Common User Interface (CUI) User’s Guide

Abstract
This publication is designed to help guide first-time users of Emperex Corporation’s products through the set of common environment-related commands. These commands are available through all Emperex Corporation’s products which have line-oriented command interfaces.

Part Number
080000 D20

Product Version
CUI D20.13

Operating System
GUARDIAN C30-D4x, Gxx

Printing Date
September 2007

Emperex Corporation
5955 Airport Road, Suite 200,
Mississauga, Ontario, Canada,
L4V 1R9

Phone: (905) 677-6666
Fax: (905) 677-6671

www.emperex.com
Document History

<table>
<thead>
<tr>
<th>Edition</th>
<th>Part Number</th>
<th>Operating System</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fourth Edition</td>
<td>080000 C40</td>
<td>Guardian C30-D30</td>
<td>November 1995</td>
</tr>
<tr>
<td>Fifth Edition</td>
<td>080000 C40</td>
<td>Guardian C30-D30</td>
<td>January 1997</td>
</tr>
<tr>
<td>D00 Release Edition 1</td>
<td>080000 D00.01</td>
<td>Guardian C30-D30</td>
<td>May 1997</td>
</tr>
<tr>
<td>D20 Release Edition 1</td>
<td>080000 D20.08</td>
<td>Guardian C30-D30</td>
<td>October 2003</td>
</tr>
</tbody>
</table>

Copyright Notice


All rights reserved. No part of this document may be reproduced in any form, including photocopying, optical character scanning recognition, facsimile (FAX) transmission, or translation to another language without the prior written consent of Emperex Corporation.

The information contained in this document is furnished under a license agreement and non-disclosure agreement. The software and associated documentation may be used only in accordance with the terms of the agreement.

Trademarks or Service Marks

The following are trademarks or service marks of Emperex Corporation.

DBOL, DDSDIFF, DDSMAKE, GMAIL, InterAccess, RMS, TBOL

The following are trademarks or service marks of HP Computers Incorporated:

DDL, Enscribe, Expand, Guardian, NonStop, PATHMAKER, TACL, TAL, Tandem, Transfer.
Table of Contents

Preface

| Purpose and Scope                  | v |
| Organization                       | v |
| Who Should Read What               | v |
| Related Manuals and Products       | vi |

Common User Interface (CUI) Facilities 1

| Capabilities                      | 1-1 |

Environment Commands 2

| History Buffer                      | 2-2 |
| ! Command                           | 2-3 |
| ? Command                           | 2-4 |
| ALIAS Command                       | 2-5 |
| ALLOW Command                       | 2-8 |
| COMMENT Command                     | 2-9 |
| DELAY Command                       | 2-10|
| ECHO Command                        | 2-11|
| ENV Command                         | 2-12|
| ENV CONTINUE Command                | 2-14|
| ENV HEADER Command                  | 2-15|
| ENV HELP Command                    | 2-16|
| ENV PAGE Command                    | 2-17|
| ENV PMSG Command                    | 2-18|
| ENV STATS Command                   | 2-19|
| EXIT Command                        | 2-20|
| FC Command                          | 2-21|
| FILEINFO Command                    | 2-23|
| FILENAMES Command                   | 2-25|
| FILES Command                       | 2-27|
| HELP Command                        | 2-29|
| HISTORY Command                     | 2-31|
| IF Command                          | 2-32|
| LOG Command                         | 2-34|
| OBEY Command                        | 2-35|
| OUT Command                         | 2-36|
| RUN Command                         | 2-37|
| SETPROMPT Command                   | 2-39|
| SYSTEM Command                      | 2-42|
| TIME Command                        | 2-43|
| VOLUME Command                      | 2-44|
Table of Contents

Interacting with the CUI

Entering Commands .................................................................3-1
Using Input Files .................................................................3-3
Preface

Welcome to the Common User Interface to all Emperex Corporation’s products. Our goal in designing the CUI is to make a basic set of commands available to all of our utilities which are already familiar to Tandem users.

We at Emperex are proud of our products. We take pride in the level of support that we provide. If you have any questions regarding this, or any other Emperex product, software, or support materials, please contact your Emperex representative.

Purpose and Scope

This publication is intended to assist new users of Emperex products in becoming familiar with the common user interface (CUI). This manual will also provide introductory information for users who have only a general knowledge of Tandem computer systems.

This publication applies to all current Tandem computer systems which run Release D00 or later of the Guardian 90 operating system.

Organization

The organization of this publication is intended to help you understand the commands which make up the common user interface (CUI).

- **Section 1** provides a brief overview of the CUI facilities.
- **Section 2** provides detailed information about each command in the CUI.
- **Section 3** describes the various ways in which users communicate with the CUI. The techniques available to users are also described.

Who Should Read What

This publication is written for all users new to Emperex products.
Here is a list of RMS manuals:

- The Introduction to RMS (Part No. 80201) provides introductory information on the capabilities of the RMS product. You should read this manual first!
- The RMS User’s Guide (Part No. 80207).
- The RMSCOM and RMSMON Reference Manual (Part No. 80203) contains detailed reference information for using RMS.
- The RMS Tutorial Guide (Part No. 80208) gives step-by-step instructions on how to use RMS.
- The RMS Utilities Reference Manual (Part No. 80206) describes the utility programs associated with RMS.
- The RMS Installation Guide (Part No. 80202).
- The RMS Programming Manual, Volumes 1 and 2 (Part Nos. 80204 and 80205).
- General Transfer/Mail™ Gateway Reference and Customization Guide (Part No. 84021 B40).

The following are D30 versions of Tandem publications and are supplemental:

- GUARDIAN Operating System User’s Guide
- GUARDIAN Programmer’s Guide
- GUARDIAN System Operations Guide
- ENSCRIBE™ Programmer’s Guide
- TACL Programming Guide
- TACL Reference Manual
- Pathway PATHCOM Reference Manual
- Pathway System Management Guide
- NonStop™ SQL/MP Reference Manual
- NonStop™ SQL/MP Report Writer Guide
You might also find these manuals helpful:

- Programmatic Network Administrator (PNA) RDL Manual (Part No. 48818)
- Programmatic Network Administrator (PNA) System Management Manual (Part No. 48815).
Notes:
The following list summarizes the conventions for syntax notation in this manual:

### Syntax Conventions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE LETTERS</td>
<td>Upper-case letters represent keywords and reserved words; you must enter these items exactly as shown.</td>
</tr>
<tr>
<td>lowercase letters</td>
<td>Lower-case italic letters represent variable items that you must supply.</td>
</tr>
<tr>
<td>Brackets [ ]</td>
<td>Brackets enclose optional syntax items. A vertically-aligned group of items enclosed in brackets represents a list of selections from which you may choose one or none.</td>
</tr>
<tr>
<td>Braces { }</td>
<td>Braces enclose required syntax items. A vertically-aligned group of items enclosed in braces represents a list of selections from which you must choose only one.</td>
</tr>
<tr>
<td>Vertical Line()</td>
<td>A vertical line separating items in a list enclosed in either braces or brackets is an alternative to vertical alignment of the options (usually in short lists that can be easily shown on one line).</td>
</tr>
<tr>
<td>Ellipsis ...</td>
<td>An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed syntax item any number of times.</td>
</tr>
<tr>
<td>Spaces</td>
<td>If two items are separated by a space, that space is required between the items. If one of the items is a punctuation symbol such as a parenthesis or a comma, spaces are optional.</td>
</tr>
<tr>
<td>Punctuation</td>
<td>Parenthesis, commas, semicolons, and other symbols or punctuation not described above must be entered precisely as shown. If any of the punctuation above appears enclosed in quotation marks, that character is a required character and you must enter it as shown.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Continue to type the command as one line.</td>
</tr>
<tr>
<td>change-xxxx</td>
<td>This is the change number, found in the left margin. It indicates that a change or addition has been made to both the software and the manual since the last edition.</td>
</tr>
</tbody>
</table>

This line indicates where the changed or new information ends.

All Returns are implicit.
1 Common User Interface (CUI) Facilities

The Common User Interface (CUI) provides a common set of commands to all Emperex Corporation’s products. After learning one product, you will already have the basics needed to learn another product.

The CUI presented in this manual is design to support line-oriented command interfaces on Tandem computer systems running the GUARDIAN 90 operating system versions C10 or higher.

Capabilities

The CUI supports the following facilities:

- **Command History.** The CUI maintains a history of commands that you enter during a session in any Emperex product. You can examine, edit, and re-execute previously-entered commands.

- **Program Execution.** The CUI allows you to start up other programs from within the CUI. You can even turn on the display of process management messages.

- **Command Logging.** The CUI can provide a complete log of interactions between the user and the product supporting the CUI.

- **Environment Control.** The CUI can display and change common environment settings, such as default volume.

- **Output Redirection.** You can have the CUI redirect its output to another file, terminal, or spooler location.

- **Input Redirection.** The CUI can read commands from alternate input files. The facility even supports sections within input files.

- **On-line Help.** The CUI manages the on-line help portion of an application. All applications using the CUI present help in the same manner.
Notes:
2 Environment Commands

The commands implemented within the CUI allow the user to interact or query files and processes that are common to all applications. The commands allow a user to redirect the input and output of application commands, execute programs, change the default volume, manipulate the command history buffer, display file names, and, in some applications get help on command syntax.

The following environment commands are supported by the CUI:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Re-executes a command.</td>
</tr>
<tr>
<td>?</td>
<td>Displays a previous command.</td>
</tr>
<tr>
<td>ALIAS</td>
<td>Controls command aliases.</td>
</tr>
<tr>
<td>ALLOW</td>
<td>Controls the number of errors and warnings allowed per command.</td>
</tr>
<tr>
<td>COMMENT</td>
<td>Places a comment into a command file.</td>
</tr>
<tr>
<td>DELAY</td>
<td>Pauses RMSCOM for a specific period.</td>
</tr>
<tr>
<td>ECHO</td>
<td>Sends text to the output file(s).</td>
</tr>
<tr>
<td>ENV</td>
<td>Displays and controls current environment settings.</td>
</tr>
<tr>
<td>ENV CONTINUE</td>
<td>Causes the application program to continue processing input after the command line options and commands have been processed.</td>
</tr>
<tr>
<td>ENV HEADER</td>
<td>Turns the display of page header information on or off when a page break occurs.</td>
</tr>
<tr>
<td>ENV HELP</td>
<td>Sets the default level of detail associated with help queries.</td>
</tr>
<tr>
<td>ENV PAGE</td>
<td>Changes the default page size.</td>
</tr>
<tr>
<td>ENV PMSG</td>
<td>Controls the display of process-control information.</td>
</tr>
<tr>
<td>ENV STATS</td>
<td>Controls the display of statistics when the command-level interface is terminated.</td>
</tr>
<tr>
<td>EXIT</td>
<td>Terminates RMSCOM processing.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>FC</td>
<td>Fixes a previous command.</td>
</tr>
<tr>
<td>FILEINFO</td>
<td>Displays detailed file information.</td>
</tr>
<tr>
<td>FILENAMES</td>
<td>Displays names of specified files.</td>
</tr>
<tr>
<td>FILES</td>
<td>Displays files in a subvolume.</td>
</tr>
<tr>
<td>HELP</td>
<td>Obtains help.</td>
</tr>
<tr>
<td>HISTORY</td>
<td>Displays previous commands.</td>
</tr>
<tr>
<td>IF</td>
<td>Controls conditional command processing.</td>
</tr>
<tr>
<td>LOG</td>
<td>Logs commands and responses to a file.</td>
</tr>
<tr>
<td>OBEY</td>
<td>Executes commands from a disk file.</td>
</tr>
<tr>
<td>OUT</td>
<td>Redirects output to a file.</td>
</tr>
<tr>
<td>RUN</td>
<td>Runs a program from within RMSCOM.</td>
</tr>
<tr>
<td>SETPROMPT</td>
<td>Changes the default prompt text.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Changes the default system.</td>
</tr>
<tr>
<td>TIME</td>
<td>Displays current date and time.</td>
</tr>
<tr>
<td>VOLUME</td>
<td>Changes the current default volume.</td>
</tr>
</tbody>
</table>

**History Buffer**

All programs supporting the CUI contain a buffer to hold commands. Commands are entered into the buffer when a command interpreter reads a line from the input file. The buffer is 2048 bytes long. A number of commands are provided to examine and manipulate commands in the buffer. These include: `?` to display a command in the buffer; `!` to repeat a command in the buffer; and `FC` to correct or modify a command in the buffer and optionally repeat it. Commands may be referenced using absolute command numbers, relative command numbers, or by the text of the command itself. The HISTORY command provides a method of displaying the contents of the history buffer.
! Command

The ! command displays and re-executes a command from the history buffer.

\[
\begin{align*}
! & \{ [ \text{-} \text{number} ] \} \\
& \{ [ \text{string} ] \}
\end{align*}
\]

\textit{number}

is an unsigned integer.

\textit{string}

is any text that does not begin with a digit.

Considerations

▼ The ! command without any parameters re-executes the previous command.

▼ The ! command with a negative number re-executes the relative command number. For example, ! -1 re-executes the previous command and is equivalent to ! with no parameters.

▼ The ! command with a positive number re-executes the command-numbered number. The HISTORY and ? commands provide absolute command numbers which are appropriate for this form of the ! command.

▼ The ! string form of the command searches the history buffer for the most recent command that begins with string and re-executes the command. Trailing blanks are ignored for this form of the ! command.

▼ This command is only available if the CUI is interactive. This command is not allowed in OBEY files.
? Command

The ? command displays a command from the history buffer.

\[
? \{ \ [\ [-]\textit{number} \ ] \ \}
\{ \ [\ \textit{string} \ ] \ \}
\]

\textit{number}

is an unsigned integer.

\textit{string}

is any text that does not begin with a digit.

Considerations

\begin{itemize}
\item ▼ \ ? alone displays the preceding command.
\item ▼ \ ? \ textit{number} displays the command-numbered number.
\item ▼ \ ? \ `-number displays the relative command number.
\item ▼ \ ? \ textit{string} displays the most recent command that starts with \textit{string}. Trailing blanks are ignored for this form of the ? command.
\item ▼ This command is only available if the CUI is interactive. This command is not allowed in OBEY files.
\end{itemize}
ALIAS Command

The ALIAS command controls the processing of command expansion aliases.

```
ALIAS [ name [ "command-text" ] ]
```

`name`

is the command alias to be added, deleted, or modified.

`command-text`

is the command string to be substituted for `name`. This string can contain any text that corresponds to a valid command.

Considerations

- ALIAS on its own, displays all current aliases.
- ALIAS name will delete the alias corresponding to name.
- ALIAS name command-text will add a new alias if name is not currently defined, or change the current definition of name if it is already defined.
- Aliases are restricted to 80 bytes. The total expansion of an alias cannot exceed 253 bytes.
- The alias facility supports parameter substitution. If %n% appears in the command-text, the n-th token in the input text will be substituted. %0% corresponds to the alias name.
- To insert a double quote into a prompt string use ".
- To insert a percent into a prompt string without expansion use \%.
- To insert a backslash required as part of an application command, four backslashes are required. Use \\\\.
- Any command can be aliased except for the ALIAS command. The command

```
ALIAS FILES "FILEINFO *
```

is a perfectly valid alias definition.

- The number of allowed aliases may vary between applications. The standard maximum number of aliases is 128.
Example

The following example illustrates parameter substitution within an alias:

```
CUI 1) ALIAS ci "info component %1%,detail"
CUI 2) ci *
yields: info component *,detail
```

Function Key Alias Definitions

Function keys can be assigned commands in a manner similar to the normal command aliases.

```
ALIAS SF16 "EXIT"
```

sets up an alias to shift-f16 to the EXIT command.

Function key aliases can accept parameters. Parameters must be entered on the command line before the function key.

```
CUI) ALIAS F1 "FILES 
CUI) $DATA.CATALOG<f1>
```

translates to a FILES command on the $DATA.CATALOG subvolume.

Note!

Currently, function keys are only supported for Tandem 65xx terminals and terminal emulator products.

Note!

ALIAS supports an additional parameter form. The "%*%" parameter will place the entire input string, except for the initial keyword, into the resulting command. For example, using the ALIAS command:

```
ALIAS aa "INFO %*%"
```

When you enter the command:

```
aa comp \SYS\APPL\BCOMP
```

in effect, it becomes:

```
INFO COMP \SYS\APPL\BCOMP
```
ALLOW Command

The ALLOW command controls the maximum number of errors or warnings that can occur within a command before the command processing terminates. If the limit is exceeded, the current command, OBEY or IN-file processing is terminated.

The ALLOW command is only relevant for OBEY files and IN-file processing.

```
ALLOW [ ALL ] [ ERRORS ]
[ NO ] [ WARNINGS ]
[ count ]
```

**ALL**

tells the CUI that it should continue processing regardless of the number of errors or warnings.
An OBEY file will continue to execute lines it can execute successfully and not stop if one line fails.

This command will **not** allow an RMS operation which requires several steps to be successfully completed in succession to continue when an error is encountered.

**NO**

tells the CUI that it should immediately stop when an error or warning occurs.

**count**

indicates the maximum number of errors or warnings that can occur before command termination. count must be greater than 0 and less than 32768.

**Considerations**

- Initially, no errors and all warnings are allowed.
- If none of ALL, NO, or count is specified, ALL is assumed.
- If none of ERRORS or WARNINGS is specified, ERRORS is assumed.
- ALLOW on its own causes the number of errors and warnings to be reset to their initial values.
COMMENT Command

The COMMENT command documents command or obey files.

```
COMMENT any text
```

or

```
== any text
```

Considerations

- The COMMENT command can appear anywhere in an obey file.
- The `==` is a valid alias for the COMMENT command and has the same semantics as COMMENT.
- `==` will not be treated as a comment if it appears as part of another command. You will have to terminate the other command first, as described below.

Examples

The following are valid comments:

```
CUI) COMMENT    This is a comment line
CUI) == This is also a comment line
CUI) ENV; == This is how to place a comment on a command line
```
DELAY Command

The DELAY command pauses RMSCOM (or any other process using the CUI) for a specific period.

\[
\text{DELAY } n \ {\text{SECONDS | MINUTES | HOURS}}
\]

\(n\)

is a valid number from 0 to 32767.

Considerations

None.
ECHO Command

The ECHO command sends text to the current output file(s).

```
ECHO "string" ...
```

\textit{string}

is any quoted string of characters.

Considerations

ECHO is really only useful in OBEY files.

Examples

The following displays text:

```
CUI) == Dump a blank line
CUI) ECHO

CUI) == Dump strings
CUI) ECHO "Hello" "world"
Hello world.
```
ENV Command

The ENV command writes to the out file the SAVED VOLUME, the CURRENT VOLUME and the current LOG file name (if one exists) as well as other environment parameters.

```
ENV
```

Considerations

None.

Examples

The ENV command produces the following report to the current OUT file:

```
CUI VERSION          C30
SAVED VOLUME         startup-volume
CURRENT VOLUME       current-volume
HELP LEVEL           { SYNTAX | DETAIL }
CURRENT SECURITY     “GGGG”
SAVED SECURITY       “GGGG”
DEFINE               =_DEFAULTS, CLASS DEFAULTS,
                      VOLUME current-volume
HOME-TERM            home-terminal
USER-ID              guardian-user-id
STATS                { ON | OFF }
PMSG                 { ON | OFF }
PAGE                 60, 80
HEADINGS             { ON | OFF }
ERRORS ALLOWED       { NONE | ALL | count }
WARNINGS ALLOWED     { NONE | ALL | count }
LOG                  logging-state
```

`startup-volume`

is the volume that the CUI received in its startup message from the process that started the CUI command interpreter.

`current-volume`

is the current volume that the CUI is using for file name expansion. It will be the same as the startup volume unless a VOLUME or SYSTEM command has been issued to change the current volume.

`home-terminal`

corresponds to the TERM parameter specified in the RUN command.
guardian-user-id

is the current process accessor id (PAID) of the CUI process.

60, 80

is the length (in lines) and the width (in characters) of the page.

logging-state

is either the name of the current log file, or the message “logging disabled” if no log file is currently being used.
ENV CONTINUE Command

The ENV CONTINUE command causes the application program to continue processing input after the command line options and commands have been processed. The default action for most applications is to terminate after processing the command line.

```
ENV CONTINUE { ON | OFF }
```

Considerations

None.
ENV HEADER Command

The ENV HEADER command turns the display of page header information on or off when a page break occurs.

```
ENV HEADER { ON | OFF }
```

Considerations

None.
ENV HELP Command

The ENV HELP command sets the default level of detail associated with help queries. Please refer to the HELP HELP documentation for the levels of detail available.

ENV HELP { ON | OFF }

Considerations

None.
ENV PAGE Command

The ENV PAGE command changes the default page size. Adjusting page sizes causes a page break to occur on the next output. The new page size takes effect on the next page.

```plaintext
ENV PAGE { ON | OFF }
```

Considerations

None.
The ENV PMSG command controls the display of process-control information when processes are started and stopped while using the RUN command.

```
ENV PMSG { ON | OFF }
```

**Considerations**

- By default, no process-control messages are displayed. ENV PMSG ON allows the display of these messages.

- The following process messages are supported:
  - Process Startup messages report the <CPU, PIN> or process name of the process started by a RUN command.
  - Normal Process Termination messages report the <CPU, PIN> or process name of the process which stopped, followed by any additional details which the terminated process specified when it shut down.

Abnormal Process Termination messages report the <CPU, PIN> or process name of the process which abended, followed by any additional details which the abnormally-terminated process specified when it halted.
ENV STATS Command

The ENV STATS command controls the display of statistics when the command-level interface is terminated. This command has no other effect during program execution.

```
ENV STATS { ON | OFF }
```

Considerations

- By default, the number of errors and warnings are displayed when the program terminates. ENV STATS OFF inhibits the display of these statistics.

- Applications supporting the CUI have the choice of displaying these statistics or not when commands are specified either in an OBEY file or on the TACL RUN command line.
EXIT Command

The EXIT command stops the current command interpreter process.

EXIT

Considerations

Termination processing for each product can be different. Please consult the appropriate reference manual for the product that you are using for specific details on what that application does when a command interpreter shuts down.
FC Command

The FC command changes or corrects any command that is still in your command history buffer.

```
FC  { [ [-]number ]  }
   { [ string ]  }
```

*number*

is an unsigned integer.

*string*

is any string not beginning with a digit.

**Considerations**

- FC alone allows you to correct the immediately-preceding command.
- FC number allows you to correct the specific command number.
- FC -number allows you to correct the command that preceded the current command by number commands; for example, FC -1 is the same as FC.
- FC string allows you to correct the most recent command that started with string. Trailing blanks are ignored for this form of the FC command.
- This command is only available if the CUI is interactive. This command is not allowed in OBEXY files.
Editing Template

The following description summarizes the syntax of the editing template for the FC command.

```
subcommand [ // subcommand ] ...
```

subcommand is any of:

```
//
```

is a separator, allowing multiple subcommands on a given line. A subcommand can
immediately follow one or more uppercase or lowercase D’s without being preceded by //.

```
{ R | r } replacement-string
```

replaces characters in the previous command, starting with the character displayed
immediately above the R or r. A replacement-string preceded by R or r can be any string
of characters, including spaces, and can itself begin with R, I, or D (or r, i or d).
Characters in replacement-string replace characters in the previous command on a
one-for-one basis.

```
{ I | i } inserting-string
```

inserts characters into the previous command in front of the character displayed above the
I or i.

```
{ D | d }
```

deletes characters in the previous command. Any original character displayed above a D
or d that begins a subcommand in the editing template is deleted.

replacement-string

is any text that does not begin with R, I, or D (or i, r, or d). Characters in
replacement-string replace characters immediately above them on a one-for-one basis. For
example, a D in replacement-string replaces the character displayed above it instead of
deleting it.

The editing template processor in CUI conforms to the Tandem editing template standard
described with the FC command in the TACL Reference Manual.
FILEINFO Command

The FILEINFO command displays information about files in the specified subvolume.

```
$DATA.RMS

FILEINFO [ /OUT list-file / ] [ filename-template ]

<table>
<thead>
<tr>
<th>Code</th>
<th>Code</th>
<th>Last Modification</th>
<th>Owner</th>
<th>RW</th>
<th>PE</th>
<th>PExt</th>
<th>SExt</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE1 100</td>
<td>407106</td>
<td>20-Jun-90</td>
<td>10:44:31</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>FILE2 101</td>
<td>97366</td>
<td>20-Jun-90</td>
<td>10:02:50</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>FILE3 100</td>
<td>317708</td>
<td>22-Jun-90</td>
<td>19:05:30</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>FILE4 100</td>
<td>270908</td>
<td>22-Jun-90</td>
<td>16:55:02</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td>FILE5 101</td>
<td>55954</td>
<td>20-Jun-90</td>
<td>08:48:19</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>FILE6 100</td>
<td>213062</td>
<td>19-Jun-90</td>
<td>16:45:26</td>
<td>15,253</td>
<td>“GGGG”</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>
```

`list-file` is the name of the file that receives the output from the FILEINFO command. If you omit this parameter, output is sent to the current OUT file that is in effect for the current CUI.

`filename-template` is file name specification template. A template must be formatted as follows:

```
[[[ \system. ] $volume. ] subvolume. ] filename
```

The following template characters can appear in any element of a filename-template, including the system and volume fields:

- `*` Matches zero or more characters.
- `?` Matches any single character.

Considerations

- ▼ If the system, volume, or subvolume parameters are not specified then the current volume is used to supply any missing parameters.

- ▼ There is no guaranteed presentation order of systems and volumes when template characters are present in the respective parameters. Subvolumes and file names are always presented in alphabetic sequence.

- ▼ If no files match the specified template, the CUI will report an error with the following message:

```
No files matched template.
```
Examples

The following is an example of the output generated by the FILEINFO command:

where:

$DATA.RMS

is the subvolume name.

FILE1..FILE6

are the names of the files in $DATA.RMS.
FILENAMES Command

The FILENAMES command displays the files in the specified subvolume.

```
FILENAMES [ /OUT list-file / ] [filename-template]
```

`list-file`

is the name of the file that receives the output from the FILENAMES command. If you omit this parameter, output is sent to the current OUT file that is in effect for the current CUI.

`filename-template`

is a filename specification template. A template must be formatted as follows:

```
[[ \system. ] $volume. ] subvolume. ] filename
```

The following template characters can appear in any element of a filename-template, including the system and volume fields:

- `*` Matches zero or more characters.
- `?` Matches any single character.

Considerations

- ▼ If the system, volume, or subvolume parameters are not specified then the current volume is used to supply any missing parameters.

- ▼ There is no guaranteed presentation order of systems and volumes when template characters are present in the respective parameters. Subvolumes and file names are always presented in alphabetical sequence.
Examples

The following is an example of the output generated by the FILENAMES command:

```
CUI) FILENAMES $DATA.RMS.FILE*

$DATA.RMS
FILE1  FILE2  FILE3  FILE4  FILE5  FILE6
```

where

$DATA.RMS

is the subvolume name.

FILE1 .. FILE6

are the names of the files in $DATA.RMS.
FILES Command

The FILES command displays the files in the specified subvolume.

```plaintext
FILES [ /OUT list-file / ] [ subvolume-template ]
```

`list-file`

is the name of the file that receives the output from the FILES command. If you omit this parameter, output is sent to the current OUT file that is in effect for the current CUI.

`subvolume-template`

is subvolume specification template. A template must be formatted as follows:

```plaintext
[[ \system. ] $volume. ] subvolume
```

The following template characters can appear in any element of a subvolume-template, including the system and volume fields:

```
*       Matches zero or more characters.
?
Matches any single character.
```

Considerations

- If the system, volume, or subvolume parameters are not specified then the current volume is used to supply any missing parameters.

- There is no guaranteed presentation order of systems and volumes when template characters are present in the respective parameters. Subvolumes and file names are always presented in alphabetic sequence.
Examples

The following is an example of the output generated by the FILES command:

```
$DATA.RMS
FILE1   FILE2   FILE3   FILE4   FILE5   FILE6
```

where:

`$DATA.RMS`

is the subvolume name.

`FILE1..FILE6`

are the names of the files in $DATA.RMS.
HELP Command

The HELP command displays the syntax of any command interpreter, CUI command or syntactic entity, or obtains an explanation of command interpreter error numbers.

The HELP command is an optional part of the CUI and is implemented by the appropriate command interpreter. Not all applications provide the same level of on-line help.

```
HELP [ / OUT list-file / ] [help-topic]
   [ , { CONSIDER } ]
   [ , { DETAIL } ]
   [ , { EXAMPLE } ]
   [ , { PREAMBLE } ]
   [ , { SECCOND } ]
   [ , { SEE } ]
   [ , { SYNTAX } ]
```

list-file

specifies the file to receive the HELP command listing. If you omit this option the output from the HELP command is sent to the current OUT file in effect for the current CUI.

CONSIDER

gets the considerations section of the help topic for the specified command.

DETAIL

gets all sections of the help topic and expands any syntactic elements found for the specified command.

EXAMPLE

gets any example sections in the help topic.

PREAMBLE

gets the description of the help for the specified command.

SECCOND

gets any security considerations for the help topic.

SEE

gets any SEE ALSO references in the help topic.
**SYNTAX**

gets the syntax section of the help topic.

**Considerations**

- If list-file is not provided then HELP runs interactively if there are subjects associated with help-topic.

- When HELP is entered with no following parameters, it displays a list of all application commands and subjects for which help is available.
HISTORY Command

The HISTORY command displays previously-entered commands from the history buffer.

```
HISTORY [ / OUT file-name / ] [ number ]
```

- **file-name**
  - is the name of the file that will receive the output generated by the HISTORY command. If `/ OUT file-name /` is not provided then the output is sent to the current out file of the CUI.

- **number**
  - is the number of commands to display from the history buffer. If number is not provided then 10 commands are displayed.

Considerations

The history buffer can contain a maximum of 2048 bytes of command text.

History Command Report Format

```
CUI 10} HISTORY
CUI command num}command
CUI command num}command
.   .   .
.   .   .
.   .   .
```

- **command num**
  - is the command number shown by the CUI when it prompts the user.

- **command**
  - is the command text that was entered at the prompt.
IF Command

The IF command controls conditional execution of commands within an OBEY input file.

```
IF argument
    [ commands ]
[ ELSE
    commands ]
ENDIF
```

**argument**

is one of the supported argument selections that evaluated to a TRUE expression:

- **T65xx**
  
evaluates to TRUE if and only if the device on which the application is running is a Tandem 6510, 6520 ... type of terminal.

- **T327x**
  
evaluates to TRUE if and only if the device on which the application is running is an IBM 327x terminal.

- **TTY**
  
evaluates to TRUE if and only if the device is neither a T65xx or T327x terminal.

**commands**

is any supported application or CUI command.

**Considerations**

It is possible to specify nested IF statements as long as there is a corresponding ENDIF statement for the respective IF command.
Example

```
IF T65xx
    ECHO "This application is running on a Tandem terminal!"
ELSE
    IF T327x
        ECHO "This application is running on an IBM terminal!"
    ELSE
        ECHO "This application is running on a conversational terminal!"
    ENDIF
ENDIF
```
LOG Command

The LOG command provides a mechanism to capture the commands and responses provided by the CUI command interpreter.

```
LOG [ file-name [ ! ] ]
```

`file-name`

is the name of the log file for commands and responses.

Considerations

- ▼ If no file name is provided and a log file is in use, the log file is closed and logging is disabled.
- ▼ If a file name is provided then any current log file is closed and the new file name is opened. If an exclamation is also provided then the contents of the new log file are purged when the file is opened, otherwise the new log output will be appended to any current contents of the file.
- ▼ If the file name provided does not exist then an Edit file is created with extent size of 2 pages.
OBEY Command

The OBEY command executes a series of commands from a disk file.

```
OBEY [ ! ] file-name [ ( section-name ) ]
```

`file-name`

is the name of a disk file containing application commands.

`section-name`

is the optional section within `file-name` which is to be used. If no section is specified, all sections of `file-name` are used.

Considerations

▼ OBEY commands may not be nested within an OBEY file.

▼ If the initial IN file or the initial OUT file is a terminal, then BREAK is monitored. If BREAK is encountered, the processing of the OBEY file is terminated.

▼ If the ! is used, commands read from `file-name` are not echoed to the OUT file.
OUT Command

The OUT command provides a mechanism to redirect the output generated by one or more application commands.

```
OUT [ file-name [ ! ] ]
```

`file-name`

is the name of the file where the output of the CUI command interpreter is sent.

Considerations

▼ If the OUT command is entered with no parameters and an alternate OUT file is in use then the alternate OUT file is closed and any further output is appended to the initial OUT file. If a file name is provided then a new alternate OUT file is opened. If an exclamation is also provided the current contents of the file are purged; otherwise the current contents are left and any new output will be appended to the file.

▼ If the file name does not exist then an Edit-format file is created with extent size of 2 pages.

Examples

OUT file-name produces the following message in file-name:

```
Out file file-name opened
```

OUT with no file name produces the following message in the initial OUT file, if an alternate OUT file was in use:

```
Out file file-name closed
```
The RUN command starts another program from within a CUI.

```
RUN program-file
    [ / run-option [ , run-option ] / ] [ startup ]
```

**program-file**

is the name of the disk file containing the object program to be run. Partial names are expanded using the current volume of the CUI. If the name does not contain a period then the program file is expanded first using $SYSTEM.SYSTEM as the default volume, then using the current volume of the CUI.

**run-option** is any of the following:

**CPU cpu-number**

is the number of the processor where the new process is to execute. Specify cpu-number as in integer in the range from 0 - 15 inclusive. If you omit this option the process executes in the same CPU as the CUI.

**IN file-name**

is the new process’s input file. This file name is sent to the new process in its startup message. If you do not include this parameter then the home terminal of the CUI is used.

**OUT file-name**

is the new process’s output file. This file name is sent to the new process in its startup message. If you do not include this parameter then the home terminal of the CUI is used.

**PRI priority**

is the new process’s execution priority. Processes with higher numbers are executed first. Specify priority as an integer in the range 1 through 199, inclusive. If you omit the PRI option then the new process runs at the same priority as the CUI.

**NAME [ process-name ]**

is the name of the process that the CUI will create. This process name is a standard Tandem process name. If no name is provided, a name is assigned by the GUARDIAN 90 Operating System.
startup

is a program parameter or a string of parameters sent to the new process in the startup message. Leading blanks and trailing spaces are deleted.

Considerations

▼ A CUI may become hung if the process that is started does not process the startup sequence properly.

▼ The param-string can contain semi-colons (;). These are correctly passed to the program being executed. Be aware that the RUN command is considered to be the last command in a line.

Examples

The command:

```
CUI 4) RUN FUP;ENV
```

will cause FUP to receive “;ENV” in its startup message. The CUI will not execute the ENV command following the termination of the FUP process.
SETPROMPT Command

The SETPROMPT command changes the prompt string shown by a command-level interface.

```
SETPROMPT "new-prompt-string"
```

`new-prompt-string`

is the actual text of the string to be used as a prompt. This string can contain meta-characters which expand to program-specific values. See the considerations section below for details.

Considerations

- The new prompt string should be short and concise. Excessive amounts of information in the prompt string can be confusing. The initial application CUI prompt string is usually of the form "!) ". This will result in the command number being printed at each prompt.

- The meta-characters described can appear in a prompt string. These characters are expanded to the indicated values:

  - `!` Expands to the current command number.
  - `$D` Expands to the current date (DDMONYYYY).
  - `$T` Expands to the current time (HH:MM:SS).
  - `$V` Expands to the current disk volume. This string will be in network form if the current volume is not of the local node ($VOLUME or \NODE.$VOLUME).
  - `$S` Expands to the current subvolume (SUBVOL).
  - `$N` Expands to the current node name.
  - `$U` Expands to the current user name (PROCESSACCESSID)
  - `$1-$9` Expands to the application supplied parameters. These parameters will be documented in the relevant application reference manual.
The following special expansion characters can be placed in a setprompt string:

```
\\! Expands to an exclamation mark, “!”.
\\$ Expands to a dollar sign, “$”.
\\\\ Expands to a backslash, “\”.
\\e Expands to an escape character, % 033.
\\b Expands to a bell, % 007.
\\r Expands to a carriage return, % 015.
" Expands to a double quote.
```

Note! Remember that if you wish a space to appear at the end of the prompt string, just before the user input, you must have a space as the last character of the new prompt.

\e will yield one character on output.

▼ If the ST parameter is used, the value of the time field in the prompt string displayed in the HISTORY command has no meaning.

▼ Prompt strings are restricted to 77 bytes on output.
Examples

The following is a sample session from Emperex’s RMS for managing application software:

RMS 1) setprompt "!>"
2>group \APPL\PROJ\EDI
3>setprompt "$2:$3 !) "
$RMP:\APPL\PROJ\EDI 4) group SERVER1
$RMP:\APPL\PROJ..\SERVER1 5) group MODULE1
$RMP:\APPL\PROJ..\MODULE1 6) setprompt "$4 !) "
   \APPL\PROJ\EDI\SERVER1\MODULE1 7) setprompt "$D RMS !) "
21MAY1990 RMS 8) setprompt "$V.$S !) "
$SYSTEM.NOSUBVOL 9) setprompt "RMS !) "
RMS 10)

Note!

It is possible on 65xx terminals to place the prompt on line 25. The following setprompt string illustrates this:

RMS 10) setprompt "!\\eo\\e6\\$N.RMSCOM $2 (\$u) \e6 & $4\r)"

The 25th line will contain some string that looks like:

\DDS.RMSCOM $RMP (SYS.MGR) \APPL\PROJ\SERVER1\MODULE
SYSTEM Command

The SYSTEM command temporarily changes your current system setting from the startup system of the command interpreter. The current system is used for file name expansion within the CUI.

\[
\text{SYSTEM [ system-name ]}
\]

*system-name*

is the name of the EXPAND node to use for file name expansion; the current volume and subvolume names remain unchanged.

Considerations

If no parameter is provided, the local system is established as the current system. You can restore the default system, volume, and subvolume using the VOLUME command.
TIME Command

The TIME command displays the current date and time to the CUI OUT file.

```
TIME
```

Considerations

The date and time are displayed in Local Civil Time format based on the time-zone of the system where the CUI is running.
VOLUME Command

The VOLUME command temporarily changes your current volume setting from the startup volume of the command interpreter. The current volume is used for file name expansion within the CUI.

```
VOLUME [ system-name ]
[ [ system-name . ] volume-name ]
[ [ system-name . ] subvol-name ]
[ [ system-name . ] volume-name.subvol-name ]
```

- `system-name` is the name of the EXPAND node to use for file name expansion; the current volume and subvolume names remain unchanged.

- `volume-name` is the name of the disk volume to use for file name expansion; the current subvolume name remains unchanged.

- `subvol-name` is the name of the disk subvolume to use for file name expansion; the current volume name remains unchanged unless `volume-name` was also provided.

Considerations

If no parameter is provided, the startup volume is established as the current volume.
Notes:
3 Interacting with the CUI

This section describes the various general considerations for using the CUI. The following topics are covered in this section:

- Entering Commands
- Using Input Files
- Continuing Commands
- Issuing Multiple Commands on a Line

Entering Commands

Users interact with this CUI through a command-line-oriented interface. There are five basic methods of specifying commands to the CUI.

Interactive Commands

The principal method of interacting with the CUI is through the interactive command interface. The CUI application will display a banner, its copyright notice, the banner of the applications’ monitor process, and finally a user-prompt. The following is a sample of a CUI session startup:

```
CUICOM Sample CUI Command Interpreter D6000C30 (01NOV95)
COPYRIGHT EMPEREX CORPORATION INC. 1991 - 2003
CUIMON - D6000C20 - (01NOV95)
CUI 1)
```

The CUI 1) is the user prompt. Users can enter commands at this point. The number in the prompt generally refers to the command number as it appears in the command history buffer described earlier.

When operating in this fashion, the CUI is considered interactive.

Input File Commands

The CUI is also capable of processing commands through an input file. This file is generally specified at the TACL or COMINT command line as follows:

```
TACL) RUN CUICOM/IN cmdfile/
```

There are restrictions placed on the commands available when the CUI is run in this fashion. History commands such as FC and ! are not allowed.
Command Line Commands

An optional part of the CUI standard is the support for the placement of commands in TACL or COMINT command lines. For example:

```
TACL) RUN CUICOM ENV;SHOW
```

This command form allows you to specify CUI commands without using a command file and without invoking a user interface. The CUI standard implicitly issues the equivalent of an EXIT command after all specified commands.

TACL Variable Input

The CUI supports the input of commands from TACL variables. For example:

```
TACL) #PUSH Cmd
TACL) #APPEND Cmd ENV
TACL) RUN CUICOM/INV Cmd/
```

The semantics of processing commands through TACL variables is the same as for normal, non-interactive input files.

TACL In-line Commands

The CUI also supports the TACL in-line command facility. For example:

```
TACL) #PUSH #INLINEPREFIX
TACL) #SET #INLINEPREFIX @
TACL) RUN CUICOM/INLINE/
TACL) @ ENV
TACL) INLEOF
```

The semantics of processing input through in-line commands is the same as for normal, non-interactive input files.
Using Input Files

The CUI is capable of processing commands through a variety of different file types. The following input files are known to be supported:

- EDIT-format files (code 101)
- ENSCRIBE files (e.g., Relative, Entry-Sequenced)
- Terminals (e.g., Tandem 65xx, IBM 327x)
- Processes (including TACL requestors and variables)

If a terminal or process is used as an input file to the CUI and the same file is specified as the output file, the CUI is considered interactive, unless there are commands on the command-line.

Continuing Commands

The CUI supports command continuation. There are situations which invariably arise where commands will not fit within the record size of the input file or device. CUI provides the capability of extending commands to a maximum of 1024 bytes.

To extend a command across multiple lines, place an ampersand (&) as the last character of a line. The CUI will automatically re-prompt its input file for additional command arguments. For example:

```
CUI 1) ENV &
CUI 1)& PMSG ON
```

is equivalent to ENV PMSG ON. Note that the CUI appends an & to the user prompt when command continuation is used.

The CUI may not be able to process the FC command, however, if continued commands are too long. The supported length of the FC command varies between CUI versions.

Issuing Multiple Commands on a Line

The CUI supports the specification of multiple commands on a line. This support extends to all types of command specifications including input files. The semi-colon (;) is the separator between commands on a single line. For example:

```
CUI) ENV; SHOW; HISTORY
```

You do not need spaces before or after the ;. These are optional.
Notes:
Index

C
Command
   EXIT
   2-1
Command History Buffer
   2-2

ECHO
   2-32
ENV
   3-2 - 3-3
Environment Commands
   ! 2-3
   ?
   2-4
   ALIAS
   2-5
   ALLOW
   2-7
   COMMENT
   2-8
   DELAY
   2-9
   ECHO
   2-10
   ENV
   2-11
   ENV PMSG
   2-17
   ENV STATS
   2-18
   EXIT
   2-19
   FC
   2-20
   FILEINFO
   2-22
   FILENAMES
   2-24
   FILES
   2-26
   HELP
   2-28
   HISTORY
   2-30, 3-3
   IF
   2-31
   LOG
   2-33
   OBEY
   2-34
   OUT
   2-35
   RUN
   SETPROMPT
   2-36
   SYSTEM
   2-38
   TIME
   2-41
   VOLUME
   2-42
   EXIT
   3-2

F
FC
   2-2, 3-1, 3-3
FC Editing Template
   2-21
FILEINFO
   2-5

G
GUARDIAN
   vi

H
Help
   1-1, 2-11
HISTORY
   2-3, 3-1
   Report Format
   2-30
History Buffer
   2-4

L
log
   1-1, 2-11

O
OBEY
   2-3 - 2-4, 2-20, 2-34
OUT
   2-11

R
Run
   3-1 - 3-2
Notes: