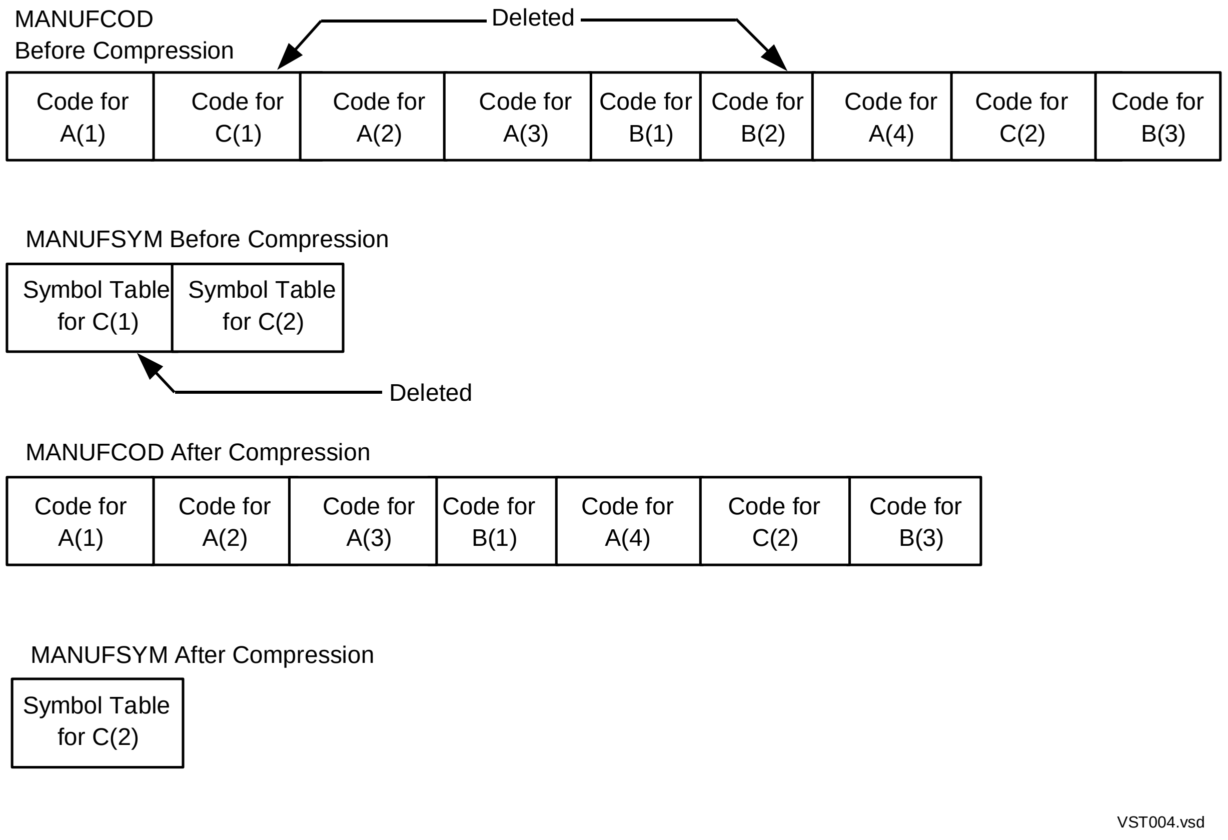


Compressing Programs

The space occupied by a deleted version of a program remains part of the code file. The space occupied by an associated symbol table remains part of the symbol table file. You can use SCUP to reclaim the space left by the deleted programs and symbol tables. SCUP rewrites the code and symbol table files to pack the programs together. SCUP preserves the access flag of the program, but renumbers the program versions starting with version 1.

Figure 1-4 is an example of a code file and a symbol table file before and after compression.

Figure 1-4. Code and Symbol Table Files Before and After Compression



SCUP Version Compatibility

All the existing SCREEN COBOL object files are completely compatible with new versions of SCREEN COBOL and SCUP that use Inspect, the interactive symbolic debugging utility.

You do not have to recompile SCREEN COBOL programs that were developed before Inspect was available unless you want to use a symbol table. Note, however, that a symbol table is required if you are using SCUP to convert SCREEN COBOL programs to web clients.

2 SCUP Commands

SCUP commands are divided into two categories:

- Utility commands, which are associated with use of the SCUP utility
- Basic commands, which are associated with control of the SCUP utility

A complete list of available commands appears in Table 2-1 and Table 2-2.

Table 2-1. SCUP Utility Commands

Command	Description
ALTER	Changes either the accessibility of SCREEN COBOL program units or the access setting of the SCREEN COBOL library.
ASSUME	Specifies the default entity type SCOBOL-OBJECT.
COMPRESS	Compresses a SCREEN COBOL object file and reclaims space from deleted programs.
CONVERT	Converts a SCREEN COBOL application (a group of program units within a SCREEN COBOL object file) to a web client consisting of Java classes and HTML pages.
COPY	Copies a set of programs from one SCREEN COBOL object file to another.
DELETE	Removes programs from a SCREEN COBOL object file directory.
FILE	Specifies the default file name for the SCREEN COBOL object file used within SCUP commands.
INFO	Displays information about SCREEN COBOL object files and programs.

Table 2-2. SCUP Basic Commands

Command	Description
CMDSYS	Sets the default system for expansion of any file names except obey file names.
CMDVOL	Sets the default volume and subvolume for expansion of any file names except obey file names.
EXIT	Stops SCUP.
FC	Provides the ability to edit or to repeat a command line.
HELP	Displays the syntax of SCUP commands.
OBEY	Causes commands to be read from a specified file.
OBEYSYS	Sets the default system for expansion of obey file names.
OBEYVOL	Sets the default volume and subvolume for expansion of obey file names.
OUT	Directs the output listing to a specified file.
SYSTEM	Sets the default system for expansion of any file names.
VOLUME	Sets the default volume and subvolume for expansion of any file names.

All commands can include comments. Comments must be delimited by a space followed by the double hyphen (--) symbol. The double hyphen symbol and all subsequent characters up to the next double hyphen, or the end of the effective record, are ignored.

SCUP Utility Commands

SCUP utility commands allow you to access and manipulate compiled programs in the SCREEN COBOL object files. These commands let you do the following:

- Display information about programs
- Copy programs from one object file to another
- Delete programs from an object file
- Logically delete and reinstate programs
- Compress existing programs within an object file
- Convert a group of programs, represented by an object file, to a web client

SCUP Basic System Commands

The SCUP basic system commands are standard commands included in many other command interfaces on HP NonStop™ systems, such as PATHCOM and PXMCOM. Many of the basic system commands pertain to file-name expansion.

Each disk file in a NonStop system is identified by a unique, symbolic file name. The name, and therefore the location, of a disk file is determined in four parts:

`\`*`system-name`*	identifies a specific NonStop system (node) within a HP *Expand* network.
`$`*`volume-name`*	identifies a physical disk pack mounted on a disk unit.
`subvolume-name`	identifies a disk file as a member of a related set of files as defined by the user.
`disk-file-name`	identifies a particular file within the subvolume.

File names of disk files are represented to subsystem programs by these parts concatenated into a contiguous string with each part separated from the other by a period:

```
\system-name.$volume-name.subvolume-name.disk-file-name
```

When only a partial file name is supplied as a command parameter, the internal representation of the file name is expanded into the full four-part file name. As a minimum, a partial file name must consist of disk-file-name.

Each process and each device, such as a tape drive or printer, is identified in a similar manner. For example:

```
\TSB.$TAPE1
```

might specify a particular tape drive on system \TSB; when operations are already on that system, only $TAPE1 is required.

A command interpreter for a given user has associated with it a default volume name and a default subvolume name. File-name expansion is accomplished through the use of these default names. If the volume name is omitted, the default volume name is used in its place. If the subvolume name is omitted, the default subvolume name is used in its place.

The three classes of file-name expansion within SCUP are:

- Obey file names
- File names within command parameters
- Object file names

The initial settings used for file-name expansion are taken from the command interpreter default settings, that is, the command interpreter VOLUME command. All file names are kept in the SCUP subsystem in fully qualified network form.

File-Name Expansion Within OBEY files

File-name expansion for file names within an obey file is controlled by the OBEYSYS and OBEYVOL basic commands. OBEYSYS and OBEYVOL set the default system, volume, and subvolume names for the OBEY command.

File-Name Expansion Within SCUP Utility Commands

File-name expansion for file names in SCUP utility commands is controlled by the CMDSYS and CMDVOL commands. CMDSYS and CMDVOL set the default system, volume, and subvolume names for all SCUP commands other than OBEY. Expansion of the OUT file name is included in this type of file-name expansion. This file-name expansion is different from the file-name expansion performed by PATHCOM.

File-Name Expansion for Object-File Names

File-name expansion for a SCREEN COBOL object-file name also is controlled by the CMDSYS and CMDVOL commands. This is a special case of command file-name expansion because the SCREEN COBOL compiler generates three object-file names. The SCREEN COBOL short-form file name has the form of a disk-file name:

```
[\system.][$volume.][subvol.]file-name
```

with the restriction that *file-name* must be five characters or fewer. File-name expansion derives the names of three disk files from the short form name, which serves as the file-name root. The names of the actual disk files are formed by appending COD, DIR, and SYM to the name supplied.

If the optional object file name is omitted from a SCUP command, the default file name is derived from the name specified in the SCUP utility FILE command. The initial setting has the form:

```
\system.$volume.subvol.POBJ
```

The SCREEN COBOL compiler names the object files POBJCOD and POBJDIR within the default system, volume, and subvolume. If you use the ?SYMBOLS compiler command, the SCREEN COBOL compiler names the symbol table file POBJSYM.

Calling SCUP

The SCUP subsystem resides in a file designated

$SYSTEM.SYSTEM.SCUP

The SCUP command issued from the command interpreter calls the SCUP subsystem.

```
SCUP [ / [ IN command-file ] [ , OUT list-file ] / ]
  [ command [ ; command ] ... ]
```

`IN` *command-file*

specifies a file from which commands are to be read. SCUP reads 132-byte records from *command-file* until it encounters an end-of-file or an EXIT command.

If this option is omitted, input is taken from the current input file of the command interpreter; this is typically the home terminal.

`OUT` *list-file*

specifies a file to which the listing is to be written.

If *list-file* is omitted, output is suppressed; this does not prohibit the later use of the basic command OUT. If the entire option is omitted, output is directed to the current output file of the command interpreter; this is typically the terminal.

command

is a SCUP command. When commands are present, SCUP executes the commands and terminates without reading *command-file*.

The following applies to the OUT option of the SCUP command:

- If the OUT file specification is a disk file name and the file already exists, the listing is appended to the file. If the OUT file specification is a disk file name and the file does not exist, an EDIT file is created.
- If the OUT file specification is a line printer or a process, a page eject is performed when the file is opened. A page eject is not performed before the file is closed.
- If the OUT file specification is a magnetic tape, two consecutive file marks are written just before the file is closed.

Interactive Mode

SCUP functions in interactive mode when commands are entered from a terminal keyboard. SCUP prints the product identification message and prompts for a command by printing the question mark symbol (?). When a command is entered, SCUP executes the command and issues another prompt. For example:

```
1> scup
SCOBOL CLIENT OBJECT MANAGER ...
?
```

Two or more commands can be grouped and separated by semicolons (;). For example:

```
?help command-1; command-2
```

is equivalent to

```
?command-1
?command-2
```

You can continue commands and parameters on a subsequent line by ending the line with an ampersand (&) character. A maximum of 132 characters including the ampersand can be entered on each line. The total number of characters for a multiline command cannot exceed 528.

SCUP verifies a command for accuracy before executing a command. If an error occurs, no part of the command is executed. If the BREAK key is pressed during execution of a command, SCUP completes the operation for any one program, then terminates the command. In this way, SCUP maintains complete programs in object files.

Noninteractive Mode

SCUP functions in noninteractive mode when commands are entered through a command file. For example:

```
1> scup / IN command-file /
```

In this example, SCUP reads commands from a command file. When an end-of-file or an EXIT command is encountered, SCUP terminates and the command interpreter is activated.

Command Descriptions

SCUP utility and basic commands are described in the following paragraphs. Command descriptions appear in alphabetic order.

ALTER Command

The ALTER command controls two types of access flags: program unit and SCREEN COBOL library.

The program unit flag designates whether a PATHWAY application can access a specific version of a program and its associated symbol table. The command performs a logical deletion or reinstatement of programs in a SCREEN COBOL object file.

The library flag designates the access setting of all programs subsequently added to the SCREEN COBOL object files.

Use the ALTER command to change these flags as follows:

- Program unit flag. Specify a program name or an asterisk for program names in the command.
- Library flag. Omit the specification of a program name or the asterisk.

```
ALTER [ SCOBOL-OBJECT ] [ file-name ]

  [ ( program-name [ ( version ) ]                  ]
  [    [ , program-name [ ( version ) ] ] ... )    ]
  [                                                 ]
  [ ( * [ ( version ) ] )                           ]

  , ACCESS { ON  }
           { OFF }
```

file-name

is a SCREEN COBOL object file name. If this parameter is omitted, the file specified with the FILE command is used.

`(`*`program-name`*`)`

is a SCREEN COBOL program name. The program name can be a single program name or several program names separated by commas.

`(*)`

is all program names in the object file.

If *`program-name`* or `*` is omitted, the ALTER command changes the access setting for the library in the object file control record. This setting determines the access of all programs subsequently added to the object file.

`(`*`version`*`)`

is the version of the specified program name. *`Version`* can be any of the following:

integer	a single version of a program
*	all versions of a program
+	all but the latest version of a program

If the version parameter is omitted, the latest version of a program is used.

```
ACCESS { ON  }
       { OFF }
```

specifies the access setting (ON or OFF) of either the object file or programs in the object file.

If ACCESS is set OFF for the object file, any program subsequently added to the object file has access set OFF.

If ACCESS is set ON for the object file, any program subsequently added to the object file has access set ON.

If ACCESS is set OFF for any program, the program becomes inaccessible for execution under the PATHWAY system; the program is not again made accessible until ACCESS is set ON.

When executing an ALTER command, SCUP allows shared access to the SCREEN COBOL object file. From the moment the ALTER command completes, any attempt to find the specified programs (either through a CALL statement in a SCREEN COBOL program or through the INITIAL setting when starting a terminal) fails. Any terminal already executing that program continues unaffected. Thus, the program can still be in use, but no new references can be made to it.

The expansion of the object file name uses the default settings specified by the CMDSYS and CMDVOL basic commands.

The following example of the ALTER command makes the latest version of the program test-prog1 and all versions of test-prog2 inaccessible to the PATHWAY system:

```
ALTER scobj (test-prog1,test-prog2(*)), ACCESS OFF
```

The following example of the ALTER command makes the second version of the program test-prog3 accessible to the PATHWAY system:

```
ALTER scobj (test-prog3(2)), ACCESS ON
```

The following example of the ALTER command designates ACCESS OFF for the object file. All programs subsequently added to the object file have ACCESS OFF.

```
ALTER scobj, ACCESS OFF
```

ASSUME Command

The ASSUME command sets the default object type of the SCUP subsystem.

```
ASSUME object-type
```

object-type

is SCOBOL-OBJECT.

Entities for which characteristics can be established belong to classes referred to as object types. (Object types in PATHCOM commands are SERVER, TCP, PROGRAM, and TERM.) SCUP supports only one object type, which is designated SCOBOL-OBJECT. The concept is introduced to allow future extension of the product to other entities.

SCOBOL-OBJECT is a reserved word within SCUP and cannot be used as a file name or program name.

If this command is not specified, SCOBOL-OBJECT is the default setting.

CMDSYS Command

The CMDSYS command sets the default system for expansion of any file names except obey file names.

```
CMDSYS [ \system ]
```

system

is a NonStop system (Expand node) name.

If the CMDSYS command is not issued, the default settings in effect when SCUP was started are used. If the system name is not included in the command, the default is set for local file-name expansion. Note that this is not the same as specifying the local system name. Omitting the system name in this command permits references to long device names; long device names are prohibited in network-form file names.

If the specified system name is invalid, or would be invalid when combined with the current CMDVOL setting, an error occurs. An error message is displayed and the CMDSYS setting is not changed.

The following are examples of the CMDSYS command:

```
CMDSYS \ny
```

```
CMDSYS \sd
```

CMDVOL Command

The CMDVOL command sets the default volume and subvolume for expansion of any file names except obey file names.

```
CMDVOL { $volume              }
       { [ $volume. ] subvol }
```

volume

is a *Guardian* operating environment volume name.

subvol

is a Guardian subvolume name.

If the CMDVOL command is not issued, the default settings in effect when SCUP was started are used. If either the volume name or subvolume name is omitted, the previous setting applies.

If either part of the new specification is invalid, or would be invalid when combined with the current defaults and the current CMDSYS setting, an error occurs. An error message is displayed and the CMDVOL defaults are not changed.

The following are examples of the CMDVOL command:

```
CMDVOL $mkt.abc
```

```
CMDVOL $engr.def
```

COMPRESS Command

The COMPRESS command decreases the size of SCREEN COBOL object files. SCUP rewrites the programs contiguous to each other in the code file and rewrites the associated symbol tables contiguous to each other in the symbol table file, thus eliminating any wasted space. This command can be used to reclaim the space left when programs are deleted.

```
COMPRESS [ file-name ]
```

file-name

is a SCREEN COBOL object file name. If the file-name option is omitted, the file specified in the FILE command is used.

The COMPRESS command renumbers the version numbers of the existing program in the object file without reordering the version priority; the earliest version is always 1. For example, assume POBJ contains four versions of a program and version 3 is deleted, leaving versions 1, 2, and 4. After the COMPRESS command executes, version 4 becomes known as version 3. The compressed object file name is displayed when the COMPRESS command terminates.

SCUP must have exclusive access to the SCREEN COBOL object file to perform a COMPRESS command. A TCP that uses the object file to be compressed must be stopped before the COMPRESS command is issued.

Note. SCUP will compress files under Safeguard protection as long as the PERSISTENT attribute is set by means of the Safeguard ADD command for the POBJDIR, POBJCOD, and POBJSYM commands.

The following example compresses a SCREEN COBOL object file called SCOBJ:

```
COMPRESS scobj

$MKT.SUBVOL.SCOBJ
COMPRESS $MKT.SUBVOL.SCOBJ
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  TEST-PROG1 (2)                        TEST-PROG1 (1)
  TEST-PROG1 (3)                        TEST-PROG1 (2)
  TEST-PROG2 (3)                        TEST-PROG2 (1)
COMPRESSED $MKT.SUBVOL.SCOBJ
```

CONVERT Command

The CONVERT command converts a group of related SCREEN COBOL program units within a SCREEN COBOL object code file to a web client consisting of Java classes and HTML pages. .

DISCLAIMER NOTICE. While this SCUP utility is a tool, you must have the right to modify third-party files. In no way does the licensing or use of our tool grant you rights, which you may not have vis-a-vis a third party.

```
CONVERT [ SCOBOL-OBJECT ] [ file-name ]
  [ , pobj-spec                       ]
  , INITPROG program-id
  [ , CONFIGFILE config-file            ]
  [ , APPNAME app-name                  ]
  [ , JAVAVOL [\system.]$volume.subvol  ]
  [ , HTMLVOL [\system.]$volume.subvol  ]
  [ , CHARSET SJIS                      ]

pobj-spec is:
  prog-name [ , prog-name ]...

  prog-name is:
    ( { program-id } [ ( version ) ] )
      { *          }
```

file-name

is the first four characters of the SCREEN COBOL object-file name.

If this parameter is omitted, the currently opened file is used. (The currently opened file is the last file specified in the FILE command.) If this parameter is omitted and a file has not previously been opened through the FILE command, the converter searches for a file named POBJCOD in the current volume and subvolume. If such a file is not found, the converter generates an error.

pobj-spec

identifies specific programs to be converted within the object file specified in *file-name*. It consists of a list of one or more *prog-name*s, separated by commas. Items within the *prog-name* syntax are defined as follows:

program-id

is a SCREEN COBOL program name specified in a PROGRAM-ID paragraph.

*

specifies all program IDs (SCREEN COBOL program names) in the object file.

version

is the version of the specified program ID or of all program IDs. If present, the value must be an integer identifying a single version number; the * and + options for *version* are not valid in the CONVERT command. If *version* is omitted, the latest version is used.

If *pobj-spec* is omitted, the converter converts the latest version of all the program IDs contained in the object file specified by *file-name*.

`INITPROG` *`program-id`*

identifies the program ID of the first program unit to be executed. This is similar to the INITIAL attribute of the TERM object in a Pathway configuration. The corresponding Java code will be the first code to be executed in the web client. This parameter is required; if it is omitted, the converter generates an error.

The converter uses the program-id value to trace the program logic to identify all the program units. For each program unit called by the program logic, the converter checks whether that program unit is available in the object file. The converter generates a warning message either if a program unit is not available in the currently opened file or if it cannot resolve the program unit reference (because of an indirect CALL using a Working-Storage variable).

`CONFIGFILE` *`config-file`*

identifies a user-supplied configuration file that provides values that are used during the conversion. The *`config-file`* must be a Guardian file with file code 101 and must have the format described in Configuration File Format. If this parameter is not specified, the converter uses default values.

`APPNAME` *`app-name`*

identifies a user-specified application name. *`app-name`* has a maximum of 30 characters and must not be the same as any other program being converted.

The converter uses this value to generate the names for the Java applet, the starting HTML page, and the control HTML page. If this parameter is omitted, the converter uses the *`program-id`* value specified in the INITPROG parameter to generate applet names and HTML file names. The syntax for *`app-name`* follows the same convention as the syntax for *`program-id`*.

`JAVAVOL [\`*`system`*`.]$`*`volume`*`.`*`subvol`*

identifies the location where the converter is to create the Java source files. You must specify a fully qualified Guardian volume and subvolume name with optional system name. If this parameter is omitted, the converter uses the system, volume, and subvolume specified or implied in the CONFIGFILE parameter. If the CONFIGFILE parameter is not specified, the Java files are created in the current system, volume, and subvolume from which the CONVERT command was executed.

`HTMLVOL [\`*`system`*`.]$`*`volume`*`.`*`subvol`*

identifies the location where the converter is to create the HTML output files. You must specify a fully qualified Guardian volume and subvolume name with optional system name. If this parameter is omitted, the converter uses the system, volume, and subvolume specified or implied in the CONFIGFILE parameter. If the CONFIGFILE parameter is not specified, the HTML files are created in the current system, volume, and subvolume from which the CONVERT command was executed.

`CHARSET SJIS`

specifies that the SCOBOL program being converted contains Japanese kanji-katakana characters.

Configuration File Format

The configuration file specified in the CONFIGFILE parameter provides default values that are used by the converter. The file format is similar to that of an .INI file. It has multiple sections, each consisting of a section heading followed by a list of name-value pairs, as follows:

```
"["section-name"]"
[ name = value ; ]...

"["section-name"]"
[ name = value ; ]...

"["section-name"]"
[ name = value ; ]...

"["section-name"]"
[ name = value ; ]...
```

`"["`*`section-name`*`"]"`

is the name of a section of the configuration file, enclosed in square brackets.

Note. The quotation marks around these square brackets show that the brackets are required characters rather than part of the syntax notation.

The section names to be used are as follows:

```
[Screen Section]
[Convert Environment]
[Compile Environment]
[Deployment Environment]
[Limits Section]
```

The names are not case-sensitive; that is, letters in the section names can be any combination of uppercase and lowercase.

`name`

is the name of a valid parameter within the specified section of the configuration file.

`value`

is a valid value for the specified parameter.

When the CONVERT command is issued, as part of initialization (after syntax checking is done), the converter reads each section of the configuration file and populates a corresponding array with the name and value for each parameter. If a parameter is not specified, the default value is used.

Screen Section Parameters

The Screen section contains parameters related to the presentation (display appearance) of the converted web client. The following table describes each parameter, its valid values, and its default value. The default value is used if no configuration file is specified or if that parameter is not specified in the configuration file.

Parameter Name	**Valid Values**	**Default Value**	**Description**
Background	Valid HTML color or valid GIF file name	White	Color or GIF artwork to be used for the background. GIF file name may not exceed 40 characters.
Foreground	Valid HTML color	Black	Color to be used for the foreground
Font Type	Arial Arial Black Arial Bold Book Antiqua Century New Century Schoolbook Courier Courier New Fixedsys Helvetica Lucida Console Tahoma Terminal Times New Roman Verdana	Courier	Font type to be used
Font Size	12 14 16	12	Font size to be used
End Page	end.html	end.html	File that is displayed when the user quits the application; the Java Library automatically looks for a file of this name to display when an application terminates successfully. end.html can be modified as desired to display any content, including images.

For proper left alignment of screen fields, the CONVERT command uses absolute coordinates to generate the fields at fixed positions on the screen. Therefore, if the font type or size is changed in the generated HTML page, the left alignment remains unchanged but some fields may overlap. To avoid overlapping fields, change the font type or size in the configuration file and convert the application again, then deploy the new HTML pages and discard the Java files.

Note. When you specify a GIF file name in the Background parameter, specify only the file name; do not include the path. Ensure that at deployment time, the specified GIF files are available at the location where the converted HTML files are placed.

Convert Environment Section Parameters

The Convert Environment section contains parameters related to the converter environment. The following table describes each parameter, its valid values and its default value. The default value is used if no configuration file is specified or if that parameter is not specified in the configuration file.

Parameter Name	Valid Values	Default Value	Description
Java Location	Valid NonStop Kernel location of the form [\system.]vol.subvol	Current location where SCUP is running	Location on the NonStop Kernel system where the converter is to place the converted Java source files.
HTML Location	Valid NonStop Kernel location of the form [\system.]vol.subvol	Current location where SCUP is running	Location on the NonStop Kernel system where the converter is to place the converted HTML files.

Compile Environment Section Parameters

Note. The Compile Environment section is not currently used by the converter or by JMAKENV. This section is defined for future use.

The Compile Environment section contains parameters related to the environment where the converted Java source files will be compiled. The following table describes each parameter, its valid values, and its default value. The default value is used if no configuration file is specified or if that parameter is not specified in the configuration file.

Parameter Name	Valid Values	Default Value	Description
OS	NSK NT	NSK	Operating system of the compile environment.

Deployment Environment Section Parameters

Note. The Deployment Environment section is not currently used by the converter or by JMAKENV. This section is defined for future use.

The Deployment Environment section contains parameters related to the environment where the converted application will be deployed. The following table describes each parameter, its valid values, and its default value. The default value is used if no configuration file is specified or if that parameter is not specified in the configuration file

Parameter Name	Valid Values	Default Value	Description
Web root	Valid path name in OSS format	/usr/tandem/webserver /root/its	Path name for the root directory of the web server where the application is to be deployed.

Limits Section Parameters

The Limits section contains parameters for configuring maximum values for some of the OPCODES statements used when converting a program unit. The following table describes each parameter, its valid range of values, and its default value. The default value is used if no configuration file is specified or if that parameter is not specified in the configuration file.

Parameter Name	Valid Values	Default Value	Description
SYMBOLS	0 - 32767	5000	Maximum number of symbol information stored per program unit.
CALLED PROGRAM UNITS	0 - 32767	500	Maximum number of PROGRAM-IDs called from a program unit.
CONDITIONAL OPCODES	0 - 32767	5000	Maximum number of conditional clauses (IF) in program unit.
DATA ITEMS	0 - 32767	5000	Maximum number of data items per program unit.
DATA SIZE	0 - 100000	80000	Maximum size for data items within data items in a program unit.
EXIT PERFORM OPCODES	0 - 32767	500	Maximum number of EXIT PERFORM statements in a program unit.
INDEX RESOLVING OPCODES	0 - 32767	1000	Maximum number of NEW^INDEX^local statements in a program unit.
GOTO OPCODES	0 - 32767	5000	Maximum number of GOTO statements in a program unit.
GOTO DEPENDING OPCODES	0 - 32767	300	Maximum number of GOTO DEPENDING statements in a program unit.

Parameter Name	Valid Values	Default Value	Description
ON ERROR CLAUSES	0 - 32767	500	Maximum number of ON-ERROR clause count in a program unit.
OVERLAY AREAS	0 - 32767	1000	Maximum number of overlay areas per program unit.
PARAGRAPHS	0 - 32767	1000	Maximum number of paragraphs per program unit.
PERFORM DEPENDING OPCODES	0 - 32767	300	Maximum number of PERFORM DEPENDING statements in a program unit.
REDEFINES	0 - 32767	1000	Maximum number of REDEFINES and RENAMES in a program unit.
SCREENS	0 - 32767	100	Maximum number of base and overlay screens per program unit.
SCREEN FIELDS	0 - 32767	2500	Maximum number of screen field descriptors per program unit.
SEND OPCODES	0 - 32767	500	Maximum number of SEND statements in a program unit.
TRANSFORM STATEMENTS	0 - 32767	100	Maximum number of TRANSFORM statements in a program unit.

Example Configuration File

The following is an example of a configuration file that changes some of the default parameters used by the converter. The converter uses default values for the omitted parameters: the display font size will be 12 and the compile environment operating system will be the NonStop Kernel.

```
[Screen Section]
Background = background.gif;
Foreground = Black;
Font Type = Times Roman;
Font Size = 14;

[Convert Environment]
Java Location = \system.$java.appa;
HTML Location = \system.$html.appa;

[Compile Environment]

[Deployment Environment]
Web Root = /usr/webserver/root;

[Limits Section]
SYMBOLS = 8000;
CONDITIONAL OPCODES = 5;
```

Summary Information

After converting individual program units, the converter generates a brief summary about the conversion that includes the following:

- Unsupported Statements

  This is the number of unsupported statements that the converter detected during the conversion and includes the following types:

  - Warnings: The number of warning messages detected by the converter. The converter issues a warning when it cannot convert a statement but it determines that excluding the statement from the conversion will not affect program execution.
  - Errors: The number of Error messages generated by the converter. The converter issues an error when it can not convert a statement but it determines that excluding the statement from the conversion will affect program execution.

- Internal Errors

  This is the number of internal errors that the converter detected during the conversion. Internal errors indicate logical errors in the conversion and are not recoverable for user.

- HTML Files Generated

  This is the number of HTML files generated for a Program unit. Each Base / Overlay Screen gets converted to one HTML page.

- Lines of Code in Java

  This is the number of lines of code generated in a Java file.

Example Summary

The following is a typical example of the summary at the end of Program unit conversion:

```
PROGRAM UNIT MYPROG CONVERTED TO $test.myapp.POBJ0000
Unsupported Statements :      3
    Warnings :     1    Errors :      2
Internal Errors        :      0
HTML Files Generated   :      2
Lines of Code in Java  :    570
```

The converter also generates a total conversion summary at the completion of the CONVERT command. This summary provides information about unsupported warnings and errors and the internal errors detected during the conversion of all the program units specified in the CONVERT command.

The following is a typical example of the Total summary:

```
Total Summary for CONVERT command
Unsupported Statements :     4
    Warnings :    1     Errors :     3
Internal Errors        :     0
```

Operational Details

General

The CONVERT command converts the Working-Storage Section and Procedure Division components of the SCREEN COBOL program units to Java classes and includes the necessary classes from the Pathway/iTS Java import library. The CONVERT command converts the Screen Section and other screen components to HTML pages, with embedded JavaScript code to link the HTML fields to their corresponding Java objects.

For each SCREEN COBOL application (set of program units within an object file), the converter generates the following Java classes and HTML files:

- Main applet
  Generated as `*initprog*_MAINAPP.java or *appname*.java`
- Starting page
  Generated as `*initprog*_START.html or *appname*_START.html`
- Initial page
  Generated as `*initprog*_PAGE1.html or *appname*_PAGE1.html`
- Control page
  Generated as `*initprog*_CONTROL.html or *appname*_CONTROL.html`
- Java classes for each program unit
- User HTML pages for the various screen sections

The CONVERT command converts the specified SCREEN COBOL program units in such a way that most of the functionality of the original SCREEN COBOL program units remains unchanged. For exceptions, and for a detailed description of the Java and HTML output from the converter, refer to the *Pathway/iTS Web Client Programming Manual*.

SCUP also creates a MAP file <*SCOBOL-OBJECT*>MAP, which contains Guardian to OSS mapping information for all Java and HTML files of the application. The CONVERT command uses a MAP file when re-converting a program unit. The JMAKENV TACL MACRO utility also uses the MAP file to deploy the converted Java and HTML files to the OSS environment. For more information about the JMAKENV TACL MACRO utility, see the *Pathway/iTS Web Client Programming Manual*.

Restrictions

The following restrictions apply to the conversion process:

- The converter works with block-mode non-IDS SCREEN COBOL program units only.
- Symbol information for the program units must be available in the POBJSYM file, which must reside in the same volume and subvolume as the SCREEN COBOL object file.
- The CONVERT command supports up to 15 levels in a group data declaration, whereas the Screen COBOL language supports up to 48 levels. This restriction applies to the actual number of different levels, not to the level numbers themselves. For example, in the following data declaration there are only four levels even though the level numbers are 01 through 20:

```
01 WS-RECORD.
   05 WS-PERSONAL-DETAILS
      10 WS-NAME
      10 WS-ADDRESS.
         20 WS-ADDRESS-LINE-1
         20 WS-ADDRESS-LINE-2
          20 WS-ADDRESS-LINE-3
```

  To prevent conversion problems, look at each data structure and remove unused group level declarations. For example, if there was no reference in the code to the group element WS-PERSONAL-DETAILS in the above structure, you can remove it to rearrange the data structure as follows, which reduces the number of levels from four to three:

```
01 WS-RECORD.
   10 WS-NAME
   10 WS-ADDRESS.
      20 WS-ADDRESS-LINE-1
      20 WS-ADDRESS-LINE-2
       20 WS-ADDRESS-LINE-3
```

Troubleshooting

- Unconditional GO TO statements

  Cause: In some cases, when there are SCOBOL statements directly after an unconditional GO TO statement, the Java compiler generates the error Statement is unreachable.

  Recovery:

  1. Remove or comment out the appropriate SCOBOL statements.
  2. Recompile the SCOBOL program, execute the CONVERT command again, and recompile the Java code.

  or

  1. Comment out the statements of the method in the generated Java file after the return statement, then recompile the Java code.

- No such method error

  Cause: Occasionally, the converter cannot distinguish between the section name and the first paragraph name within the section, and generates the error No such method.

  Recovery: Manually check the generated code. If a method exists with the same name as the section name or the first paragraph name, replace the method call with the name of the section or first paragraph.

- InvocationTargetException

  Cause: Occasionally a converted application throws a InvocationTargetException to the caller of the program unit. This can occur because SCOBOL handles the ON ERROR clause of BEGIN-TRANSACTION differently from the generated Java code and may occur if the same application is run in multiple browsers on the same Windows desktop.

  After the execution of the ON ERROR clause in SCOBOL, execution continues to the statement next to the BEGIN-TRANSACTION statement. However, in the generated Java code, control returns to the caller of the program unit.

  BEGIN-TRANSACTION, SEND, and END-TRANSACTION statements are host-delegated tasks that are performed by the gateway (PathTCP3 and the SCOBOL GATEWAY) on the NonStop system. If the link between the client and the gateway is abnormally terminated (for example, due to the loss of the socket link with the client or due to the abnormal termination of TMF), the applet executing the Java code executes the ON ERROR clause much like SCOBOL would execute it. In other words, if the link is terminated when the client is in transaction mode (between BEGIN-TRANSACTION and END-TRANSACTION statements), then the ON ERROR clause is executed by the client. If the ON ERROR clause is not present, the applet throws error 3301, Transaction Error, and the application stops.

  After executing the ON ERROR clause, an exception (ScobolTxnException) is thrown to the caller of the program unit, which catches it as an InvocationTargetException. The control, therefore, returns to where the program unit was invoked.

  Recovery:

  - From a single Windows desktop, run the application in a single browser. To restart the application, close the browser and then restart it.
  - Or, modify the catch block of the CALL statement in the generated Java code to catch the exception ScobolTxnException. Add code in the catch block to continue the execution in a more appropriate manner.

- Purge error for file *`old_system_name.vol.subvol.htmlfile`* (file-system error)

Cause. SCUP returns this error when you attempt to reconvert a program unit of the existing SCOBOL object after transferring the Java, HTML, and MAP files from the original location to another location. SCUP purges the original files and then re-creates them to reflect the changes. If SCUP is not able to locate any of the original files mentioned in the MAP file for purging, it returns this error.

Recovery. Complete the following steps:

1. Delete the MAP file and all files in the HTMLVOL and JAVAVOL volumes that correspond to the application.
2. Restart the conversion.

Or

1. FUP COPY all the content of the MAP file to an editable file.
2. Manually update the locations listed in the editable file to reflect the target system location.
   For example, to move the `MAINAPP` file from `\SYS1` to `\SYS2`, change the corresponding entry in the editable file as follows:
   Change `\SYS1.$FC4.USERAUTH.MAINAPP` to `\SYS2.$FC9.SCUPVOL.MAINAPP`.
3. FUP PURGEDATA the MAP file.
4. FUP COPY the content of the editable file to the MAP file.

Command Examples

- The following example directs the converter to convert the latest version of all program units in the object file `abcd`. The starting HTML page is called `MYPROG_START.html`, and it will use a background color of white and a foreground color of black. The font used will be 12-point Courier. The compile environment for the converted code will be the NonStop Kernel. The converted Java source and HTML files are created in the current SCUP system, volume and subvolume.

```
FILE abcd
CONVERT, INITPROG myprog
```

- The following example directs the converter to convert the latest version of all program units in the object file `defg`. The starting HTML page is called `MYAPP_START.html`. The output attributes are obtained from the configuration file `mycfg`.

```
CONVERT defg, INITPROG myprog, CONFIGFILE mycfg, APPNAME
myapp
```

- The following example directs the converter to convert version 1 of all program units in the object file `defg`. The starting HTML page is called `MYAPP_START.html`. The output attributes are obtained from the configuration file `mycfg`. The converted Java files are created in `$abc.java`, and the HTML files are stored in `$abc.html`, both on the current system.

```
CONVERT defg (*(1)), INITPROG myprog, CONFIGFILE mycfg, &
APPNAME myapp, JAVAVOL $abc.java, HTMLVOL $abc.html
```

COPY Command

The COPY command copies a set of programs from one SCREEN COBOL object file to another. If a program named in the COPY command has an associated entry in the symbol table file, the symbol table is copied to the destination symbol table file.

```
COPY [ SCOBOL-OBJECT ] [ source-file-name ]

  { ( program-name [ ( version ) ]                }
  {   [ , program-name [ ( version ) ] ] ... ) }
  {                                               }
  { ( * [ ( version ) ] )                         }

  [ , destination-file-name ]
```

source-file-name

is a SCREEN COBOL object file name and is the source file for the COPY command. This file name must be different from destination-file-name.

If this parameter is omitted, the file specified in the FILE command is used.

(*program-name*)

is a SCREEN COBOL program name. The program name can be a single program name or several program names separated by commas.

(*)

is all program names in the object file.

(*version*)

is the version of a specified program name. *Version* can be any of the following:

integer	a single version of a program
*	all versions of a program
+	all but the latest version of a program

If the version parameter is omitted, the latest version of a program is used.

destination-file-name

is a SCREEN COBOL object file name and is the destination file for the COPY command. This file name must be different from *source-file-name*.

If this parameter is omitted, the file specified in the FILE command is used.

The COPY command copies programs from the source object file to the destination object file in the order specified. When a program is copied to the destination file, the program is given a new version number that designates it the latest version of the program name in the object file. The old and new program name and version numbers are displayed after the copy operation is complete.

If two or more versions of the same program are specified in the COPY command, the last program specified becomes the latest version in the destination object file. This gives you control over the priority assigned to program versions when copying from one file to another.

If the * or + form of version number is specified, the programs receive new version numbers, but retain the same version priority in the destination object file. In effect, the latest version in the source file is copied last and remains the latest version in the destination file for this form of version specification.

If the destination object file specified in the COPY command does not exist, SCUP creates the file before executing the COPY command. If the destination file does exist, SCUP adds the named program to the programs already present in the specified object file (in the same manner as the SCREEN COBOL compiler).

When executing a COPY command, SCUP allows shared access to the SCREEN COBOL object files. The COPY command does not disrupt concurrent users of the object files, even if the name of the program being added is the same as one already present. This allows additions to be made to object files that are in use by the TCP.

The object file-name expansion uses the default settings specified by the CMDSYS and CMDVOL commands.

The following example copies two versions of a program called TEST, retaining the version priority of the programs:

```
COPY file1(test(3),test(5)), file2

$MKT.OLDSV.FILE1      COPY TO         $MKT.OLDSV.FILE2
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  TEST (3)            TEST (1)
  TEST (5)            TEST (2)
```

The following example copies two versions of program TEST, reversing the version priority of the programs:

```
COPY file1(test(5),test(3)), file2

$MKT.OLDSV.FILE1      COPY TO         $MKT.OLDSV.FILE2
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  TEST (5)            TEST (1)
  TEST (3)            TEST (2)
```

The following example copies all versions of program TEST, retaining the version priority of the programs:

```
COPY file1(test(*)), file2:

$MKT.OLDSV.FILE1    COPY TO      $MKT.OLDSV.FILE2
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  TEST (3)          TEST (1)
  TEST (4)          TEST (2)
  TEST (5)          TEST (3)
```

The following example copies the latest versions of program TEST into FILE2.

```
COPY file1(*), file2

$MKT.OLDSV.FILE1    COPY TO      $MKT.OLDSV.FILE2
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  TEST1 (2)          TEST1 (1)
  TEST3 (1)          TEST3 (1)
  TEST7 (2)          TEST7 (1)
```

DELETE Command

The DELETE command removes programs from the SCREEN COBOL object file directory; the code and symbol table file space is not reclaimed. Issue the COMPRESS command after the delete operation to reclaim this space.

```
DELETE [ SCOBOL-OBJECT ] [ file-name ]

  { ( program-name [ ( version ) ]                  }
  {      [ , program-name [ ( version ) ] ] ... ) }
  {                                                  }
  { ( * [ ( version ) ] )                            }
```

file-name

is a SCREEN COBOL object file name.

If this parameter is omitted, the file specified in the FILE command is used.

(*program-name*)

is a SCREEN COBOL program name. The program name can be a single program name or several program names separated by commas.

(*)

is all program names in the object file.

(*version*)

is the version of a specified program name. *Version* can be any of the following:

integer	a single version of a program
*	all versions of a program
+	all but the latest version of a program

If the version parameter is omitted, the latest version of a program is used.

The DELETE command does not reorder or change the version numbers of the remaining programs in the object file. If the latest version of a program is deleted, the previous version becomes the latest version. The deleted program name and version number are displayed after the DELETE operation is performed.

When executing a DELETE command, SCUP allows shared access to the SCREEN COBOL object file. From the moment the DELETE command completes, any attempt to find the specified programs (either through the CALL verb in a SCREEN COBOL program or through the INITIAL setting when starting a terminal) will fail. Any terminal already executing that program continues unaffected. Thus, the program can still be in use, but no new references to it can be made.

The object file-name expansion uses the default settings specified by the CMDSYS and CMDVOL commands.

The following example deletes the first two versions of a program called TEST:

```
DELETE $mkt.subvol.file1(test(1),test(2))
```

```
$MKT.SUBVOL.FILE1
DELETED  TEST (1)
DELETED  TEST (2)
```

The following example deletes all but the latest version of all programs in POBJ:

```
DELETE (*(+))
```

The following example deletes all versions of all programs in POBJ:

```
DELETE (*(*))
```

EXIT Command

The EXIT command stops SCUP.

```
EXIT
```

FC Command

The FC command provides the ability to edit or to repeat a command line.

```
FC
```

When this command executes, it displays the previous command line up to 132 characters and prompts for editing input with a period (.). FC accepts three subcommands:

R	(*replacement-string*)	replace one or more characters
I	(*insertion-string*)	insert one or more characters
D		delete one character

Subcommands and their associated strings are entered beneath the displayed command line and terminated with a carriage return. Replacement, insertion, and deletion begins with the character positioned directly above the subcommand (R, I, D).

Subcommand R replaces characters in the command line with replacement-string on a one-for-one basis. Subcommand I inserts characters in the command line with insertion-string on a one-for-one basis. Subcommand D deletes the character above the letter D in the command line; the subcommand can be repeated for each character that is to be deleted. If a string but no subcommand is entered, the default subcommand, R, is used.

After the line is edited, FC again displays the command line and prompts for another subcommand. FC terminates when it receives only a carriage return; the corrected command line is then executed.

The FC command can be terminated without execution by pressing BREAK; pressing CTRL/Y; or entering a double slash (//) in columns 1 and 2, immediately followed by a carriage return.

The following are examples of FC subcommand positioning:

```
>the subbsysten
>      d            <--- deletes the extra letter b
>  iscup            <--- inserts the word scup
>           rm      <--- replaces the letter n with m
```

FC subcommands can be separated by the double slash.

FILE Command

The FILE command specifies the default SCREEN COBOL object file name to be used in SCUP commands. The commands that allow an optional object file name (ALTER, COMPRESS, CONVERT, COPY, DELETE, and INFO) use the file name specified in this command as the default name.

```
FILE file-name
```

file-name

is a SCREEN COBOL object file name.

The file-name expansion uses the default settings specified by the CMDSYS and CMDVOL commands. The specification of a default object file remains in effect until the program terminates or another FILE command is executed. The new file name is displayed after the command is executed.

The expansion of the object file name follows the file-name expansion specifications. The initial setting for the default object file name uses the default settings from the command interpreter and the default name POBJ:

\system.$*volume*.*subvol*.POBJ

The SCREEN COBOL compiler names the object files POBJCOD, POBJDIR, and POBJSYM (if the SYMBOLS compiler command was specified) within the default system, volume, and subvolume.

HELP Command

The HELP command displays the syntax of SCUP commands.

```
HELP [ command-name   ]
     [ < param-name > ]
```

command-name

is the name of a SCUP command for which the syntax is displayed.

< *param-name* >

is any parameter value that is enclosed in angle brackets when displayed by the command-name option. The angle brackets must be included.

If parameters are omitted, the names of all SCUP commands are displayed.

The following are examples of the HELP command:

```
HELP
HELP copy
HELP <program-name>
```

INFO Command

The INFO command displays information about program units in SCREEN COBOL object files or information about the object files. Program information includes program names and versions, access flag settings, the program size (amount of disk space), the version of the SCREEN COBOL compiler, and the compilation date. Information about the object file includes extents and the library access setting, which applies to new programs added.

```
INFO [ SCOBOL-OBJECT ] [ / OUT list-file / ][ file-name ]
  [ ( program-name [ ( version ) ]                          ]
  [   [ , program-name [ ( version ) ] ] ... ) ] [, DETAIL ]
  [                                                         ]
  [ ( * [ ( version ) ] )                                   ]
```

list-file

is a file name to which the listing is directed. The default setting for the OUT file name is that from the command interpreter or the previous OUT command. The redirection of the listing output is temporary and only applies to this occurrence of this command.

file-name

is the SCREEN COBOL object file name that you specified in the SCREEN COBOL run command, not the output file name assigned by the system. For example, POBJ is valid for *file-name*, not POBJCOD, POBJDIR, or POBJSYM.

If this parameter is omitted, the file specified in the FILE command is used.

(*program-name*)

is a SCREEN COBOL program name. The program name can be a single program name or several program names separated by commas.

(*)

is all program names in the object file.

If *program-name* or * (for a program name) is omitted, no information about programs in the object file is displayed.

(*version*)

is the version of a specified program name. *Version* can be any of the following:

integer	a single version of a program
*	all versions of a program
+	all but the latest version of a program

If the version parameter is omitted, the latest version of a program is used.

DETAIL

displays information about the program such as the code size, data size, number of bytes used for screen storage, screen count, and descriptor count. The explicit references to external SCREEN COBOL program names also are displayed.

Note. DESCRIPTOR COUNT gives the number of data descriptors used for this program unit. One data descriptor is used for each data item. Predefined data items such as OLD-CURSOR also have data descriptors; those descriptors are included in the DESCRIPTOR COUNT. SCREEN COUNT gives the number of screens defined for this program unit.

When executing an INFO command, SCUP allows shared access to the SCREEN COBOL object file. This allows information to be displayed for object files that are in use by the TCP.

The object file-name expansion uses the default settings specified by the CMDSYS and CMDVOL commands.

The following example displays information about the object file. The information includes the access setting for programs added to the file and object file extents.

```
INFO $data.subvol.file1

$DATA.SUBVOL.FILE1
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
ACCESS ON
PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
```

The following example displays information about all versions of a program called TEST-PROG.

```
INFO $data.subvol.file1 (test-prog:(*))

$DATA.SUBVOL.FILE1
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
ACCESS ON
PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
  TEST-PROG (1)     OFF   4096  (C00) 03 JUL 1987  12:39:32
  TEST-PROG (2)     ON    4096  (C00) 07 JUL 1987  08:17:07
  TEST-PROG (3)     ON    4096  (C00) 25 JUL 1987  17:28:14
```

The following example displays detailed information about all but the latest versions of the program TEST-PROG:

```
INFO $data.subvol.file1 (test-prog(+)), DETAIL

$DATA.SUBVOL.FILE1
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
ACCESS ON
PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
  TEST-PROG (1)       ON    6144  (C00) 03 JUL 1987  09:33:32
    TERMINAL TYPE = T16-6530
    CHARACTER SET = UK
    CODE SIZE = 233
    DATA SIZE = 94
    SCREEN COUNT = 1
    SCREEN SIZE TOTAL = 2110
    DESCRIPTOR COUNT = 6
    SYMBOL TABLE

PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
  TEST-PROG (2)       OFF   6144  (C00) 25 JUL 1987  14:57:38
    TERMINAL TYPE = T16-6520
    CODE SIZE = 281
    DATA SIZE = 107
    SCREEN COUNT = 1
    SCREEN SIZE TOTAL = 2110
    DESCRIPTOR COUNT = 7
    PROGRAM NAMES REFERENCED
      TEST-PROG
      SUB-PROG
```

The following example displays detailed information about the latest version of the program called TEST-PROG:

```
?INFO $data.subvol.file1 (test-prog), DETAIL

$DATA.SUBVOL.FILE1
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
ACCESS ON
PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
  TEST-PROG (1)       ON    6144  (C00) 27 JUL 1987  11:23:02
    TERMINAL TYPE = T16-6530
    CHARACTER SET = UK
    CODE SIZE = 233
    DATA SIZE = 94
    SCREEN COUNT = 1
    SCREEN SIZE TOTAL = 2110
    DESCRIPTOR COUNT = 6
    SYMBOL TABLE
    PROGRAM NAMES REFERENCED
      FUNC-PROG
```

The SYMBOL TABLE line indicates a symbol table exists for version 1 of TEST-PROG.

OBEY Command

The OBEY command causes commands to be read from a specified file.

```
OBEY file-name
```

file-name

is a file name.

If the file name is not fully qualified, the name is expanded with the defaults supplied by the OBEYSYS and OBEYVOL commands or by the SYSTEM and VOLUME commands.

Commands are read from the named file and processed until an end-of-file is encountered. When the end-of-file indicator is reached, the obey file is closed and command input reverts to the file from which the OBEY command was read. Additional OBEY commands can appear within an obey file. Obey files can be nested to a depth of four; if the limit is exceeded, an error occurs and an error message is displayed.

If default settings such as VOLUME or OBEYVOL are changed in an obey file, these settings are not automatically returned to their previous states.

If any part of the specification is invalid, if the file does not exist, or if the file cannot be opened, an error occurs. An error message is displayed and the current source for command input is not changed.

If an error is detected during processing of commands from an obey file, that obey file and any other obey files currently open (in the case of nested obey files) are closed. If the original input file for SCUP was a terminal, the terminal is issued a prompt. If the input file was not a terminal, SCUP terminates.

The following are examples of the OBEY command:

```
OBEY abc.scupcmd
```

```
OBEY scupcmd
```

OBEYSYS Command

The OBEYSYS command sets the default system for expansion of obey file names.

```
OBEYSYS [ \system ]
```

system

is a Guardian node name.

If the OBEYSYS command is not issued, the default settings in effect when SCUP was started are used. If the node name is omitted, the default is set for local file-name expansion. Note that this is not the same as specifying the local system name. Omitting the node name in this command permits references to long device names; long device names are prohibited in network-form file names.

If the specified node name is invalid, or would be invalid when combined with the current OBEYVOL setting, an error occurs. An error message is displayed and the OBEYSYS setting is not changed.

The following are examples of the OBEYSYS command:

```
OBEYSYS \ny
```

```
OBEYSYS \sd
```

OBEYVOL Command

The OBEYVOL command sets the default volume and subvolume for expansion of obey file names.

```
OBEYVOL [ $volume              ]
        [ [ $volume. ] subvol ]
```

volume

is a Guardian volume name.

subvol

is a Guardian subvolume name.

If the OBEYVOL command is not issued, the default settings in effect when SCUP was started are used. If either the volume name or subvolume name is omitted, the previous setting applies. If both names are omitted and the file from which the OBEYVOL command was read is a disk file, the defaults for obey file-name expansion are set to the system, volume, and subvolume of the current command input file. If both names are omitted and the file from which the OBEYVOL command was read is not a disk file, an error occurs; an error message is displayed and the OBEYVOL defaults are not changed.

If either part of the new specification is invalid, or would be invalid when combined with the current defaults and the current OBEYSYS setting, an error occurs. An error message is displayed and the OBEYVOL defaults are not changed.

The following are examples of the OBEYVOL command:

```
OBEYVOL $mkt
```

```
OBEYVOL $engr.def
```

OUT Command

The OUT command directs the output listing for SCUP commands to a specified file.

```
OUT file-name
```

file-name

is a file name.

The initial setting for *file-name* is the current output file of the command interpreter; this is typically the terminal. The setting remains in effect until SCUP terminates or another OUT command is executed.

If the file name has the form of a disk file and the file does not exist, an EDIT file is created. If the named file is an existing disk file, the output is appended to the file.

If the file name is invalid or if the file cannot be opened, an error occurs and an error message is displayed.

SYSTEM Command

The SYSTEM command sets the default system for expansion of any file names.

```
SYSTEM [ \system ]
```

system

is a Guardian node name.

A SYSTEM command is equivalent to entering both an OBEYSYS and a CMDSYS command with the same specification.

The following are examples of the SYSTEM command:

```
SYSTEM \ny
```

```
SYSTEM \sd
```

VOLUME Command

The VOLUME command sets the default volume and subvolume for expansion of any file names.

```
VOLUME { $volume              }
       { [ $volume. ] subvol }
```

volume

is a Guardian volume name.

subvol

is a Guardian subvolume name.

A VOLUME command is equivalent to entering both an OBEYVOL and a CMDVOL command with the same specification. Note that this does not necessarily mean that the settings for OBEYVOL and CMDVOL would be identical following a VOLUME command. For example, assume the default for OBEYVOL is $V1.A and the default for CMDVOL is $V2.B. A subsequent command of VOLUME X yields $V1.X for OBEYVOL, and $V2.X for CMDVOL.

The following are examples of the VOLUME command:

```
VOLUME $mkt
```

```
VOLUME $mkt.abc
```

3 Using SCUP

This section illustrates the use of the SCUP utility to control and manipulate SCREEN COBOL object files. The examples show building a new SCREEN COBOL object file ($DATA.NEWSV.POBJ) from selected program units in two existing SCREEN COBOL object files ($DATA.OLDSV.POBJ and $MKT.OLDSV.SCOB).

Note. This section does not describe use of the CONVERT command. For further information about the CONVERT command, refer to the examples under CONVERT Command on page 2-10 and to the *Pathway/iTS Web Client Programming Manual*.

- The command to call SCUP is:

  ```
  1>scup
  ```

  SCUP displays the product identification message and issues a prompt.

  ```
  SCREEN COBOL UTILITY ...
  ?
  ```

- The VOLUME command sets the default volume and subvolume to $DATA.OLDSV.

  ```
  ?volume  $data.oldsv
   VOLUME IS $DATA.OLDSV
  ```

- The default object file name is set to POBJ ($DATA.OLDSV.POBJ for this example), making the files POBJCOD, POBJDIR, and POBJSYM the default SCREEN COBOL object files.

- The INFO command lists all versions of all program units in object file $DATA.OLDSV.POBJ.

  ```
  ?info  (*(*))
  EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  ACCESS ON
  $DATA.OLDSV.POBJ
  PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
    MAIN-PROG (1)       ON    6144  (C00) 25 JUN 1987  17:28:14
    TEST-PROG (1)       ON    1024  (C00) 03 JUL 1987  12:39:32
    TEST-PROG (2)       ON    1024  (C00) 27 JUL 1987  08:17:07
    TEST-PROG (3)       ON    1024  (C00) 29 JUL 1987  07:07:14
  ```

- Another INFO command lists all versions of program FUNC-PROG in object file $MKT.OLDSV.SCOB.

  ```
  ?info  $mkt.scob (func-prog(*))
  $MKT.OLDSV.SCOB
  EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  ACCESS ON
  PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
    FUNC-PROG (1)       ON    4096  (C00) 03 JUL 1987  11:43:14
    FUNC-PROG (2)       ON    4096  (C00) 29 JUL 1987  20:17:16
    FUNC-PROG (4)       ON    4608  (C00) 25 JUN 1987  13:40:18
  ```

- The DELETE command deletes all versions except the latest version of program FUNC-PROG in object file $MKT.OLDSV.SCOB.

```
?delete  $mkt.oldsv.scob (func-prog(+))
$MKT.OLDSV.SCOB
DELETED FUNC-PROG (1)
DELETED FUNC-PROG (2)
```

- The ALTER command sets to OFF the access to version 2 of program TEST-PROG in object file $DATA.OLDSV.POBJ. The OFF setting makes the version unavailable to the TCP for execution in the PATHWAY system.

```
?alter  (test-prog(2)), access off
$DATA.OLDSV.POBJ
ACCESS OFF
  TEST-PROG (2)
```

- The COPY command copies the latest version of program MAIN-PROG and all versions of program TEST-PROG in object file $DATA.OLDSV.POBJ to object file $DATA.NEWSV.POBJ. The copy creates the destination file because the destination file did not exist. The copy updates the program version numbers to show the latest versions created in the destination object file.

```
?copy  (main-prog, test-prog(*)), newsv.pobj
$DATA.OLDSV.POBJ             COPY TO    $DATA.NEWSV.POBJ
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  MAIN-PROG (1)                          MAIN-PROG (1)
  TEST-PROG (1)                          TEST-PROG (1)
  TEST-PROG (2)                          TEST-PROG (2)
  TEST-PROG (3)                          TEST-PROG (3)
```

- The FILE command sets the default object file name to $MKT.OLDSV.SCOB, making the files SCOBCOD and SCOBDIR the default SCREEN COBOL object files.

```
?file  $mkt.oldsv.scob
FILE IS $MKT.OLDSV.SCOB
```

- The COPY command copies the latest version of program FUNC-PROG in object file $MKT.OLDSV.SCOB to object file $DATA.NEWSV.POBJ. The copy updates the program version number to show the latest version created in the destination object file.

```
?copy  (func-prog), $data.newsv.pobj
$MKT.OLDSV.SCOB              COPY TO    $DATA.NEWSV.POBJ
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  FUNC-PROG (4)                          FUNC-PROG (1)
```

- The COMPRESS command compresses object file $MKT.OLDSV.SCOB to reclaim the disk space used by deleted programs in the object file. The operation also renumbers the program versions so that each of the earliest versions is version 1.

```
?compress  $mkt.oldsv.scob
COMPRESS $MKT.OLDSV.SCOB
EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
```

```
  FUNC-PROG (4)                           FUNC-PROG (1)
   COMPRESSED $MKT.OLDSV.SCOB
```

- The FILE command sets the default object file name to $DATA.NEWSV.POBJ, making the files POBJCOD and POBJDIR the default SCREEN COBOL object files.

  ```
  ?file  $data.newsv.pobj
  FILE IS $DATA.NEWSV.POBJ
  ```

- The ALTER command sets to OFF the access to the latest version of program TEST-PROG in object file $DATA.NEWSV.POBJ.

  ```
  ?alter  (test-prog), access off
  $DATA.NEWSV.POBJ
  ACCESS OFF
    TEST-PROG (3)
  ```

- The INFO command lists all versions of all program units in object file $DATA.NEWSV.POBJ.

  ```
  ?info  (*(*))
  $DATA.NEWSV.POBJ
  EXTENTS: DIR(00016,00032) COD(00032,00064) SYM(00032,000128)
  ACCESS ON
  PROGRAM (VERSION)    ACCESS SIZE  COMPILATION VER/DATE
    FUNC-PROG (1)       ON    4608  (C00) 25 JUL 1987  13:40:18
    MAIN-PROG (1)       ON    6144  (C00) 25 JUL 1987  17:28:14
    TEST-PROG (1)       ON    1024  (C00) 03 JUN 1987  12:39:32
    TEST-PROG (2)       OFF   1024  (C00) 27 JUN 1987  08:17:07
    TEST-PROG (3)       OFF   1024  (C00) 07 JUL 1987  07:07:14
  ```

- The ALTER command changes the library access flag to OFF so that new programs added to the file will have access OFF and not be available to the TCP until their access is explicitly changed to ON.

  ```
  ?alter , access off
  $DATA.NEWSV.POBJ
  ACCESS OFF
  ```

- The EXIT command terminates SCUP and returns control to the command interpreter.

  ```
  ?exit
  2>
  ```

A SCUP Messages

The SCUP program displays diagnostic messages during various phases of processing. This appendix describes two types of SCUP messages:

- General SCUP messages, which can be returned from a number of different commands
- Converter messages, which can be returned during conversion of a SCREEN COBOL requester to a web client by means of the CONVERT command

SCUP messages consist only of displayed text, with no visible message numbers.

General SCUP Messages

The following general messages can be issued during the operation of various SCUP commands.

```
ERROR - CONTINUATION LIMIT EXCEEDED
```

Cause. The continuation line limit on command input was exceeded.

Effect. The command fails.

Recovery. Shorten the command line.

```
ERROR - CONTINUATION LINE IS EOF
```

Cause. An end-of-file occurred on a continuation line.

Effect. The command fails.

Recovery. Remove the continuation character from the last line of the file called by the IN or OBEY command.

```
ERROR - FILE NOT SCOBOL - file-name
```

Cause. The file name specified in the command was not a SCREEN COBOL object file name.

Effect. The command fails.

Recovery. Specify a valid SCREEN COBOL object file name. The length of the file name must be 5 characters or fewer.

```
ERROR - ILLEGAL CHARACTER
```

Cause. The command contained an illegal character.

Effect. The command fails.

Recovery. Remove the illegal character.

```
ERROR - ILLEGAL ENTRY
```

Cause. The HELP command contained an illegal entry.

Effect. The command fails.

Recovery. Specify a valid HELP command entry. The HELP command accepts only SCUP command names and symbol names enclosed in angle brackets.

```
ERROR - ILLEGAL NAME
```

Cause. The command contained an illegal name.

Effect. The command fails.

Recovery. Check the name for accuracy and enter the correct name.

```
ERROR - ILLEGAL NAME LENGTH
```

Cause. The length of a name in the command was greater than 30 characters.

Effect. The command fails.

Recovery. Check the name for accuracy and enter the correct name.

```
ERROR - ILLEGAL NUMERIC LENGTH
```

Cause. The length of a number in the command was illegal.

Effect. The command fails.

Recovery. Check the number for accuracy and enter the correct number.

```
ERROR - ILLEGAL SYNTAX
```

Cause. The command contained illegal syntax.

Effect. The command fails.

Recovery. Correct the command syntax.

```
ERROR - OBEY LIMIT EXCEEDED
```

Cause. The limit of four nesting levels for the OBEY command was exceeded.

Effect. The command fails.

Recovery. Reduce the number of OBEY commands that call files containing OBEY commands.

```
ERROR - PROGRAM NAME NOT IN FILE - program-name (version-number)
```

Cause. The specified program was not included in the specified SCREEN COBOL object file. *`program-name`* and *`version-number`* identify the missing program.

Effect. The command fails.

Recovery. Use the INFO command to check the program name for accuracy and correct the name as needed.

```
ERROR - PROGRAM VERSION NOT IN FILE - program-name (version-number)
```

Cause. The specified program version was not included in the specified SCREEN COBOL object file. *`program-name`* and *`version-number`* identify the missing program version.

Effect. The command fails.

Recovery. Use the INFO command to check the program name for accuracy and correct the name as needed.

```
ERROR - SOURCE AND COPY FILES SAME (file-name)
```

Cause. The source and destination file names in the COPY command were the same (both were *`file-name`*).

Effect. The command fails.

Recovery. Change one of the file names.

```
FILE ERROR - file-name - message (errnum)
```

Cause. A file-system or sequential I/O procedure error (identified by *`message`* and *`errnum`*) occurred during operation of the command.

Effect. The command fails.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified.

```
FILE NAME ERROR - file-name (errnum)
```

Cause. An invalid file name (identified by *`file-name`*) was entered and a file-system error (identified by *`errnum`*) occurred.

Effect. The command fails.

Recovery. Check the file name for accuracy and correct the name as needed. If you need further information, refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual*.

```
FILE OPEN ERROR file-name - message (errnum)
```

Cause. The file specified by *`file-name`* could not be opened because of a file-system error (identified by *`message`* and *`errnum`*).

Effect. For commands other than CONVERT, the command fails. For the CONVERT command, either conversion fails for that program unit or the entire conversion fails, depending on the error.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. If the error is “nonexistent file (11)”, verify the existence of the file in the current volume and subvolume and reenter the command.

```
SUBVOLUME NAME ERROR
```

Cause. An invalid subvolume name was entered.

Effect. The command fails.

Recovery. Check the subvolume portion of the specified file name for a accuracy and correct it as needed.

```
SYSTEM NAME ERROR (system-num)
```

Cause. An invalid system name was entered.

Effect. The command fails.

Recovery. Check the specified system name for accuracy and correct it as needed.

```
SYSTEM NOT ACCESSIBLE
```

Cause. The named system could not be accessed by the user. The system might not be on the network.

Effect. The command fails.

Recovery. Correct the network problem.

```
VOLUME NAME ERROR (system-num)
```

Cause. An invalid volume name was entered.

Effect. The command fails.

Recovery. Check the volume portion of the specified file name for a accuracy and correct it as needed.

```
WARNING - PROGRAM DELETED DURING PROCESSING - program-name (version)
```

Cause. A program unit was deleted from the object file during command processing. This situation might occur if multiple references are made to the same version or if another user has deleted the program. This is a warning message and does not indicate a fatal error.

Effect. Command processing continues.

Recovery. Correct the situation as needed.

Converter Messages

The following messages can be issued by the SCUP converter during the operation of the CONVERT command, which converts a SCREEN COBOL requester to a web client.

Converter messages are of two severity levels:

- Warning messages
- Error messages

A warning message indicates that the converted code might not run properly because of some situation related to the contents of the object file; however, recovery does not require making changes to SCREEN COBOL or Java code. The converter inserts messages into the converted Java code in the form of comments to indicate the location of the problem.

An error message indicates that conversion of at least one program unit failed, and changes are needed to SCREEN COBOL or Java code. The converter inserts messages into the converted Java code to indicate the location of the problem. These messages are not comments, and the resulting code will not compile correctly; the messages will be flagged as errors by the Java compiler. For some errors, the converter proceeds to convert the next program unit; in other cases, the converter stops and all subsequent program units must also be reconverted.

To correct the problem indicated by an error message, you must do one of the following, depending on your maintenance strategy for the web client:

- Correct the problem in the SCREEN COBOL source code and then reconvert the affected program units.
- Correct the problem in the converted Java code.

In either case, if any program units were not converted because of the error, reconvert those program units.

```
Converter internal error (errnum)
```

Cause. An internal error occurred in the converter.

Effect. The conversion fails for that program unit.

Recovery. Contact your service provider and provide the following information:

- Descriptions of the problem and accompanying symptoms
- The error number displayed in *errnum*
- A copy of the program unit source code with all copy statements replaced with the actual contents.

If your local operating procedures require contacting the HP Global Customer Support Center (GCSC), supply your NonStop system number and the numbers and versions of all related products as well.

```
Error - APPNAME app-name mismatch. A prior version of the converted
software exists with a different value for the APPNAME parameter.
```

Cause. A prior attempt was made to convert the application by specifying the same INITPROG parameter but with a different APPNAME parameter.

Effect. The conversion process stops.

Recovery. Reissue the command with the correct APPNAME parameter. If you want to convert the application using the new APPNAME value, delete the MAP file and all the files in the HTMLVOL and JAVAVOL volumes that correspond to the application. Then restart the conversion. The MAP file is created in the location where the CONVERT command is issued.

```
Error - Configuration file not found config-file
```

Cause. Either the configuration file name is invalid or the converter was unable to locate the specified configuration file.

Effect. The conversion fails.

Recovery. Check whether the *`config-file`* name is valid; if not, correct it. If the *`config-file`* name is valid, make sure it is located in the specified location. Then restart the conversion.

```
Error - Configured limit violation for (limit-param) : Actual (actual-num)
Allowed (allowed-num)
```

Cause. The converter exceeded the predefined limits for the specified parameter.

Effect. The converter skips the program unit and continues to convert the remaining program units.

Recovery. In the Limits Section of the configuration file, specify a value for the limit parameter (*`limit-param`*) that is greater than *`actual-num`*. Restart the conversion.

```
Error - Creation error for MAP file map-file-name (errnum)
```

Cause. The converter could not create the specified MAP file because of the file-system error indicated.

Effect. The conversion fails.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Restart the conversion.

```
Error - INITPROG program-id mismatch. A prior version of the converted
software exists with a different value for the INITPROG parameter.
```

Cause. A prior attempt was made to convert the application by specifying a different value for the INITPROG parameter.

Effect. The conversion process stops.

Recovery. Reissue the command with the correct INITPROG parameter. If you want to convert the application using the new INITPROG value, delete the MAP file and all the files in the HTMLVOL and JAVAVOL volumes that correspond to the application. Then restart the conversion. The MAP file is created in the location where the CONVERT command is issued.

Recovery. Reissue the command with the correct INITPROG parameter. If you want to convert the application using the new INITPROG value, delete all the files in the HTML and Java volumes before restarting the conversion.

```
Error - Invalid parameter entry (entry-name) in configuration
file config-file
```

Cause. An invalid parameter entry was found in the specified configuration file.

Effect. The conversion fails.

Recovery. Edit the configuration file to assign a valid parameter value and restart the conversion.

```
Error - Missing symbol information program-unit
```

Cause. The specified program unit did not include symbol information.

Effect. The conversion fails for that program unit.

Recovery. Compile the program unit with the SYMBOLS option and restart the conversion.

```
Error - Open error for file config-file (errnum)
```

Cause. The converter could not open the specified configuration file because of the indicated file-system error.

Effect. The conversion fails.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Restart the conversion.

```
Error - Section section-name not found in configuration
file config-file
```

Cause. The specified section name was not found in the configuration file.

Effect. The conversion fails.

Recovery. Edit the configuration file to include the missing section and restart the conversion.

```
Error - Parameter param-name is not terminated with a semicolon in
configuration file
```

Cause. The specified parameter entry is not terminated with a semicolon.

Effect. The conversion fails.

Recovery. Correct the syntax error in the configuration file and restart the conversion.

```
Error - Precision may be lost (data-item-name)
```

Cause. The number of digits in the specified numeric data item (including the decimal point and the number of digits before and after the decimal point) exceeded 16. In this situation, precision might be lost in the converted code.

Effect. The converter generates an error message in the converted Java code and conversion continues for that program unit. If the converted Java code is not manually corrected, the converted program unit fails at run time with ScobolException 3304.

Recovery. There are two recovery methods:

- Edit the SCREEN COBOL source program to reduce the size of the numeric data item to fit within 16 digits and then convert the program unit again.
- Edit the Java program to correct the size of the corresponding Java object and change the return value of the getHostLength() method for all group classes to which the data item belongs.

```
Error - Program unit compiled with pre C20 version compiler
```

Cause. An attempt was made to convert a program unit that had been compiled with a C20 or earlier version of the SCREEN COBOL compiler.

Effect. The conversion fails.

Recovery. Compile the SCREEN COBOL source using a compiler whose version is C30 or later, then reconvert the application.

```
Error - Purge error for the file file-name (errnum)
```

Cause. The converter could not purge the old copy of the specified file because of the file-system error indicated.

Effect. The conversion fails.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Restart the conversion.

```
Error - Unable to allocate flat segment (errnum)
```

Cause. The converter could not allocate an extended segment. *`errnum`* indicates the file-system error that occured during the SEGMENT_ALLOCATE_ call.

Effect. The CONVERT command fails.

Recovery. Refer to the description of the file-system error or sequental I/O error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Then restart the conversion.

```
Error - Unable to create HTML file html-file-name (errnum)
```

Cause. One of the following two situations occurred:

- A file-system error or sequential I/O error (identified by *`errnum`*) occurred while the converter was creating the HTML source file specified by *`html-file-name`*.
- If *`errnum`* is blank, the current program file contained more than 10,000 screens.

Effect. The conversion fails.

Recovery. Recovery depends on the presence or absence of an *`errnum`* value in the message:

- If *errnum* is specified in the message, refer to the description of the file-system error or sequental I/O error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Then restart the conversion. If no corrective action is possible, contact your service provider.
- If no *errnum* is specified, the HTML files generated thus far are named SCRN*nnnn*, where *nnnn* is a value from 0000 and 9999. Restart the conversion and specify a different volume-subvolume combination for creating HTML files (either in the syntax of the CONVERT command or in the configuration file).

```
Error - Unable to create Java file java-file-name (errnum)
```

Cause. One of the following two situations occurred:

- A file-system error or sequential I/O error (identified by *errnum*) occurred while the converter was creating the Java source file specified by *java-file-name*.
- If *errnum* is blank, more than 10,000 program units had the same first four characters.

Effect. The conversion fails.

Recovery. Recovery depends on the presence or absence of an *errnum* value in the message:

- If *errnum* is specified in the message, refer to the description of the file-system error or sequental I/O error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Then restart the conversion. If no corrective action is possible, contact your service provider.
- If no *errnum* is specified, the Java files generated thus far are named aaaa*nnnn*, where *aaaa* is the first four characters of the SCREEN COBOL library files (POBJ) and *nnnn* is a value from 0000 and 9999. Restart the conversion and specify a different volume-subvolume combination for creating Java files (either in the syntax of the CONVERT command or in the configuration file).

```
Error - Unable to deallocate flat segment (errnum)
```

Cause. The converter could not deallocate an extended segment. *errnum* indicates the file-system error that occured during the SEGMENT_ALLOCATE_ call.

Effect. The subsequent SCUP command may fail.

Recovery. Terminate the SCUP process and restart it.

```
Error - Unsupported statement encountered during conversion of program
unit program-unit: unsupported-statement
```

Cause. The converter encountered a SCREEN COBOL statement that is not supported for conversion.

The following statements are not supported for conversion and can appear in this message:

DEVICEINFO
RECEIVE UNSOLICITED MESSAGE
REPLY UNSOLICITED MESSAGE
SEND MESSAGE
TERMINALINFO
RESTART-TRANSACTION

Effect. The conversion program continues with the conversion; however, the converted program unit does not compile correctly.

Recovery. Either modify the SCREEN COBOL source code to remove the unsupported statements or edit the generated code to handle the error messages generated by the converter in converted code.

```
Error - The Program Unit not eligible for conversion
```

Cause. This error is returned when one of the following occurs:

- The current program unit contains a TERMINAL IS clause identifying an unsupported terminal value. The supported values for the TERMINAL IS clause are:

  IBM-3270
  T16-6520
  T16-6530
  T16-6540

- The current program unit contains a message section.

Effect. The conversion fails for that program unit.

Recovery. The program unit cannot be converted.

```
FILE I/O error file-name (errnum)
```

Cause. A file-system error (identified by *errnum*) occurred during an I/O operation on the file specified by *file-name*.

Effect. The conversion fails for that program unit.

Recovery. Refer to the description of the file-system error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Then restart the conversion. If no corrective action is possible, contact your service provider.

```
HTML file error for html-file-name (errnum)
```

Cause. A file-system error or sequential I/O error (identified by *`errnum`*) occurred while the converter was writing the HTML source file specified by *`html-file-name`*.

Effect. The conversion fails for that program unit.

Recovery. Refer to the description of the file-system error or sequental I/O error in the *Guardian Procedure Errors and Messages Manual* and take corrective action as specified. Then restart the conversion. If no corrective action is possible, contact your service provider.

```
This service is only available through Professional Services.
Contact your HP representative to enable the feature.
```

Cause. The user is not authorized to run the CONVERT command.

Effect. The CONVERT command fails.

Recovery. Contact the HP Professional Services group to get your application converted or contact your HP representative to get authorization.

```
Warning - INITPROG program-id not found in file file-name
```

Cause. The converter was unable to locate the specified program ID in the INITPROG parameter in the currently opened file, identified by *`file-name`*.

Effect. The conversion continues after generating the warning. The application will fail at run time if the program identified by the INITPROG parameter was not converted.

Recovery. Check whether the INITPROG parameter is valid; if it is invalid, correct it. If the INITPROG parameter is valid, ensure that the program unit is in the POBJ file specified by *`file-name`*. Then restart the conversion.

```
Warning - The following program units are not available in the current
file dir-file-name:
program-id
program-id
...
```

Cause. The converter traces the call sequence in the POBJ file starting with the INITPROG. The converter displays this message if it was unable to locate one or more program IDs in the specified POBJDIR file (*`dir-file-name`*). The displayed message identifies the missing program IDs.

Effect. The conversion continues after generating the warning. The application will fail at run time if the program units identified by the specified *program-id*s are not converted.

Recovery. Include the missing program units in the current POBJ file and reconvert the application.

```
Warning - Unable to resolve program units used in CALL working-storage
statement
```

Cause. The converter traces the call sequence in the POBJ file starting with the INITPROG. The converter displays this message if it was unable to resolve a CALL sequence because the call sequence was of the form CALL WS1, so that it could not be resolved at the time of conversion. The displayed message identifies the associated program IDs.

Effect. The conversion continues after generating the warning. The application will fail at run time if the working-storage item represented by the identifier in the CALL statement contains a program unit name that does not exist in the current POBJDIR file.

Recovery. If a failure occurs at run time, convert the program unit identified in the working-storage item referenced in the CALL statement. If no run-time failure occurs, no recovery action is necessary.

```
Warning - Unsupported statement encountered during conversion of program
unit program-unit: unsupported-statement
```

Cause. The converter encountered a SCREEN COBOL statement that is not supported for conversion.

The following statements are not supported for conversion and can appear in this message:

CHECKPOINT
DISPLAY RECOVERY
EXIT RECOVERY
PRINT SCREEN
RECONNECT MODEM
SCROLL
USE

Effect. The conversion program continues with the conversion. The converted program will run correctly, but it will not provide the effect of any unsupported statements from the list above.

Recovery. Informational message only; no corrective action is needed.

Index