KeyStroke Language (KSL) 9.0.00 User Guide

July 2015

Contacting BMC Software

You can access the BMC Software website at http://www.bmc.com. From this website, you can obtain information about the company, its products, corporate offices, special events, and career opportunities.

United States and Canada

Address **BMC SOFTWARE INC**
2101 CITYWEST BLVD
HOUSTON TX
77042-2827
USA

Telephone ▪ **713 918 8800** ▪ **800 841 2031** **Fax** **713 918 8000**

Outside United States and Canada

Telephone **(01) 713 918 8800** **Fax** **(01) 713 918 8000**

Restricted rights legend

Customer support

You can obtain technical support by using the BMC Software Customer Support website or by contacting Customer Support by telephone or e-mail. To expedite your inquiry, see “Before contacting BMC.”

Support website

You can obtain technical support from BMC 24 hours a day, 7 days a week at http://www.bmc.com/support. From this website, you can:

- Read overviews about support services and programs that BMC offers
- Find the most current information about BMC products
- Search a database for issues similar to yours and possible solutions
- Order or download product documentation
- Download products and maintenance
- Report an issue or ask a question
- Subscribe to receive proactive e-mail alerts when new product notices are released
- Find worldwide BMC support center locations and contact information, including e-mail addresses, fax numbers, and telephone numbers

Support by telephone or e-mail

In the United States and Canada, if you need technical support and do not have access to the web, call 800 537 1813 or send an e-mail message to customer_support@bmc.com. (In the subject line, enter **SupID:<yourSupportContractID>**, such as SupID:12345). Outside the United States and Canada, contact your local support center for assistance.

Before contacting BMC

Have the following information available so that Customer Support can begin working on your issue immediately:

- Product information
  - Product name
  - Product version (release number)
  - License number and password (trial or permanent)
- Operating system and environment information
  - Machine type
  - Operating system type, version, and service pack or other maintenance level such as PUT or PTF

  - System hardware configuration
  - Serial numbers
  - Related software (database, application, and communication) including type, version, and service pack or maintenance level
- Sequence of events leading to the issue
- Commands and options that you used
- Messages received (and the time and date that you received them)
  - Product error messages
  - Messages from the operating system, such as `file system full`
  - Messages from related software

License key and password information

If you have questions about your license key or password, contact BMC as follows:

- (USA or Canada) Contact the Order Services Password Team at 800 841 2031, or send an e-mail message to ContractsPasswordAdministration@bmc.com.
- (Europe, the Middle East, and Africa) Fax your questions to EMEA Contracts Administration at +31 20 354 8702, or send an e-mail message to password@bmc.com.
- (Asia-Pacific) Contact your BMC sales representative or your local BMC office.

Third party Software

For the provisions described in the BMC License Agreement and Order related to third party products or technologies included in the BMC Product,
see https://docs.bmc.com/docs/display/workloadautomation/Control-M+Workload+Automation+Documentation and click **Third-party software (TPS).**

Contents

Figures

Tables

About This Guide

This guide includes the following topics:

Conventions Used in This Guide

Notational conventions that may be used in this guide are explained below.

Standard Keyboard Keys

Keys that appear on the standard keyboard are identified in boldface, for example, **Enter, Shift, Ctrl+S** (a key combination), or **Ctrl S** (a key sequence).

WARNING

The commands, instructions, procedures, and syntax illustrated in this guide presume that the keyboards at your site are mapped in accordance with the EBCDIC character set. Certain special characters are referred to in this documentation, and you must ensure that your keyboard enables you to generate accurate EBCDIC hex codes. This is particularly true on keyboards that have been adapted to show local or national symbols. You should verify that

$ is mapped to x'5B'
# is mapped to x'7B'
@ is mapped to x'7C'

If you have any questions about whether your keyboard is properly mapped, contact your system administrator.

Pre configured PFKeys

Many commands are pre configured to specific keys or key combinations. This is particularly true with regard to numbered PFKeys, or pairs of numbered PFKeys. For example, the END command is pre configured to, and indicated as, **PF03/PF15**. To execute the END command, press either the **PF03** key or the **PF15** key.

Instructions to enter commands may include

- only the name of the command, such as, enter the END command
- only the PF keys, such as, press **PF03/PF15**
- or both, such as, press **PF03/PF15,** or enter the END command

Command Lines and Option Fields

Most screens contain a command line, which is primarily used to identify a single field where commands, or options, or both, are to be entered. These fields are usually designated COMMAND, but they are occasionally identified as COMMAND/OPT or COMMAND/OPTION.

Option field headings appear in many screens. These headings sometimes appear in the screen examples as OPTION, or OPT, or O.

Names of Commands, Fields, Files, Functions, Jobs, Libraries, Members, Missions, Options, Parameters, Reports, Subparameters, and Users

The names of commands, fields, functions, jobs, libraries, members, missions, options, parameters, reports, subparameters, users, and most files, are shown in standard UPPERCASE font.

User Entries

In situations where you are instructed to enter characters using the keyboard, the specific characters to be entered are shown in this **UPPERCASE BOLD** text, for example, type **EXITNAME.**

Syntax statements

In syntax, the following additional conventions apply:

- A vertical bar (|) separating items indicates that you must choose one item. In the following example, you would choose *a*, *b,* or *c*:

  ```
  a | b | c
  ```

- An ellipsis (. . .) indicates that you can repeat the preceding item or items as many times as necessary.

- Square brackets ([ ]) around an item indicate that the item is optional. If square brackets ([ ]) are around a group of items, this indicates that the item is optional, and you may choose to implement any single item in the group. Square brackets can open ([ ) and close ( ]) on the same line of text, or may begin on one line of text and end, with the choices being stacked, one or more lines later.

- Braces ({ }) around a group of items indicates that the item is mandatory, and you must choose to implement a single item in the group. Braces can open ({) and close (}) on the same line of text, or may begin on one line of text and end, with the choices being stacked, one or more lines later.

Screen Characters

`All syntax, operating system terms, and literal examples are presented in this typeface.` This includes JCL calls, code examples, control statements, and system messages. Examples of this are:

- calls, such as

```
CALL 'CBLTDLI'
```

- code examples, such as

```
FOR TABLE owner.name USE option, . . . ;
```

- control statements, such as

```
//PRDSYSIN DD * USERLOAD PRD(2) PRINT
```

- system messages, both stand-alone, such as `You are not logged on to database database_name`, and those embedded in text, such as the message `You are not logged on to database database_name`, are displayed on the screen.

Variables

Variables are identified with *italic* text. Examples of this are:

- In syntax or message text, such as
  `Specify database` *database_name*
- In regular text, such as
  replace database *database_name1* with database *database_name2* for the current session
- In a version number, such as
  EXTENDED BUFFER MANAGER for IMS 4.1.*xx*

Special elements

This book includes special elements called *notes* and *warnings*:

NOTE

Notes provide additional information about the current subject.

WARNING

Warnings alert you to situations that can cause problems, such as loss of data, if you do not follow instructions carefully.

Information New to This Version

Additional information that is new to this version is described in Appendix A of the *INCONTROL for z/OS Upgrade Guide* and What's New section of the *INCONTROL for z/OS Release Notes*.

Related Publications

INCONTROL for z/OS Installation Guide

A step-by-step guide to installing INCONTROL products using the INCONTROL™ Installation and Customization Engine (ICE) application.

INCONTROL for z/OS Administrator Guide

Information for system administrators about customizing and maintaining INCONTROL products.

INCONTROL for z/OS Utilities Guide

Describes utilities designed to perform specific administrative tasks that are available to INCONTROL products.

INCONTROL for z/OS Security Guide

A step-by-step guide to implementing security in INCONTROL products using the ICE application.

INCONTROL for z/OS Messages Manual

A comprehensive listing and explanation of all IOA and INCONTROL messages and codes.

Control-M User Guide

Guide to Control-M features, options and usage.

Control-M/Restart User Guide

A complete guide to Control-M/Restart features, options and implementation.

Control-M/Tape User Guide

Comprehensive information about the functions, features, and operation of Control-M/Tape.

Control-M/Analyzer User Guide

Explanation of Control-M/Analyzer facilities and implementation techniques. Online, step-by-step instructions are provided.

Control-D User Guide

A complete guide to using Control-D for printing and decollation of reports.

Control-D and Control-V User Guide

A complete guide to using Control-V for printing, decollation and migration of reports.

Control-O User Guide

A complete guide to Control-O features, options and implementation.

Chapter 1

Using KeyStroke Language (KSL)

The IOA standard KeyStroke Language (KSL) is a general purpose language that simulates, in batch, keystrokes that are entered in the IOA Online facility. KSL language statements (commands) are specified in programs called scripts.

The most common use of KSL scripts is to generate reports from the IOA Core and INCONTROL product repositories. Utilities are also frequently written in KSL scripts.

Once you are familiar with KSL, you can write your own scripts to create reports and utilities. Once a KSL script is defined it can be reused.

NOTE

Many of the functions performed using KSL can now be performed more easily and efficiently using CTMAPI and IOAAPI. For more information on CTMAPI refer to the *Control-M for z/OS User Guide*. For more information on IOAAPI refer to the *INCONTROL for z/OS Administrator Guide*. BMC Software recommends that you use CTMAPI and IOAAPI instead to KSL whenever possible.

Activating KeyStroke Language Scripts

The IOARKSL procedure activates KeyStroke Language scripts, either as a standard batch execution or as a started task.

KSL scripts can be activated at any time, even if the monitors are not active.

Standard batch execution example

```
//KSL      EXEC IOARKSL
//DAKSLPRM DD   *
 parameters
//
```

The following are the basic DD statements you can use with the IOARKSL procedure.

Statement	Description
//DAKSLPRM DD	The script input parameters. Record length must be 80. Columns 73 through 80 are ignored.
//DAKSLOUT DD	A listing of all invoked command members and error and execution messages. When TRACE ON is activated, it contains a listing of all executed commands and screen images of all input and output screen functions performed during script execution.
//DAKSLREP DD	Script output.
//DACALL DD	Name of the library containing KSL script members (for the CALLMEM command). Multiple libraries can be concatenated.

Started task example

```
S IOARKSL,PARM='scriptname script-parameters'
```

Return codes for the IOAKRKSL procedure

Code	Description
0	Ended OK
8	Error in input parameters
12	Severe execution error
other	Generated by the script

Principles of Operation

KSL is composed of screen control commands and editing commands. Screen control commands correspond to operations of the terminal keys. Editing commands are required to edit the printed page.

At the beginning of a script, you are positioned in the on-line field of the IOA Primary Option menu. If the IOA entry panel is mandatory at your site, you are positioned at the entry panel and you should include commands that enter your user ID and password first.

The following is an example of a KSL script that prints the contents of a specific job scheduling definition is illustrated below:

```
TYPE '2'
ENTER
TYPE 'DPTT.CTM.SCHEDULE'
CURSOR NEWLINE
TYPE 'APD'
CURSOR NEWLINE
TYPE 'APDP0020'
ENTER
PRINTSCREEN 3 23
END
```

This script produces a printout of the first screen of the job scheduling definition. The following explains each step of the above example.

Table 1 Step-by-Step Explanation of Script Example (part 1 of 2)

Command	Description
`TYPE '2'`	Equivalent to typing option 2 in the IOA Primary Option menu.
`ENTER`	Equivalent to pressing **Enter** on your terminal. As a result, you are "entering" the Online Scheduling Facility entry panel.
`TYPE 'DPTT.CTM.SCHEDULE`	On entry to the screen, the cursor is always positioned on the library name field. Type the scheduling library name.
`CURSOR NEWLINE`	The cursor moves to the table name field.
`TYPE 'APD'`	Type the table name.
`CURSOR NEWLINE`	The cursor moves to the job name field.
`TYPE 'APDP0020'`	Type the job name.

Table 1 Step-by-Step Explanation of Script Example (part 2 of 2)

Command	Description
ENTER	The job scheduling definition for the specified job is displayed in the Job Scheduling Definition screen.
PRINTSCREEN 3 23	The contents of the screen from line 3 through 23 are printed.
END	End of report.

A KSL script is a representation of your keystrokes while you are working with the Online facility. Everything that you can display on the screen, you can print. Every selection criterion that can be applied online can be applied in batch mode. The same language is used to work on the screen and on the output of the KSL script.

Modifying scripts

An important advantage of using KSL is that once a script is created, it can be stored in a member in a library. This enables requests to be submitted in batch mode as often as required (daily, weekly, monthly, and so on), and during off-peak hours not convenient for online requests.

The following example expands the previous script into a more general purpose script.

```
TYPE '2'
ENTER
TYPE 'DPTT.CTM.SCHEDULE'
CURSOR NEWLINE
TYPE 'APD'
CURSOR NEWLINE
TYPE 'APDP0020'
ENTER
LABEL PRTSCR
   *Define a label to which we can later branch from another command (GOTO).
PRINTSCREEN 3 23
CURSOR POS 23 2
   *Position the cursor on the last line of the job's data on the screen.
IFSCREEN ' ' GOTO ENDREPORT
IFSCREEN '======= >' GOTO ENDREPORT
   *If the last line of data on the screen is either blank or the end-of-data message, do
   *not print any more job data.
CURSOR HOME
   *Position the cursor on the Command field in the screen.
PF08
   *Scroll down one more page.
GOTO PRTSCR
```

```
   *Go to label PRTSCR and print the screen (loop again).
LABEL ENDREPORT
END
```

This script is easy to define, but filling in a different library, table name or job name each time you want to print a job scheduling definition is awkward.

It would be much easier if you could supply the library name, table name and job name at the time the script is executed.

Scripts can be defined with special variables (for example, *%A1*, *%A2*, described later in this chapter) instead of "hard-coded" values. When activating the script, the values for the special variables can be passed as parameters.

In the following example, special variable *%A3* represents the job name, *%A2* represents the table name, and *%A1* represents the library name. Other features, such as a header for the report produced by this script, are also presented.

```
HEADERSIZE 5
BOTTOMSIZE 1
HEADERLINE 3 1 'SCHEDULE DEFINITION OF JOB'
HEADERLINE 3 28 '%A3'
HEADERLINE 3 38 'TABLE'
HEADERLINE 3 48 '%A2'
HEADERLINE 3 58 'LIBRARY'
HEADERLINE 3 68 '%A1'
HEADERLINE 4 1 '-------------------------'
HEADERLINE 5
TYPE '2'
ENTER
TYPE '%A1'
CURSOR BTAB
CURSOR NEWLINE
TYPE '%A2'
CURSOR BTAB
CURSOR NEWLINE
TYPE '%A3'
ENTER
LABEL PRTSCR
PRINTSCREEN 3 23
CURSOR POS 23 2
IFSCREEN ' ' GOTO ENDREPORT
IFSCREEN '======= >' GOTO ENDREPORT
CURSOR HOME
PF08
GOTO PRTSCR
LABEL ENDREPORT
PF03
PF03
```

```
PF03
RETURN
```

Assume that the above script is kept in the REPJOB member. You can produce a printout of two jobs from a scheduling library by the following request:

```
//KSL EXEC IOARKSL
 CALLMEM REPJOB DPTT.CTM.SCHEDULE APD APDP0020
 CALLMEM REPJOB DPTT.CTM.SCHEDULE APD APDP0035
 END
```

Language Syntax

- A command line is processed from column 1 to 72. A command cannot exceed column 72. Columns 73 to 80 are ignored.
- A command line can contain all blanks.
- A command line with * in column 1 is considered a remark.
- Each line in a script can optionally have one continuation line. To add a continuation line, place an asterisk (*) in column 72 of the initial line.
- A KSL variable must start with the character % and can be 2 through 40 characters long. A blank designates the end of the variable name.
- KSL variables are only accessible by the KSL script in which they are defined. Any reference to the same variable in another command member (or in another invocation of the same command member) is totally independent and has no effect on the current member environment.
- The value of an AutoEdit variable applies in all command members invoked by a KSL script.
- To share information between a KSL script and other command members invoked in the same KSL run, either store the information in local AutoEdit variables, or specify the relevant information using the CALL, CALLMEM, or EXEC command.
- Values for the variables *%A1* through *%A9* (arguments) cannot be set by the SETVAR command. They can only be specified as parameters of a CALLMEM command.
- Special variables *%RC* and *%MSG* are also valid during the same invocation of a command member. Therefore, if you use the SETVAR command to assign a value to the variable *%RC* and then execute RETURN, the value of the variable is lost.

- Special AutoEdit variables and functions must start with characters %%$. They are set using command SETOLOC and are resolved according to the same rules that apply to the IOA AutoEdit facility.

- When an expression contains both KSL and special AutoEdit variables and functions, the KSL variables are resolved first.

A label is valid through the same invocation of a command member. Any reference to the same label in another command member (or in another invocation of the same command member) is totally independent and has no effect on the current member environment.

- The IOA KSL and SAMPLE libraries contain general purpose command members that can be used to solve typical report functions (for example, scroll and print).

- BMC Software recommends that you active the TRACE ON command when performing an update function with the KeyStroke Language. It is also more convenient to write new reports with the TRACE ON.

- KSL scripts may not work in a customized environment. For this reason, it is highly recommended that you run KSL using backup libraries that specify the default values for the IOA environment.

NOTE

KSL and Control-M have different AutoEdit processors. Therefore, if a KSL script containing KSL AutoEdit terms is submitted under Control-M, the Control-M AutoEdit %%RANGE statement must be used in the JCL to ensure that the Control-M AutoEdit processor skips (that is, it does not process) the KSL script.

Special considerations

Mixed case characters - The Control-M Active Environment screen (screen 3) supports mixed case (uppercase and lowercase) characters. KSL also supports mixed case characters for this screen, and product-supplied scripts have been updated accordingly.

If you are using a modified script, or a script that does not support mixed-case characters, BMC Software recommends that you change your KSL scripts to be mixed case compatible. As an alternative, you may change the format of the screen to uppercase only in the $$ACT member in the IOA MSG library, but changing the screen might affect the performance of other KSLs.

KSL and customized screens - The performance or the accuracy of the output produced by a KSL script may be affected if you have customized the IOA screens in certain ways.

For example, if you change the OWNER field in the Job Scheduling Definition screen (Screen 2), to a protected field (from its default status as an unprotected field), KSL REPCTRDF will no longer operate correctly.

KSL script authorization - If the script is to execute successfully, the user submitting a KSL script must be authorized to perform the Online functions performed by the script.

Chapter 2

KSL commands and variables

This chapter describes commands and variables used in KSL scripts.

Commands

Certain commands accept KSL and/or AutoEdit variables. When both KSL and AutoEdit variables are specified, KSL variables are resolved (replaced) first.

Table 2 KSL Screen Commands (part 1 of 2)

Command	Description
CLEAR	Equivalent to pressing the **Clear** key on the keyboard.
CURSOR	Depending on the parameters listed below, CURSOR moves the cursor to the ■ BTAB — Beginning of the previous unprotected input field on the screen. ■ HOME — First unprotected input field on the screen. ■ NEWLINE — First unprotected input field that appears on the line following the current cursor position line. ■ POS *line-no col-no* — Specific position on the screen. *line- no and col-no* can contain constants, or any valid expression consisting of KSL variables and/or AutoEdit variables. ■ TAB — Beginning of the next unprotected input field on the screen.
ENTER	Equivalent to pressing the **Enter** key on the keyboard.

Table 2 KSL Screen Commands (part 2 of 2)

Command	Description
FIND{'*textstring*' | *expression*}	Searches for text on the screen from the current position of the cursor. If the text is found, the cursor is positioned at the beginning of the text. For more information, see special variable %FINDRC, described in Table 6 on page 38. ■ *textstring* — A character string constant. When specifying constants, quotes are not necessary unless spaces are embedded in textstring. To specify a quote inside the text, use two consecutive single quotes. ■ *expression* — Any expression consisting of constants, KSL variables, and/or AutoEdit variables.
PA01-PA03	Equivalent to pressing program attention keys on the keyboard.
PF01-PF24	Equivalent to pressing program function keys on the keyboard.
SCREENSIZE *line-no col-no*	Defines the logical terminal size. Valid terminal sizes are: ■ 24 lines x 80 columns (Default) ■ 32 lines x 80 columns ■ 43 lines x 80 columns ■ 27 lines x 132 columns
TYPE{'*textstring*' | *expression}*	Operates the keyboard by "typing" the text on the screen from the current position of the cursor. If the text is too long for the current data field, an error message is produced and the script stops executing. ■ *textstring* — A character string constant. When specifying a constant, the text must be enclosed in single quotes (''). To specify a quote inside the text, use two consecutive single quotes. ■ *expression* — Any valid expression consisting of KSL variables and/or AutoEdit variables. The expression must be enclosed in single quotes ('').

Table 3 KSL Flow Commands (part 1 of 5)

Command	Description
CALL *progname* [*argument1 argument2 ... argumentn*]	Calls an external program (*progname*). The arguments are optional. A maximum of nine arguments can be passed. Use command CALL when the called program expects to receive a list of parameters. The parameters are passed in a format compatible with ASSEMBLER, COBOL and FORTRAN. ■ *progname* — Name of the called program. *progname* can consist of a constant, or may contain any valid KSL and/or AutoEdit expression. ■ *argumentx* — Any text (not containing blanks), constant, KSL variable, or AutoEdit variable to be sent to the called program. (The variable data can contain blanks.)
	Note: The return code of the called program is stored in special variable *%CALLRC*, which is described in Table 6 on page 38.
CALLMEM *memname* [*argument1 argument2 ... argumentn*]	Calls a predefined KSL script that is located in the member *memname* in the library allocated to the DACALL DD statement. The arguments are optional. A maximum of nine arguments can be passed.
	Note: The return code of the called program is stored in special variable *%CALLRC*, which is described in Table 6 on page 38.
END	Indicates the end of the KSL script. This is a mandatory command in the main script commands list. When activated at any level, the script is terminated.

Table 3 KSL Flow Commands (part 2 of 5)

Command	Description
EXEC *progname* [*argument1 argument2 ... argumentn*]	Calls an external program (*progname*). The arguments are optional. A maximum of nine arguments can be passed. Use command EXEC when the called program expects to receive an argument in a format similar to JCL's EXEC PGM argument format. (All arguments are merged into a single argument.) ■ *progname* — Name of the called program. *progname* can consist of a constant, or can contain any valid KSL and/or AutoEdit expression. ■ *arguments* — Text to be passed to the called program. An argument can consist of any text (not containing blanks), a constant, or a KSL and/or AutoEdit variable. (The variable data can contain blanks.) (Described later in this chapter).
	Note: The return code of the called program is stored in variable *%CALLRC*, which is described in Table 6 on page 38.
GOTO *label_name*	Branches to the specified label name, which must be in the same command member.

Table 3 KSL Flow Commands (part 3 of 5)

Command	Description
IFSCREEN { 'textstring' | 'expression' | COLOR col | ATTR attr | HILITE hilite | BEEP } GOTO label	If all parts of the conditional expression evaluate to true, script flow branches to the specified label name, which must be in the same command member. A parameter cannot be specified more than once within the same conditional expression. Each part of the conditional expression is true if the: ■ '*textstring*' — Text on the screen at the current cursor position is equal to the specified text. The text must be enclosed in single quotes (''). To specify a quote inside the text, use two consecutive single quotes. ■ '*expression*' — Text on the screen at the current cursor position is equal to the specified expression. expression can be any expression consisting of KSL variables and/or AutoEdit variables. expression must be enclosed in single quotes. ■ COLOR *col* — Color of the screen at the current cursor position is equal to the specified color (*col*). Valid *col* values are: — WHITE — GREEN — RED — BLUE — PINK — YELLOW — TURQUOISE — NOCOLOR

Table 3 KSL Flow Commands (part 4 of 5)

Command	Description
IFSCREEN (continued)	■ ATTR *attr* — Screen attribute at the current cursor position is equal to the specified attribute (*attr*). Valid *attr* values are: — U — Unprotected — P — Protected — L — Low — H — High — D — Dark — N — Numeric — S — Skipped — UL — Unprotected and low — UH — Unprotected and high — UD — Unprotected and dark — NL — Numeric and low — NH — Numeric and high — ND — Numeric and dark — PL — Protected and low — PH — Protected and high — PD — Protected and dark — SPL — Skipped, protected and low — SPH — Skipped, protected and high — SPD — Skipped, protected and dark ■ HILITE *hilite* — Highlight of the screen at the current cursor position is equal to the *hilite* value specified. Valid *hilite* values are: — REVERSE — Reverse video — USCORE — Underline — BLINK — Blink — NONE — No highlight ■ BEEP— Terminal beep

Table 3 KSL Flow Commands (part 5 of 5)

Command	Description
IFVAR *operand operator operand* GOTO *labname*	Where: ■ *operand* is a KSL variable or constant and/or AutoEdit variable. Constants must be enclosed in single quotes. ■ *operator* is one of the following operators. Used to compare the specified operands. EQ — is equal to NE — is not equal to GT — is greater than GE — is greater than or equal to LT — is less than LE — is less than or equal to ■ *labname* — Label name to which script branches. Specified using command LABEL (described below).
	Note: Operands are compared as character strings from left to right. For example, 91 is greater than 1000, because 9 is greater than 1. IFVAR is used together with command GOTO to permit branching based on different runtime conditions. If the condition is true, flow branches to the specified label name (must be in the same command member).
LABEL *name*	Defines a label to which script flow can branch.
MAXCOMMAND *number*	*number* is the maximum number of commands that can be executed in the script. Default: 400. This is designed to prevent an accidental loop.
PAUSE *n*	Where *n*= hundredths of seconds. Causes the script to "wait" the specified amount of time.
RETURN [*return-code*]	Returns to the calling script. *return-code* must be a number from 0 through 4095. When command RETURN is activated, control is passed to the command after the CALLMEM command in the calling member. The variable *%CALLRC* in the calling member is set to the value of the return code. Default: 0.

Table 4 KSL Print Commands (part 1 of 3)

Command	Description
BOTTOMLINE *line-no pos-in-line* {'*textstring*' | *varname*}	Assigns contents to a line in the page footer. The footer contents are valid throughout the script until overridden by another BOTTOMLINE command for the same line in overlapping positions. Command BOTTOMSIZE overrides the current BOTTOMLINE specifications. ■ *line-no* is the relative number of the line in the footer. ■ *pos-in-line* is the relative position in the line in the footer. ■ '*textstring*' must be enclosed in single quotes (''). To specify a quote inside the text, use two consecutive single quotes. ■ *varname* is a valid KSL variable.
BOTTOMSIZE *line-no*	Specifies the number of lines in the report bottom (footer) (1 minimum – 15 maximum). The bottom size is valid throughout the script until a new BOTTOMSIZE command is activated.
HEADERLINE *line-no pos-in-line* {'*textstring*' | *varname*}	Assigns contents to a line in the page header. The header contents are valid throughout the script until overridden by another HEADERLINE command for the same line in overlapping positions. Command HEADERSIZE overrides the current HEADERLINE specifications. ■ *line-no* is the relative number of the line in the header. ■ *pos-in-line* is the relative position in the line in the header. ■ '*textstring*' must be enclosed in single quotes (''). To specify a quote inside text, use two consecutive single quotes. ■ *varname* is a valid KSL variable.
HEADERSIZE *line-no*	Specifies the number of lines in the report header (1 minimum – 15 maximum). The header size is valid throughout the script until a new HEADERSIZE command is activated.

Table 4 KSL Print Commands (part 2 of 3)

Command	Description
PAGESIZE *line-no col-no*	Defines the maximum size of a printed page. ■ *line-no* is the number of lines in the page. Default: 60. ■ *col-no* is the column number. In the current version, the column number must be 132.
PRINTLINE *line-no*	Prints the contents of the line identified by *line-no* (a number from 1 through 9999).
PRINTNEWPAGE	Skips to a new page. Each occurrence of command PRINTNEWPAGE in a KSL script must be preceded by commands SCREENSIZE, PAGESIZE, HEADERSIZE and BOTTOMSIZE.
PRINTSCREEN *from-line until-line*	Prints the screen contents of the specified line range. ■ *from-line* — The first line of screen contents to be printed. ■ *until-line* — The last line of screen contents to be printed.
SETLINE *identifier pos-in-line* font {'*textstring*' | *varname*}	Assigns contents to a line that is about to be printed. ■ *identifier* is the number (from 1 through 9999) that identifies a line. ■ *pos-in-line* is the number that identifies a position in the line. ■ '*textstring*' must be enclosed in single quotes (''). To specify a quote inside the text, use two consecutive single quotes. ■ *varname* is a valid KSL variable.

Table 4 KSL Print Commands (part 3 of 3)

Command	Description
TRACE {ON|OFF}	Simplifies problem resolution using the TRACE (debug) facility while defining a script. ■ ON — Produces a complete printed output of every command execution, screen I/O, and command member invocation. It is highly recommended that KSL utilities that are performing updates of any database operate with TRACE ON to simplify problem resolution. Command TRACE can be activated any number of times within a script to turn on/off the TRACE facility. ■ OFF — Does not produce a printed output. Default.

Table 5 KSL Processing Commands (part 1 of 4)

Command	Description
ALLOC DD *ddname* DS dsname[MEM *memname*][{*SHR*|*OLD*|*MOD*}]	Dynamically allocates the data set *dsname* to the specified DD name. ■ *memname* – Member name for PDS members. Mandatory for PDS members. Must be left blank for non-PDS members. *memname* can be any valid member name consisting of a constant, KSL variable, or AutoEdit variable. ■ *ddname* – Any DD name consisting of a constant, KSL variable, or AutoEdit variable. ■ *dsname* – Any data set name consisting of a constant or KSL variable. ■ *SHR, OLD, MOD* – Specify the data set disposition. Optional. Default is *SHR*. Before a data set can be accessed (for example, with OPENFILE, GETFILE), it must be allocated and assigned a DD name. Similarly, when the data set no longer needs to be accessed, the data set and DD name must be released (for more information, see command FREE in this table). This DD name is local to the script that creates it.
CLOSEFILE *ddname*	Closes sequential data set *ddname*. *ddname* is any DD name consisting of a constant, KSL variable, or AutoEdit variable.

Table 5 KSL Processing Commands (part 2 of 4)

Command	Description
FREE DD *ddname*	Dynamically frees the data set allocated to the specified DD name. (The DD name is assigned by command ALLOC.) Activate this command when a data set no longer needs to be accessed by the script.
GETFILE *ddname* INTO *varname*	Stores in the specified variable the contents of the next record in the sequential file referenced by *ddname*, where ■ *ddname* is any DD name consisting of a constant, KSL variable, or AutoEdit variable. ■ *varname* is a KSL variable name where contents are stored.
OPENFILE ddname { INPUT / OUTPUT / UPDATE } [ENDFILE label]	Opens sequential data set *ddname* for access. ■ *ddname* is any DD name consisting of a constant, KSL variable, or AutoEdit variable. ■ INPUT opens the sequential data set as a read-only file. No changes can be made to the file. ■ OUTPUT allows data to be written to the file (write access). ■ UPDATE allows the file to be read and modified (read and write access). Default. ■ ENDFILE *label* specifies a label name to which script processing flow branches when the end of sequential data set *ddname* is reached. ENDFILE is mandatory when *ddname* is opened for INPUT or UPDATE access.
PUTFILE *ddname* FROM *varname*	Writes the contents of variable *varname* in the next record of the sequential file referenced by *ddname*. ■ *ddname* is any DD name consisting of a constant, KSL variable, and/or AutoEdit variables. ■ *varname* specifies a KSL variable whose contents are written.
	Note: AutoEdit variables (that is, variables beginning %%) cannot be used with a PUTFILE command. Instead, such a variable must be passed to a KSL variable (that is, a variable beginning with a single %). The KSL variable must be specified in the PUTFILE command.

Table 5 KSL Processing Commands (part 3 of 4)

Command	Description
SETOGLB { %%var = value %%var = expression %%autoedit - control - statement }	Assigns the appropriate value to the variable name. This command is used to create (set) a global AutoEdit variable. *expression* may contain a KSL variable.
	Note: This command is only available if Control-O is installed at your site. Global AutoEdit variables can only be used when the Control-O monitor is online. For further information, see the DO SET statement in the *Control-O User Guide*.
SETOLOC { %%var = value %%var = expression %%autoedit - control - statement }	Assigns the value of the expression to the variable name. This command is used to create (set) a Local AutoEdit variable. expression may contain a KSL variable.
SETVAR *varname* CURSOR *length*	Assigns extracted text to variable *varname*. This command is used to create (set) variables. ■ *varname* is the name of the KSL variable in which the text is stored. ■ *length* is the length (in characters) of the extracted text. The text at the current cursor position of the specified length is extracted and assigned to the variable name.
SETVAR *varname* DATA{'*textstring*' | *expression*}	Assigns a text string to variable *varname*. This command is used to create (set) variables. ■ *varname* is the name of the KSL variable in which the text is stored. ■ '*textstring*' is the textstring assigned to the variable name. *textstring* is a character string constant. When specifying a constant, the text must be enclosed in single quotes (''). To specify a quote in the text, type two consecutive single quotes. ■ *expression* is any valid expression containing KSL variables and/or AutoEdit symbols. *expression* must be enclosed in single quotes if it contains embedded blanks.

Table 5 KSL Processing Commands (part 4 of 4)

Command	Description
SETVAR *varname* SCREEN *from-line from-col length*	Assigns extracted text to variable *varname*. This command is used to create (set) variables. ■ *varname* is the name of KSL variable in which the text is stored. Variable value is determined by the screen contents at a specific screen position. The screen position is specified by: ■ *from-line* – Starting from this line position, screen contents are extracted. ■ *from-col* – Starting from this column position, screen contents are extracted. ■ *length* – The number of characters) in the extracted text.
SHOUT TO *destination* [URGENCY *urgency*] TEXT {*textstring*|*expression*}	Sends ("shouts") textstring to the specified destination. ■ destination – a 1 through 16 character destination. For valid destination values, refer to the description of the SHOUT parameter in the *Control-M for z/OS User Guide*.

Variables

KSL variables can be used to add flexibility to a KSL script. These variables are assigned using a KSL command (such as SETVAR) and are resolved during the run of the KSL script.

A KSL variable must start with % and can be 2 through 40 characters long. A blank designates the end of the variable name.

NOTE

The second character in a KSL variable name must not be a percent sign. KSL assumes that a variable beginning with %% is an AutoEdit variable.

If a KSL script is to search for a prerequisite condition whose name begins with a single percent sign (%), KSL assumes it is a KSL user-defined variable and does not recognize the searched-for condition.

KSL variables are only accessible by the KSL script in which they are defined. A reference to the same variable in another command member (or in another invocation of the same command member) is totally independent and has no effect on the current member environment.

When an expression contains both KSL and special AutoEdit variables and functions, KSL variables are resolved first.

For more information about syntax and KSL variables, see "Language Syntax" on page 22.

Special KSL Variables

Some KSL variables are reserved by, and have a special meaning for, KSL:

Table 6 Special KSL Variables

Variable	Description
%A1-%A9	Passes the specified arguments to a called script. The number corresponds to the position of the argument in command CALLMEM. The argument is replaced throughout the called script member at invocation time.
%CALLRC	Contains the return code specified in the RETURN command when returning from command CALLMEM. Also contains the return code from programs called by the CALL or EXEC commands.
%FINDRC	Provides the return code of the result of the last FIND. If the last FIND was successful, has a value of 0. If the last FIND was unsuccessful, *%FINDRC* has a value of 4.
%MSG	Specifies text assigned at script termination, which appears as a message in the job's SYSLOG and the script execution listing. Only the value of variable *%MSG* at the script member issuing command END is displayed.
%RC	Supplies the return code of the script. The value at successful script termination is the condition code of the step. Valid values are: 0 through 4095.
%SCRCOL	Current column position of the cursor.
%SCRLINE	Current line position of the cursor.

Chapter 3

KeyStroke OpenAccess (KOA)

The KeyStroke OpenAccess facility (KOA) permits two-way communication between Control-O and any VTAM application. Examples of VTAM applications are performance monitors, online systems, and so on. Information can be collected from these applications and used as part of the Control-O decision process. Commands under these applications can also be triggered based on Control-O rules. The possible uses of KOA include:

- Acquiring performance data from online performance monitors (for example, OMEGAMON or MV Manager for INCONTROL) and reacting to the acquired data.

- Logging on to problem management systems to open or update problem tickets.
- Logging on to electronic mail systems and sending alert messages to Customer Support.
- Logging on to online communication systems (for example, CICS, IMS/DC, IDMS/DC, COM-PLETE) to verify the availability and serviceability of online systems and their applications.
- Logging on to the IOA Online facility (using the IOA VTAM monitor).
- Logging on to online communication systems to activate facilities.
- Logging on to VM, as well as its operator ID.
- Logging on to network management products (such as NetView).

An open interface to VTAM applications is an integral part of Control-O as a console automation product. The KeyStroke OpenAccess facility uses the same syntax and the same concepts as the standard IOA KeyStroke Language (KSL). Therefore, the same language and syntax can be used both for accessing VTAM applications and for reporting. The KeyStroke OpenAccess facility simulates the keystrokes of a user working in VTAM application screens the same way KSL does for users working in IOA screens. Extended AutoEdit capabilities allow communication between Control-O/KOA and VTAM applications.

KOA Language Overview

The KOA language commands form a "superset" (extended version) of the KSL language commands. All KSL commands and parameters are included in the KOA language, but in addition, KOA includes communication commands that provide and control access to VTAM applications.

KOA syntax is the same as the syntax for KSL, which is described earlier in this chapter. KOA commands are described in detail later in this chapter.

KOA Logic

KOA commands are combined and stored in scripts. When the script is activated, KOA processes the commands listed in the script. A typical script initiates a session with a VTAM application, and includes the following basic steps:

Session Logon

KOA runs as a VTAM application, which emulates a terminal (according to specified parameters) and establishes a session with the VTAM application through the LOGON command.

Session Processing

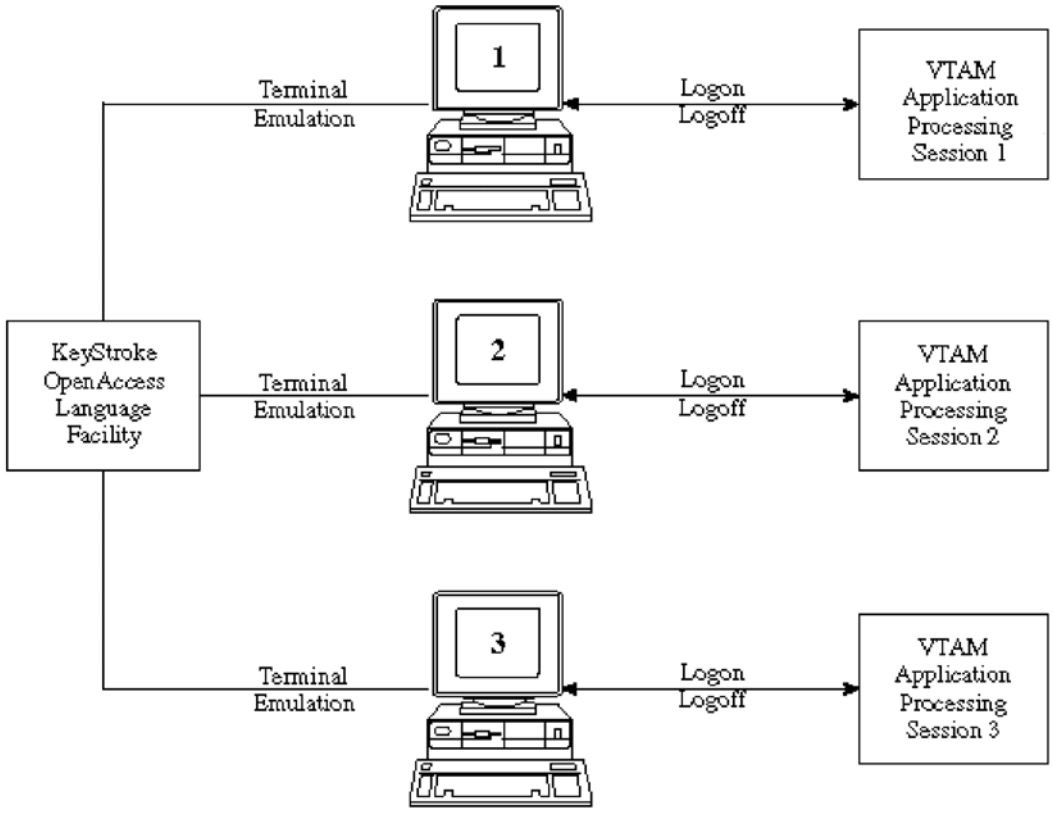
The script runs, performing tasks – such as extracting and saving data from an online application; running reports; and sending ("shouting") messages to various destinations.

Session Logoff

The VTAM application session is disconnected through the LOGOFF command.

With KOA, several VTAM application sessions can be active at the same time. A unique session identifier is assigned to each session, so that a KOA script can switch back and forth between active sessions.

Figure 1 VTAM Session LOGOFF



Two-way communication is established and maintained throughout a VTAM application session. An implied "send and receive" relationship forms the basis of communication – for every keyboard input that is sent by KOA, a matching response is received from the VTAM application. For example, KOA transmits (sends) the PF01 key command, and the VTAM application responds by displaying help information.

This two-way communication represents the default method of exchanging information between KOA and VTAM. The KOA language also contains commands that allow you to change this method. For example, updated screens from the VTAM application can be received repeatedly without KOA first sending a request.

For information about creating and running scripts, see "Activating the KOA Language" on page 48.

Principles of Operation

KOA scripts can be generated automatically by the KOA Recorder, which is described in "KOA Recording" on page 72, or manually, as described below.

To manually write a script, imagine yourself in front of a terminal that is connected to VTAM only – for example, the opening screen that you receive when your terminal is switched on. You are not yet connected to any specific application. You must then log on to an application.

To explain how the script language works, we will use simplified examples of KOA scripts.

Example 1

The following KOA script logs on to CICS and prints all open data sets.

Figure 2 KOA Script – Example 1

```
LOGON APPLID CICSP SESSID CICP
LABEL CHECK
CURSOR POS 1 15
IFSCREEN `WELCOME TO CICS/MVS' GOTO CONTINUE
GETSCREEN
GOTO CHECK
LABEL CONTINUE
CLEAR
TYPE `CEMT INQUIRE DATASET(*) OPEN'
ENTER
PRINTSCREEN 1 24
PF03
CLEAR
TYPE `CSSF LOGOFF'
ENTER
LOGOFF
END
```

A detailed explanation of each step in the above sample script is shown below:

Table 7 KOA Script – Example 1 Explanation (part 1 of 2)

Step	Description
`LOGON APPLID CICSP SESSID CICP`	Issue a command to log on to application CICSP, assigning session name CICP to this session.
`LABEL CHECK`	Specify label CHECK for script flow branching purposes. This reference point marks the portion of the script that checks if the logon to CICS is successful.

Table 7 KOA Script – Example 1 Explanation (part 2 of 2)

Step	Description
`CURSOR POS 1 15`	Position the screen cursor at row 1, column 15. If the logon was successful, CICS's welcome message is displayed at this screen position.
`IFSCREEN 'WELCOME TO CICS/MVS' GOTO CONTINUE`	Determine if the CICS/MVS welcome message is currently displayed. If displayed, proceed to label CONTINUE. If not, proceed to the next script command.
`GETSCREEN`	Receive (accept) any available messages. This is a test to see if the CICS welcome message has been sent.
`GOTO CHECK`	Proceed to label CHECK to continue testing for the CICS welcome message.
`LABEL CONTINUE`	Specify label CONTINUE for script branching purposes. Script flow continues here when the logon to CICS is successful.
`CLEAR`	Clear the screen.
`TYPE 'CEMT INQUIRE DATASET(*) OPEN'`	Type the specified command at the keyboard. This command displays all open data sets.
`ENTER`	Send the screen to CICS, and in response, receive the CEMT screen back from CICS.
`PRINTSCREEN 1 24`	Print the contents of the screen from line 1 through line 24.
`PF03`	Equivalent to pressing the PF03 key, which deactivates the CEMT command.
`CLEAR`	Clear the screen.
`TYPE 'CSSF LOGOFF'`	Issue CICS's logoff command.
`ENTER`	Equivalent to pressing **Enter** on the terminal keyboard. You have now logged off from CICS.
`LOGOFF`	Issue KOA's logoff command.
`END`	Terminate the script.

As you can see, the KOA script is actually a representation of the keystrokes that are entered while working with a VTAM application. Everything that can be seen on the screen can be printed; and any keystroke that can be performed on a terminal can be performed by KOA.

Example 2

The following KOA script is activated by Control-O when a console message indicates that a production job that needs exclusive access is waiting for a reserved data set. The script looks for all users (TSO users and jobs) currently using the data set. If a TSO user is found, a message is sent to the user. Otherwise, a message is sent to the production manager.

Assume that the rule that activates this script passes two arguments to the script:

Table 8 KOA Script – Example 2 Explanation (part 1 of 4)

Step	Description
%A1	Data set needed by a production job (%A2), currently reserved by other users and jobs.
%A2	Job which needs exclusive access to a data set (%A1).
ON SCREENERROR GOTO END	If an exception occurs at any time during script execution, proceed to label END, which terminates the script.
LOGON APPLID OMEGAMON SESSID OM	Log on to application OMEGAMON, assigning session name OM to this session.
ENTER	Equivalent to pressing **Enter** on the keyboard. In this situation, **Enter** bypasses the OMEGAMON opening screen.
CURSOR POS 1 2	Position the cursor on the screen.
TYPE 'ZOPTEA'	Type OMEGAMON command ZOPTEA, which lists display options on the screen.
ENTER	Send the screen to OMEGAMON, and in response, receive a screen from OMEGAMON.
CURSOR POS 14 24	Position the cursor.
TYPE 'CSR'	Type OMEGAMON command CSR to change the scroll amount from page to cursor. Changing the scroll amount to CSR allows the script to scroll from a specific cursor position, instead of screen by screen.
ENTER	Equivalent to pressing **Enter** on the keyboard, thus sending the new scroll amount.
CURSOR POS 2 1	Position the cursor.

Table 8 KOA Script – Example 2 Explanation (part 2 of 4)

Step	Description
`TYPE 'LOC %A1'`	Type OMEGAMON command LOC, which locates data set %A1 and lists the users and jobs which are currently accessing that data set (in either Shared or Exclusive mode).
`ENTER`	Send the previous command, to display the user list.
`CURSOR POS 3 1`	Position the cursor.
`IFSCREEN '+' GOTO END`	If the value at the cursor position is +, this indicates that message: `+ DSNAME _____ NOT CURRENTLY ALLOCATED` is displayed. In this situation, the data set is available, and the script should proceed to label END. If the data set is allocated (the message is not displayed), proceed to the next script command.
`CURSOR POS 4 1`	Position the cursor.
`PF08`	Equivalent to pressing PFKey PF08 on the keyboard. Pressing PF08 when scroll amount CSR is specified shifts the list of users up, so that the current cursor line is displayed at the top of the screen.
`LABEL LOOP`	Specify label LOOP to mark the portion of the script that determines if any more data set users exist.
`CURSOR POS 2 1`	Position the cursor.
`SETVAR %START_OF_LINE` `CURSOR 1`	Save the current screen contents (one character in length) to KOA variable %START_OF_LINE. Note that if a user exists, it is preceded by the symbol > on the screen.

Table 8 KOA Script – Example 2 Explanation (part 3 of 4)

Step	Description
`IFVAR %START_OF_LINE NE '>' GOTO END`	If the value of variable %START_OF_LINE is not equal to >, indicating that no more users are accessing the data set, proceed to label END (because there are no more users to notify). If %START_OF_LINE is equal to >, messages must be sent to the appropriate users.
`LABEL TSO-USER`	Specify label TSO-USER. This chapter of the script determines if the user of the data set is a TSO user.
`SETVAR %TYPE SCREEN 2 26 3`	Store screen contents (of the specified position) in KOA variable %TYPE. If the user is a TSO user, the current screen contents contain TSO.
`IFVAR %TYPE NE `TSO' GOTO BAT-JOB`	If the value of variable %TYPE indicates that the user is not a TSO user, proceed to label BAT-JOB. If this user is a TSO user, proceed to the next script command.
`SETVAR %USERNAME SCREEN 2 39 8`	Store the TSO user name in KOA variable %USERNAME.
`SHOUT TO TSO-%USERNAME MESSAGE 'PLEASE FREE FILE %A1 IMMEDIATELY'`	Send a message (using the Shout facility) to the TSO user %USERNAME. This message asks the user to free data set %A1 immediately.
`SHOUT TO TSO-%USERNAME MESSAGE 'PRODUCTION JOB %A2 IS WAITING FOR IT.'`	Inform TSO user %USERNAME that %A1 must be freed so job %A2 can access the data set.
`GOTO CONTINUE`	Proceed to label CONTINUE.
`LABEL BAT-JOB`	Specify label BAT-JOB to mark the portion of the script that sends messages to the production manager if the user currently accessing the data set is a batch job.
`SETVAR %TYPE SCREEN 2 26 3`	Save the user type in variable %TYPE.
`IFVAR %TYPE NE 'BAT' GOTO CONTINUE`	If the user is not a batch job, proceed to label CONTINUE. If the user is a batch job, process the following script commands.
`SETVAR %JOBNAME SCREEN 2 39 8`	Save the batch job name in variable %JOBNAME.

Table 8 KOA Script – Example 2 Explanation (part 4 of 4)

Step	Description
`SHOUT TO U-PRODMNGR MESSAGE 'CTO -PRODUCTION JOB %A2 WAITING FOR FILES'`	Send (shout) a message to U-PRODMNGR, informing that job %A2 is waiting for data set access.
`SHOUT TO U-PRODMNGR MESSAGE 'CTO -FILE %A1 IS HELD BY JOB %JOBNAME'`	Inform U-PRODMNGR (through the Shout facility) that data set %A1 is accessed by job %JOBNAME.
`LABEL CONTINUE`	Specify label CONTINUE to mark the portion of the script that repositions the cursor to the next user in the user list.
`CURSOR POS 3 1`	Position the cursor.
`PF08`	Redisplay the user list starting with the next line.
`GOTO LOOP`	Proceed to label LOOP to process other users holding the data set.
`LABEL END`	Specify label END as a reference, marking the end of the script.
`LOGOFF`	Disconnect from VTAM application OMEGAMON.
`END`	Terminate the script.

Because this script uses parameters (`%A1` and `%A2`) instead of literal data set and job names, the script is easy to invoke—just specify the appropriate information each time the script is activated from within a rule.

Sample KOA Scripts and Reports

The IOA SAMPLE library contains examples of KOA scripts and KSL reports. Each script and report invokes a few CALLMEM members to perform its functions.

Additional KOA samples are located in the Control-O SolveWare libraries, and described in the *Control-O SolveWare Reference Guide*.

Activating the KOA Language

The following methods are available for activating KOA scripts:

Activate the script within a Control-O rule.

Example

```
ON COMMAND ...
DO KSL=scriptname %%x %%y %%z
DO ...
```

In the above example, Control-O continues to process the rest of the rule while the script is executing. This is the default method of script processing when activated by a rule. The example below processes the script in a different manner.

```
DO SET=%%WAITKSL = YES
DO KSL=scriptname %%x %%y %%z
   WAITMODE x
DO IF %%KSLRC EQ 0
       DO ...
ENDIF
```

Activate the script as a batch job. The script can also be activated under Control-M. The same method is used to run KSL reports.

Procedure IOARKOA is used to activate the KOA script. Important DD statements are:

Table 9 IOARKOA Important DD Statements

DD Statement	Description
//DAKSLPRM DD	Script parameters (input) – record length must be 80. (Columns 73-80 are ignored.)
//DAKSLOUT DD	A listing of all invoked command members, and error/execution messages. When TRACE ON is activated, it contains a listing of all executed commands and screen images of all the input/output screen functions performed during the script execution.
//DAREPORT DD	A report, when created by KOA.
//DACALL DD	PDS name containing script members (for command CALLMEM). Multiple libraries can be concatenated.

Activate the script as a started task. Pass the script name and script parameters as procedure parameters, using the following syntax:

```
S IOARKOA,PARM=`TRANID=KOA,scriptname script-parameters'
```

KOA Commands and Variables Summary

A brief summary of the KOA commands and variables is shown below. These commands are in addition to the KSL commands discussed elsewhere in this chapter.

Table 10 KOA Screen Commands

Command	Description
ATTN	Equivalent to pressing the **Attn** key on the keyboard.
CLEAR	Equivalent to pressing the **Clear** key on the keyboard.

Table 11 KOA Flow Commands

Command	Description
ON SCREENERROR	Branches to a labeled script command when a communication exception condition exists.

Table 12 KOA Communication Commands

Command	Description
COLOR	Sets the terminal's color capabilities.
GETSCREEN	Receives updated information (screens and messages) from the VTAM application.
LOGON	Logs on to a VTAM application session.
LOGOFF	Logs off (disconnects) from a VTAM application session.
SCREENMODE	Determines which screen types are received from VTAM.
SETSESS	Switches to the specified session.
TIMEOUT	Limits the number of seconds KOA waits for a response before disconnecting the session.

Table 13 KOA Special Variables

Variable	Description
%SESSID	Current session identifier.
%VTAMERR	VTAM communication error description.
%VTAMFDBK	VTAM communication error information.
%VTAMRC	VTAM communication return code.

KOA Commands and Variables

The KOA commands and variables are described in greater detail below.

Braces ({ }) indicate that one of the items listed between the braces must be specified. Square brackets ([ ]) denote optionality; none or several of the items listed between the brackets can be specified.

Certain commands accept KOA and/or AutoEdit variables. When both KOA and AutoEdit variables are specified, KOA variables are resolved (replaced) first.

KOA Commands

Table 14 KOA Screen Commands

Command	Description
ATTN	Equivalent to pressing the **Attn** key on the keyboard.
CLEAR	Equivalent to pressing the **Clear** key on the keyboard.

Table 15 KOA Flow Commands

Command	Description
ON SCREENERROR [GOTO label]	Permits branching based on communication exception conditions. An example of an exception is a VTAM application that does not accept a logon command. This command can be activated any number of times within a script to change the method of exception handling. Each time command ON SCREENERROR is activated, the previous setting is overridden. The new setting takes effect from the point it is specified in the script onward.
GOTO	The script flow branches to the specified label name (must be in the same command member). If no GOTO parameter is specified, no branching occurs.

Table 16 KOA Communication Commands (part 1 of 4)

Command	Description
COLOR {ON|OFF}	Specifies color and highlight capabilities of terminal emulation. If LOGMODE is not specified when logging in to the VTAM application, KOA uses this command to determine the terminal emulation type. For more information see "KOA Implementation Considerations" on page 55. ON – Emulates a terminal which supports extended data stream (colors and highlights). OFF – Emulates a terminal which does not support extended data stream (colors and highlights). Default.
GETSCREEN	Instructs KOA to refresh the screen with updated information from the VTAM application, without waiting for keyboard input.
LOGON APPLID applid SESSID sessid [DATA data | LOGMODE logmode | TERMINAL terminal]	

Table 16 KOA Communication Commands (part 2 of 4)

Command	Description
	Logs on to a VTAM application. The APPLID and SESSID parameters are mandatory. Other optional parameters (listed below) can be specified to define terminal name and emulation type. ■ APPLID *applid* – Specifies the name of the VTAM application. applid can consist of a constant or can contain any valid KOA or AutoEdit expression. Mandatory. ■ SESSID *sessid* – Specifies the 4-character, unique identifier of the session. sessid is supplied by the user. Within the same KOA script, a user can operate several active VTAM application sessions concurrently. Therefore, each session must have its own unique session ID. Mandatory. ■ DATA *data* – Passes data to the VTAM application as part of the logon process (such as TSO user ID). If *data* contains embedded blanks, it must be enclosed in single quotes. *data* can consist of a constant, or can contain any valid KOA and/or AutoEdit expression. Optional. ■ LOGMODE *logmode* – Defines terminal characteristics (such as size or color) by specifying a predefined VTAM *logmode* name. The *logmode* name must be 1-8 characters in length. Optional. If LOGMODE is not specified, KOA uses information provided in commands COLOR and SCREENSIZE to find a valid predefined *logmode* in CTOPARM which supports these characteristics. For more information refer to "KOA Implementation Considerations" later in this chapter. ■ TERMINAL *terminal* – Specifies a predefined VTAM terminal (logical unit) name. The terminal name must be 1-8 characters in length. Optional. If TERMINAL is not specified, KOA uses information provided in member CTOPARM to dynamically select a terminal name. For more information refer to "KOA Implementation Considerations" later in this chapter.

Table 16 KOA Communication Commands (part 3 of 4)

Command	Description
LOGOFF	Logs off or disconnects from the VTAM application.
SCREENMODE {UNLOCKED | ANY}	Indicates type of screens to be received by the KOA script. This command can be activated any number of times within a script to re-specify the screen mode. ■ UNLOCKED – Only unlocked screens (screens that allow keyboard input) are received. This parameter ensures that KOA is always able to send information back to the VTAM application. Default. ■ ANY – Any screen (locked or unlocked) is received. KOA may receive a screen which does not allow keyboard input. However, KOA will not be able to respond to this type of screen.
SCREENMODE {RECEIVE | NORECEIVE }	Indicates if information (messages and screens) should be received from the VTAM application. This command can be activated any number of times within a script to re-specify the screen mode. ■ RECEIVE – information is sent to and received from the VTAM application. Each time KOA sends information to the VTAM application, KOA waits for a response. Default. ■ NORECEIVE – Information is sent to the VTAM application, but no response is received automatically. To request a response in this mode, command GETSCREEN should be used.
SCREENMODE {GETUNSOL | IGNUNSOL}	Indicates type of information (messages and screens) to be received from the VTAM application. This command can be activated any number of times within a script to respecify the screen mode. ■ GETUNSOL – Both solicited and unsolicited messages and screens are received from the VTAM application and can be displayed (by command GETSCREEN). ■ IGNUNSOL – Unsolicited information (that is, messages and screens which are not expected by KOA) are ignored, and are not displayed. Default.
SETSESS *sessid*	Switches (shifts) to the session with the specified session ID.

Table 16 KOA Communication Commands (part 4 of 4)

Command	Description
TIMEOUT *value*	Specifies how many seconds KOA should wait for a response from the VTAM application before disconnecting the session. If not specified, the default value (as specified in the Control-O Installation Parameters) determines the TIMEOUT value.

KOA Special Variables

Table 17 KOA Special Variables

Variable	Description
%SESSID	ID of the current session.
%VTAMERR	Describes briefly the VTAM communication return code %VTAMRC (see below). For a list of VTAM session return codes see Appendix B, "KOA VTAM Exception Codes."
%VTAMFDBK	Specifies VTAM communication-specific error information. For a list of VTAM communication errors see Appendix B, "KOA VTAM Exception Codes."
%VTAMRC	Specifies the VTAM communication return code. A value of 0 indicates a successful operation. For a list of VTAM session return codes see Appendix B, "KOA VTAM Exception Codes."

KOA Implementation Considerations

VTAM application programs (for example, TSO, CICS, IMS/DC, OMEGAMON) communicate using Logical Units. A Logical Unit (LU) is any item that is used to send information to, and receive information from, VTAM application programs. Standard examples of LUs are terminals and terminal emulation programs. The KOA facility initiates VTAM application sessions and communicates with VTAM applications as if it were a terminal. Therefore, KOA can perform any action that a terminal can perform.

For information on installing the KOA facility, see the *INCONTROL for z/OS Installation Guide*.

Session Characteristics

When a Logical Unit (LU) starts a session (LOGON), it provides a set of parameters that identify the LU to the VTAM application. The parameters include screen size; ability to display extended data stream (such as color or highlights); and other parameters that determine session protocol. Parameters describing the screen in a VTAM session are grouped into a LOGMODE entry. LOGMODE entries are, in turn, stored in LOGMODE tables.

KOA emulates different types of terminals according to user specifications. The KOA language provides commands such as SCREENSIZE and COLOR for automatic selection of LOGMODE entries from a predefined list (which is created during KOA installation). Four different terminal sizes are currently available, and each size may or may not support extended data stream.

You can override the KOA-selected LOGMODE by specifying the LOGMODE parameter of command LOGON. In this case, LOGMODE (and not the commands SCREENSIZE and COLOR) determines terminal characteristics.

Certain online systems (such as IMS/DC and, optionally, CICS) require that LUs be predefined, and that they be stored in internal tables. Each definition in the table contains the name and characteristic for that LU. When logging on to these online systems, the LOGMODE information is ignored and the predefined values are used. In this case, the SCREENSIZE, COLOR and LOGMODE commands and parameters have no effect.

Initiating a Session

A session between a terminal and a VTAM application is initiated by logging on to the application and specifying the name (as defined to VTAM) of the requested VTAM application. KOA command LOGON performs exactly the same function. In addition, for each session that is initiated, KOA selects a terminal name (which is used during communication with the VTAM application). This terminal name is dynamically selected from a predefined pool of terminal names as defined in member CTOPARM.

The TERMINAL parameter of command LOGON can be used to force a specific terminal name for the session. This may be useful when the VTAM application performs security checks based on terminal name.

As part of the logon process, the LU can pass information (messages) to the VTAM application. For example, the user ID can be passed to TSO. The DATA parameter of command LOGON performs this function.

NOTE

When a session is initiated, the VTAM application usually sends the first message (for example, a welcome message or a sign-on screen). Therefore, after a session is initiated, KOA expects to receive a message from the application. However, there are applications that expect the first message to come from the terminal (LU) instead. For these cases, the command SCREENMODE NORECEIVE should be specified before command LOGON, and KOA should be the first to send a message after the logon process is completed.

As part of session initiation, the VTAM application and the LU may exchange certain information before the first screen is sent. For example, the application may require the specification of physical terminal characteristics (for example, screen size or screen display data type) before sending the first screen. Although this has no effect on a user working with a standard terminal (because this message switching is very quick), the KOA script must provide for these situations.

Most cases are handled automatically, and KOA receives the first screen immediately following command LOGON. However, in certain situations the user receives an "empty" screen (or several "empty" screens) before the expected first screen. Scripts must be designed to handle these situations.

Example

The following sample script logs on to CICS. This script is designed to handle the receipt of empty screens.

Table 18 KOA Sample Script to Log on to CICS (part 1 of 2)

Step	Description
`LOGON APPLID CICSPROD SESSID CICS`	Log on to application CICSPROD, and name the session CICS.
`LABEL FIRSTSCR`	Specify label FIRSTSCR for script flow branching purposes.
`CURSOR POS 1 5`	Position the cursor at line 1, column position 5.
`IFSCREEN `WELCOME TO CICS' GOTO CONTINUE`	Determine if the CICS welcome message is currently displayed. If displayed, proceed to label CONTINUE. If not, proceed to the next script command.
`GETSCREEN`	Assume that an "empty" screen has been received, and request the next message or screen from the VTAM application.

Table 18 KOA Sample Script to Log on to CICS (part 2 of 2)

Step	Description
`GOTO FIRSTSCR`	Branch back to label FIRSTSCR to test the current screen until the CICS welcome message is displayed.
`LABEL CONTINUE`	Specify label CONTINUE for script flow branching purposes.

Exchanging Messages

KOA expects to receive a screen (information) back from a VTAM application each time a screen is sent to the application. This implied send and receive method of exchanging information is the default method of communication between KOA and VTAM. This means that whenever a screen is sent to the VTAM application (such as through ENTER, ATTN, CLEAR, or PFxx), KOA automatically performs a receive operation in order to receive an updated screen from the application. (Note that the same process is automatically performed after the logon process completes.) From the point of view of the script, the current screen is the application's response to the previously sent screen.

To eliminate the implied receive and override the default method, command SCREENMODE NORECEIVE should be used (the default is SCREENMODE RECEIVE). In NORECEIVE mode, KOA does not wait for responses from the VTAM application, and the script continues processing. When operating in NORECEIVE mode, KOA command GETSCREEN should be specified in order to receive a message from the VTAM application.

The KOA default method of exchanging information ensures that, at any stage, the script operates on a screen that is ready to receive input (that is, the VTAM application is waiting for input). For example, when a user communicates with an application, the keyboard is always unlocked (reset) so the user can enter data. KOA scripts must operate under similar conditions. To guarantee these conditions, KOA repeatedly receives messages from the application – until a message is received indicating that the application is now waiting for input from the LU.

Some situations require a different method of exchanging information. For example, a KOA script can determine independently if or when to send data to – or request additional messages from – the VTAM application. This method may be required for situations where the application sends messages at fixed intervals, without first unlocking the screen. To choose this mode of information exchange, command SCREENMODE ANY should be used (the default is SCREENMODE UNLOCKED). More complex KOA scripts must be written to facilitate this method of exchanging information.

When the current KOA SCREENMODE is ANY, a message may not be "sendable" to the application because the screen (keyboard) is locked. A user at a terminal, in this situation, simply presses the attention key and waits until the keyboard is unlocked. In KOA, command ATTN sends an attention request to the application, and receives an unlocked screen from the application.

Terminating a Session

Two different methods exist for terminating a session between a user's terminal and a VTAM application:

Application logoff

Issue a (VTAM application) command to terminate the session. For example, /RCL (in IMS/DC), CESF LOGOFF (in CICS), and LOGOFF (in TSO) instruct the application to terminate the session, and free the terminal for other usage. This is the proper method for requesting session termination from the VTAM application.

VTAM logoff

Use the keyboard SysRq (System Request) function or simply switch off the terminal. This will probably free the terminal for other usage, but might not close the application session properly. For example, a TSO session will not be terminated, but is available for a reconnect from another terminal.

KOA supports both methods of session termination. It is recommended that both methods be specified within the script. KOA scripts should terminate the session with an 'application logoff' and then use the KOA command LOGOFF to perform a `VTAM logoff' which releases the terminal for use.

For example, to terminate a CICS session:

Table 19 KOA Sample Script to Terminate a CICS Session

Command	Explanation
`SCREENMODE NORECEIVE`	Change the screen emulation to NORECEIVE.
`TYPE `CESF LOGOFF'`	Type CICS command LOGOFF at the keyboard.
`ENTER`	Send the command to CICS.
`LOGOFF`	Issue the KOA LOGOFF command to disconnect the session.

When a session is terminated in an orderly way, the application usually sends an "end-of-session" message (such as when TSO sends an IKJ56470I message) and then releases the terminal. Using the default method of information exchange (implied send/receive), KOA receives that message, and supplies a return code to the script that indicates that the session is no longer active. The script can either handle that return code while it is terminating the session, or ignore (not receive) the last message from the application.

For example, terminate a TSO session:

Table 20 KOA Sample Script to Terminate a TSO Session

Command	Explanation
`TYPE `LOGOFF'`	Type the TSO LOGOFF command at the keyboard.
`ENTER`	Send the command to TSO.
`IFVAR %VTAMERR NE `SESS-NOT-ACT' GOTO CHKERR`	Verify that the TSO session is not active. If it is still active, branch to label CHKERR (not included in this sample) for error processing. If the TSO session is not active, proceed to the next script command.
`LOGOFF`	Issue the KOA LOGOFF command to disconnect the session.

Unexpected Messages

During a session between a VTAM application and a terminal, most received messages are expected. For example, a user knows that pressing PF01 displays a Help screen. However, there are situations where unexpected messages (unsolicited messages) are displayed on the terminal.

An example of an unexpected message is a message sent by the operator to a TSO user. When the message is received, the user simply presses the **Enter** key and continues working. Similarly, a KOA script can also handle unexpected messages, and can continue processing based on the user's specifications.

To handle unexpected messages, KOA is designed so that each time the script sends a message to the application, it first checks if the VTAM application has sent a message to KOA. A message to KOA is not expected at this time (because the script was about to send its own message). The unexpected message is handled using one of the methods listed below:

If the current SCREENMODE is IGNUNSOL (default), KOA ignores the unexpected message and sends its own message to the application.

If the current SCREENMODE is GETUNSOL, KOA does not send its own message, and returns to the script with an appropriate return code. Based on the return code, the script determines whether to accept (with command GETSCREEN), or to ignore the unexpected message (by sending the same message as before).

The sample script below illustrates the handling of unexpected messages during a TSO session:

Table 21 KOA Sample Script to Handle Unexpected Messages During a TSO Session

Command	Explanation
`LABEL SENDISPF`	Specify label SENDISPF for script flow branching purposes.
`TYPE `ISPF'`	Type the TSO command ISPF at the keyboard.
`ENTER`	Send the command.
`IFVAR %VTAMERR NE `UNSOLMSG' GOTO CONTINUE`	Determine if any unexpected messages were received. If none were received, proceed to label CONTINUE. If unexpected messages were received, proceed to the next script command.
`GETSCREEN`	Receive (accept) the unexpected message.
`CLEAR`	Clear the screen.
`GOTO SENDISPF`	Branch back to label SENDISPF – try to send command ISPF to TSO again. You may need to implement a counter or other mechanism to prevent an infinite loop if the ISPF screen is never displayed.
`LABEL CONTINUE`	Specify label CONTINUE for script flow branching purposes. Continue script processing.

AutoRefresh Handling

Several products (especially performance monitors) periodically refresh the terminal screen without expecting input from the terminal. This automatic process (AutoRefresh) does not conform to the default KOA information exchange method, as explained in "Exchanging Messages" on page 58.

When a script sends a message to an application, KOA waits until a reply is received. Once the reply is received, the script continues (according to the default method). In order to receive a message from the application without first sending a message (as in the AutoRefresh case), specify command GETSCREEN. This command instructs KOA to receive any immediately available messages from the application. If no messages are currently available, KOA does not wait for a message from the application (as in the default method). Instead, KOA supplies a return code indicating that no message is available. Before retrying the operation, KOA can be instructed to either wait for a specific amount of time, or perform other actions.

The following example demonstrates AutoRefresh handling using OMEGAMON. (Commands /AUPON and /AUPOFF are used to start and stop AutoRefresh.):

Table 22 KOA Sample Script for Handling AutoRefresh using OMEGAMON

Command	Explanation
`TYPE `/AUPON'`	Type the OMEGAMON command /AUPON at the keyboard.
`ENTER`	Send the command.
`LABEL REFRESH`	Specify label REFRESH for script flow branching purposes.
`PAUSE 1000`	Wait 10 seconds.
`GETSCREEN`	Receive (accept) any available messages.
`IFVAR %VTAMERR EQ `NODATA' GOTO REFRESH`	Determine if special variable %VTAMERR indicates that no messages were available. If no messages were available, proceed to label REFRESH. If messages were available, proceed to the next script command.
`SETVAR %CPU SCREEN 7 12 2`	Set variable %CPU to the two screen characters at line 7, column 12. At this screen position, CPU usage is displayed.
`IFVAR %CPU LT 80 GOTO REFRESH`	If CPU usage (%CPU) is less than 80%, branch to label REFRESH.
`TYPE `/AUPOFF'`	Type the OMEGAMON command /AUPOFF at the keyboard.
`ENTER`	Send the command. Continue script processing.

Using KOA to Access the IOA Online Facility

The IOA Online facility can be activated under the VTAM environment, and is therefore available for KOA processing.

NOTE

The IOA Online facility is also available under various online communication systems (for example, CICS, IMS/DC, TSO). Therefore, KOA scripts can access the IOA Online environment directly through these facilities.

Since KOA is basically an extension of KSL, every KSL script (with minor modifications) can be run as a KOA script. This is achieved by including a few communication commands in the script, as in the sample script (member RUKORDER in IOA SAMPLE library) shown below:

Table 23 KOA Sample Script to Access the IOA Online Facility

Command	Explanation
KSL Commands:	
`TRACE OFF`	Turn the Trace facility off.
`MAXCOMMAND 999999`	Limit the number of times a command can be executed to 999999.
`CALLMEM SET2480`	Call predefined KSL script SET2480, which sets screen size.
`CALLMEM SET60132`	Call predefined KSL script SET60132, which sets page size.
KOA Communication Commands:	
`LOGON APPLID IOAVTAM SESSID IOA DATA TMNK`	Issue KOA command LOGON to log on to application IOAVTAM – the IOA Online environment. Call this session IOA and send TMNK as data to the logon process.
KSL Script Call:	
`CALLMEM ORDERRUL RULORDER LIBRARY RULNAME ODATE O`	Call predefined KSL script ORDERRUL, and pass arguments RULORDER, LIBRARY, RULNAME, ODATE and O.
KOA Communication Command:	
`LOGOFF`	Disconnect from the IOA Online environment.
KSL Command:	
`END`	Terminate the script.

Working with Control-O AutoEdit Variables

The Control-O AutoEdit facility is also available under KOA. All AutoEdit functions available under Control-O rules are supported. This expands the versatility of KOA and provides an additional communication path (method of sending information) to Control-O. The following AutoEdit commands are available:

```
SETOLOC %%cto-var = expression
SETOGLB %%cto-var = expression
SETVAR %koa-var DATA expression
```

where *expression* is any valid Control-O AutoEdit expression, which may contain KOA variables. KOA variables are resolved first.

Since KOA scripts can be executed independently of the Control-O monitor, the following restrictions apply:

- Global variables can be used only when the Control-O monitor is active.
- A Global variable's scope is limited to the CPU where the KOA script is running.

The following sample script illustrates the use of AutoEdit and KOA variables in a KOA script. The script tracks the number of times CPU usage was higher than 90% (according to information from a performance monitor).

Table 24 KOA Sample Script Using AutoEdit and KOA Variables (part 1 of 2)

Command	Explanation
`SETVAR %CPU SCREEN 7 12 2`	Set KOA variable %CPU to the two screen characters at line 7, column 12. At this screen position, CPU usage is displayed.
`IFVAR %CPU LT 90 GOTO CHKAGAIN`	Determine if CPU usage (%CPU) is less than 90%. If less than 90%, proceed to label CHKAGAIN. If not, proceed to the next script command.
`SETOLOC %%LOCCPU = %CPU`	Set local AutoEdit variable %%LOCCPU to the value of KOA variable *%CPU*.
`SETOGLB %%COUNT%%LOCCPU = %%COUNT%%LOCCPU %%$PLUS 1`	Increment global AutoEdit variable %%COUNT%%LOCCPU, which tracks the number of times CPU usage was 90% or higher.

Table 24 KOA Sample Script Using AutoEdit and KOA Variables (part 2 of 2)

Command	Explanation
SETVAR %PRLINE DATA `%CPU %%COUNT%%LOCCPU'	Set KOA variable %PRLINE to (variable names)`%CPU %%COUNT%%LOCCPU'. %PRLINE is used for printing purposes.
LABEL CHKAGAIN	Specify label CHKAGAIN for script flow branching purposes. Continue script processing.

Exception Handling

KOA scripts can communicate with any VTAM application that runs in the SNA network. This type of operating environment is very dynamic, and various events can occur during KOA operation. Therefore, scripts should be designed to handle potential exceptions.

The KOA language supplies the following variables that provide the cause of an exception:

Table 25 KOA Exception Handling Variables

Variable	Description
%VTAMRC	Contains the error ID.
%VTAMERR	Provides a short description of the error ID.
%VTAMFDBK	Supplies VTAM return codes which explain the situation.

To simplify script design, it is recommended that a general exception handling section be included in the script. This section should establish exception handling procedures for any possible exception, thus eliminating the need to check the result (successful, or unsuccessful) of each individual communication operation.

Command ON SCREENERROR allows specification of a label to which script flow branches if a communication operation is unsuccessful. Flow can be directed to different exception handling sections throughout the script. This is achieved by specifying different labels as parameters to command ON SCREENERROR at different points in the script. A script can also execute without any exception handling section (by specifying command ON SCREENERROR without parameters). In this last case, no exception handling procedures are initiated when an exception occurs.

The exception handling routine usually examines the three KOA variables listed above to check the cause of the exception.

There are two exceptions that are not controlled by the general exception handling section of a script:

When the session receives an unexpected (unsolicited) message, and command SCREENMODE GETUNSOL was specified. In this case, KOA variable %VTAMERR contains the value UNSOLMSG.

When command GETSCREEN is specified, but no data is available from the VTAM application. In this case, KOA variable %VTAMERR contains the value NODATA.

For these two situations, design the script so that it checks return codes after each communication operation.

Communicating With Control-O

KOA scripts communicate with Control-O using several methods:

KOA Parameters

A maximum of nine arguments can be included in a DO KSL statement. These arguments are assigned to KOA variables *%A1* through *%A9* (where the first argument is assigned to variable *%A1*, the second argument to *%A2*, and so on). For more information see the DO KSL parameter in the *Control-O User Guide*.

KOA Return Code

KOA scripts supply return codes through variable *%RC*. Return codes can be checked by a Control-O rule in the following way:

Table 26 KOA Sample Script to Check KOA Return Codes

Command	Explanation
`DO KSL CICSTEST WAITMODE Y`	Execute KOA script CICSTEST and wait until script execution is completed.
`IF %%KSLRC EQ 0`	Determine if variable %%KSLRC returns a return code of 0, indicating that the KOA script ended successfully. If successful, process the following script commands. If not, skip the following script commands.
`DO ...`	Process these script commands.
`ENDIF`	Indicate end of script commands relevant to return code 0.

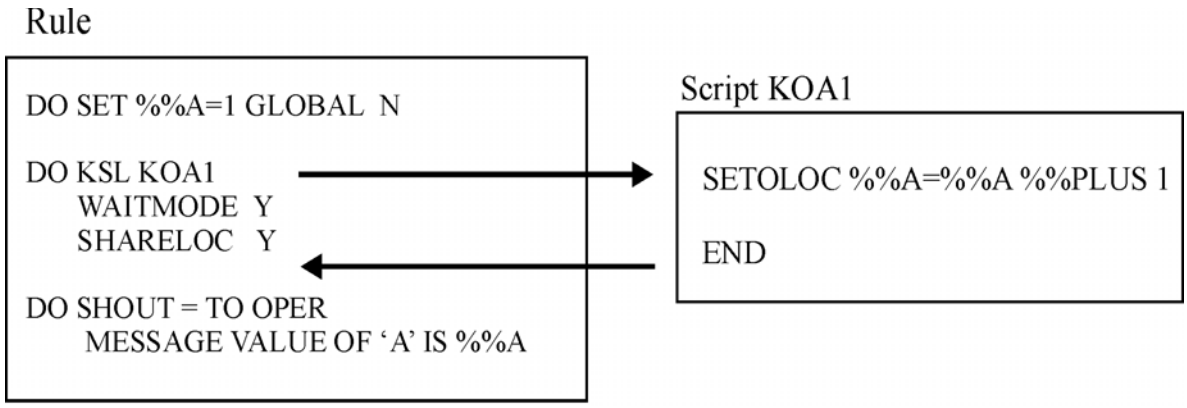
AutoEdit

Control-O Global variables can be accessed or set by a KOA script. KOA command SETOGLB sets Global variables. KOA commands that accept AutoEdit expressions as parameters can access Global variables.

Local and System AutoEdit variables are normally only accessible in the KOA script. However, when the WAITMODE subparameter is set to Y (Yes) and the SHARELOC subparameter is set to Y (Yes) in a DO KSL statement, the calling rule's Local and System variable environments are shared with the KOA script.

Example

Figure 3 Example of Control-O Global Variables Being Accessed or set by a KOA Script



Explanation

The above rule sets variable %%A to 1 and then calls script KOA1. KOA1 increments that same variable by 1. When the rule then shouts the message (shown below), %%A is resolved to 2:

```
CTO282I VALUE OF `A' IS 2
```

Shout Facility

KOA's SHOUT command allows a script to send messages to the console, TSO users, or the IOA Log. Messages to the console can be specified to trigger Control-O rules.

Using a Preset KOA Environment

Execution of DO KSL statements can be optimized by indicating (in the INITPROC subparameter) a script that handles a preset environment. In the text below, KOA scripts for preset environments are discussed in detail.

For information about DO KSL statements see the DO KSL parameter in the Control-O User Guide. For server-related installation details see the *INCONTROL for z/OS Installation Guide*.

The sample rule below invokes KOA script CICSINQ to check whether or not a data set is open. If the data set is closed, the rule invokes script CICSOPEN to open the data set. Script CICST specified in the INITPROC subparameter handles the preset environment.

Note the following about the sample rule below:

- IMMEDIATE–If scripts are to be executed by an Immediate server, specify Y (Yes) in the IMMEDIATE subparameter.
- INITPROC–If scripts are to be executed by a Special server, a Special server named CICST must be defined to Control-O.

If scripts are to be executed by a General server, either INITPROC should be blank or there must not be a special server defined by the specified name (CICST).

Figure 4 Sample Control-O for Optimizing DO KSL Statements using KOA Scripts

```
 RL: CICSOPEN   LIB CTOW.WORKO.RULES                              TABLE: AA
COMMAND ===>                                                    SCROLL===> CRSR
--------------------------------------------------------------------------------
  ON COMMAND  = CICSOPEN *
     JNAME          JTYPE         SMFID        SYSTEM       USERID
     ROUTE          DESC          CONSOLEID    CONSOLE
     APPEARED     TIMES IN      MINUTES                        And/Or/Not
  OWNER IOAADMIN GROUP                          MODE PROD    RUNTSEC
  DESCRIPTION   CHECK IF A FILE IS OPEN FOR CICS USAGE. IF NOT, OPEN IT
  DESCRIPTION
  ================================================================================
  /* GET THEFILENAME FROM THE COMMAND LINE
  DO SET      = %%FILE = %%$V2                                    GLOBAL  N
  DO
  /* INVOKE KOA CICSINQ (CEMT INQUIRY FILE ( ))
  DO KSL      = CICSINQ %%FILE
    WAITMODE   Y                                      TIMEOUT 9999    STOP Y
     INITPROC   CICST         SHARELOC Y               IMMEDIATE N
  DO
  /* IF NEEDED INVOKE KOA CICSOPEN (CEMT SET FILE( ) OPEN)
  IF       %%DSTATUS NE O
  DO KSL      = CICSOPEN %%FILE
     WAITMODE   Y                                      TIMEOUT 9999    STOP Y
     INITPROC   CICST         SHARELOC Y               IMMEDIATE N
FILL IN RULE DEFINITION. CMDS: EDIT, SCHED, OPT, SHPF                   07.55.22
```

Figure 5 Sample KOA Script (CICST) invoked by Control-O Rule Subparameter INITPROC to handle the Preset Environment

```
IFVAR `%A1' EQ INIT  GOTO INIT
IFVAR `%A1' EQ TERM  GOTO TERM
IFVAR `%A1' EQ RESET GOTO RESET
GOTO EXIT
LABEL INIT
LOGON APPLID CICSTEST SESSID CICS
CLEAR
GOTO EXIT
LABEL RESET
CLEAR
GOTO EXIT
LABEL TERM
SCREENMODE NORECEIVE
TYPE `CESF LOGOFF'
ENTER
LOGOFF
GOTO EXIT
LABEL EXIT
RETURN
```

Figure 6 Sample KOA Script (CICSINQ) invoked by Control-O Rule to Check whether a Data Set is Open

```
TYPE `CEMT INQUIRY FILE( %A1 )'
ENTER
SETVAR %STATUS SCREEN 3 22 1
SETOLOC %%DSTATUS = %STATUS
PF03
RETURN
```

Figure 7 Sample KOA Script (CICOPEN) invoked by Control-O Rule to Open a Closed Data Set

```
TYPE  `CEMT SET FILE( %A1 ) OPEN'
ENTER
PF03
RETURN
```

Execution of the DO KSL statements mentioned in the above rule depends on which type of server is used. Below is a step-by-step description of how these statements are executed for each type of server.

Table 27 Step-by-Step Description of DO KSL Statements in Sample Control-O Rule

Rule	Immediate Server	General Server	Special Server
DO KSL = CICSINQ	1. An Immediate server is created.	1. A General server is created if one does not already exist.	1. A Special server is created if one does not already exist for the necessary environment.
	2. Server invokes CICST with the INIT parameter.	2. Server invokes CICST with the INIT parameter.	2. Server invokes CICST with the INIT parameter.
	3. Server invokes CICSINQ.	3. Server invokes CICSINQ.	3. Server invokes CICSINQ.
	4. Server invokes CICST with the RESET parameter.	4. Server invokes CICST with the RESET parameter.	4. Server invokes CICST with the RESET parameter.
	5. Server invokes CICST with the TERM parameter.	5. Server invokes CICST with the TERM parameter.	5. Special server is ready for a new request.
	6. Immediate server is terminated.	6. General server is ready for a new request.	
DO KSL = CICSOPEN	1. An Immediate server is created.	1. Server invokes CICSOPEN.	1. Server invokes CICSOPEN.
	2. Server invokes CICST with the INIT parameter.	2. Server invokes CICST with the RESET parameter.	2. Server invokes CICST with the RESET parameter.
	3. Server invokes CICSOPEN.	3. Server invokes CICST with the TERM parameter.	3. Special server is ready for a new request.
	4. Server invokes CICST with the RESET parameter.	4. General server is ready for a new request.	4. As the server is being shut down, it invokes CICST with the TERM parameter.
	5. Server invokes CICST with the TERM parameter.		
	6. Immediate server is terminated.		

Preset Environments and Batch Jobs

Before a KOA script designed for a preset environment can be executed, the preset environment must be defined. To achieve this, Control-O normally invokes the necessary scripts to preset the environment. When a KOA script designed for a preset environment is to be executed in batch, a controlling KOA script must be included to simulate Control-O actions. The controlling script invokes the scripts which create and maintain the preset environment.

The following job contains a controlling KOA script. It invokes the same scripts shown in the previous example.

Figure 8 Sample Job Containing a Controlling KOA Script

```
//JOB1    JOB ....
//STEP0001 EXEC IOARKOA,OUTDUMP=X
//DAKSLPRM DD *
*
  SETVAR %FIL DATA `FILE1'
  CALLMEM CICST INIT
  CALLMEM CICSINQ %FIL
  CALLMEM CICST RESET
  IFVAR %%DSTATUS EQ `0' GOTO SKIPOPEN
  CALLMEM CICSOPEN %FIL
  CALLMEM CICST RESET
LABEL SKIPOPEN
  CALLMEM CICST TERM
  END
*
//
```

Updating KOA Scripts Requiring a Preset Environment

Servers executing DO KSL requests sequentially (that is, General and Special servers) load the KOA script the first time it is executed. When the same KOA script is again requested, the server does not reload the script, but instead uses the previously loaded version from memory.

If a KOA script is updated, it is necessary to restart the server so that it is forced to reload the scripts. A server can be restarted by the Server Status screen or by specifying operator command:

```
F CONTROLO,SERVER=name,TERM
```

The server terminates, and restarts automatically upon the next DO KSL request.

Testing a Script

When testing a KOA script, it may be necessary to make changes. To avoid restarting a General or Special server after each change in the script:

Temporarily define the DO KSL statement with the IMMEDIATE subparameter set to Y. This causes an Immediate server to be created for each request.

Test scripts by submitting them in batch jobs (described earlier in this chapter).

KOA Recording

The KOA Recorder facility, which is available under IOA TSO and IOA cross memory, makes it easy to create KOA scripts. The KOA Recorder generates a script by automatically recording your keystrokes and saving the keystrokes in a file. This eliminates the need to write scripts manually and reduces the chance of error. You can edit the file as necessary.

Scripts generated by the KOA Recorder can be submitted in the same way as manually created scripts, that is, by using batch procedure IOARKOA.

The KOA facility is described in "KOA Recording" on page 72

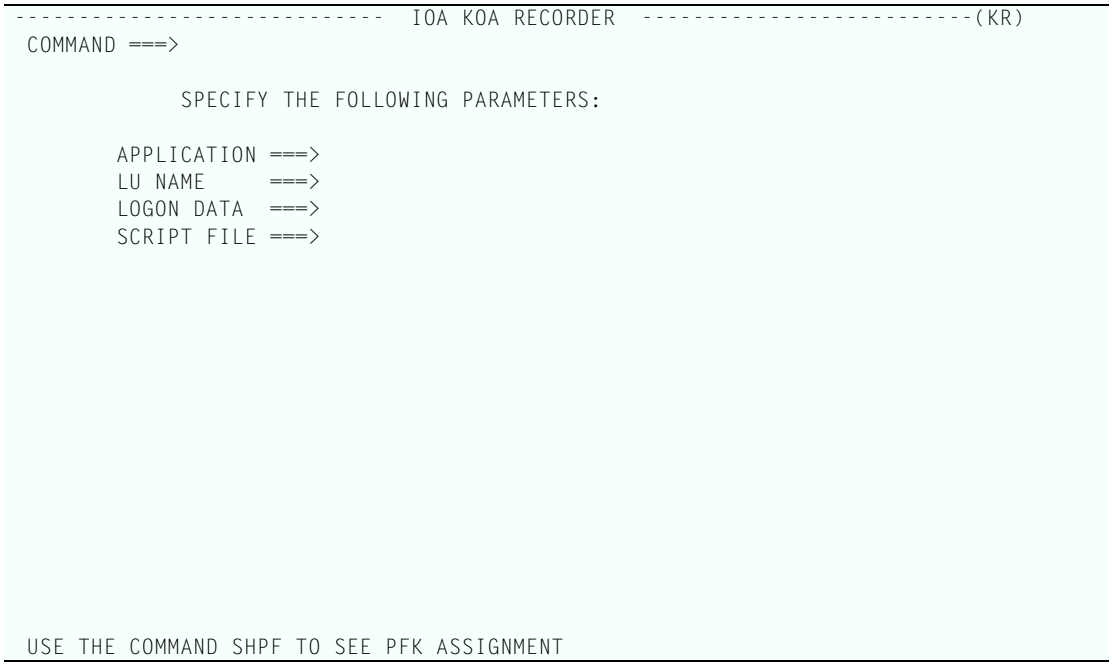
NOTE

The KOA Recorder facility simulates a hardware terminal (SNA LUTYPE2) and supports terminal models 2, 3, 4 and 5, with or without extended data stream capabilities. Therefore, corresponding VTAM LOGMODEs should be used with the KOA Recorder. Ensure that the LU for a VTAM logmode attribute used in the recording session corresponds to the logmode of the recorded session. For more information about defining VTAM LUs and logmodes, refer to the *INCONTROL for z/OS Administrator Guide*.

To start the KOA Recorder, select Option OK, KOA RECORDER, on the IOA Primary Option menu. The IOA KOA Recorder screen, which is illustrated in Figure 9, is displayed.

KOA Recorder Screen

Figure 9 KOA Recorder Screen



To record a script, specify a value for each of the fields in Table 28.

Table 28 Fields of the KOA Recorder Screen

Fields	Description
APPLICATION	VTAM application to be tested, such as TSO or CICS).
LU NAME	Logical unit name associated with the application. Logical units are defined in the CTOPARM member of the IOAPARM installation library. Mandatory.
LOGON DATA	Initial keystrokes to send to the application. These keystrokes are usually a logon ID. Optional.
SCRIPT FILE	PS file, or PDS library and member, in which the KOA Recorder saves your keystrokes. Mandatory. The script file must have an FB record format and a block size of 3,120 bytes. If you specify an existing (preallocated) file, it is overwritten. If the specified file does not exist, the KOA Recorder will automatically create it.

To begin the actual recording, press **Enter**. Both the recording and the VTAM application that you specified begin simultaneously.

When you quit your VTAM application (for example, in a TSO session, by typing LOGOFF at the TSO READY prompt), the KOA Recorder stops recording, and the KOA Recorder Screen reappears.

To leave the KOA Recorder screen, press **PF03/PF15** (**END**).

Sample Output Script

The following is an example of steps performed by a user while the KOA Recorder is in session. The KOA Recorder will record these steps as a script.

1 Display the Rule Definition Entry Panel.

2 Locate the DAILY table and select the DAILY rule.

3 Modify the DAILY rule so that MODE LOG is Y.

4 Exit the DAILY rule, exit the DAILY table, and save the updated rule.

5 Force the DAILY Rule table.

6 Display the Automation Log panel

7 Check that the DAILY rule was loaded and triggered.

8 Quit IOA.

9 Log off TSO and stop recording.

From these steps, the KOA Recorder generates the script shown in Figure 10.

Figure 10 Sample KOA Output Script

```
LOGON APPLID TSO      SESSID RCRD TERMINAL KSLT0001
TYPE 'N18A'
ENTER
TYPE 'N18     '
ENTER
TYPE 'OR                                              '
ENTER
TYPE 'CTOP.PROD.RULES              '
ENTER
TYPE 'L DAILY                                        '
ENTER
TYPE 'S'
ENTER
CURSOR NEWLINE
TYPE 'S'
ENTER
CURSOR POS 2  15
TYPE 'F PROD                                          '
ENTER
TYPE 'LROD'
ENTER
PF03
PF03
TYPE 'Y'
ENTER
TYPE 'F'
ENTER
TYPE 'Y'
ENTER
TYPE '=OL                                             '
ENTER
TYPE 'F DAILY                                         '
ENTER
PF08
TYPE '=X                                              '
ENTER
TYPE 'LOGOFF'
ENTER
END
```

Chapter 4

Using KSL with Control-M/Restart

Control-M/Restart provides several predefined KSL scripts as a utility for producing reports. Descriptions and sample output of these reports are provided below. The scripts are located in the IOA SAMPLE and IOA KSL libraries and can be modified according to your site requirements. The name of the member that contains the script appears in parentheses below

The central source of information for the reports is the IOA Log, which maintains an audit trail of all job restart (Control-M/Restart) events. Other relevant information can come from the Control-M Active Jobs file or from user tables.

Automatic Restart Definition utility (REPCTRDF)

Basic restart definitions for jobs in existing tables do not need to be manually entered. The REPCTRDF KSL utility automatically generates basic restart definitions for all jobs in a specified table. By using the REPCTRDF utility, you can in a matter of minutes implement basic restart definitions for Control-M/Restart for your entire site.

Specify the parameters described in Table 29 for the REPCTRDF utility:

Table 29 Parameters for Automatic Restart Definition utility (REPCTRDF)

Parameter	Description
library	Library name. Must be a partitioned data set containing tables
table	Table name
confirm	Determines the value of the CONFIRM parameter in the DO IFRERUN statement for the job. Valid values are: ■ Y (Yes): Restart step with CONFIRM Y. ■ N (No): Restart step with CONFIRM N.
tasktype	Specific task type, or ALL for all task types. (“Dummy” jobs are not be updated.)

The following statements are added to the job scheduling definitions (if they are not already there):

```
ON PGMST ANYSTEP  PROCST          CODES S***   U****  C2000          A/O
  DO IFRERUN   FROM $ABEND   .          TO          .            CONFIRM N
```

NOTE

The CONFIRM field is assigned the value specified in the CONFIRM parameter in the REPCTRDF KSL utility.

Manual Restart Confirmation report (REPLOGCN)

The Manual Restart Confirmation report details restart jobs that were manually released for execution using the Control-M/Restart CONFIRM option within a specified period (for example, week, month).

Figure 11 Manual Restart Confirmation report

```
BMC SOFTWARE, INC.                    IOA KEY-STROKE REPORTING LANGUAGE   (VER 6.1.0)
DATE 09/09/00 PAGE 000001
                 I  O  A   -   L  O  G - MANUAL RESTART CONFIRMATION REPORT     FROM 090900  TO 090900
+--------+--------+----------+----------+---------+-------------------------------------------------
| DATE   | TIME   | USERID   | JOBNAME  |  ODATE  | MESSAGE
+--------+--------+----------+----------+---------+-------------------------------------------------
| 090900 | 084646 | PROD     | PRODJOB1 | 090900  | RESTART CONFIRMED
| 090900 | 084824 | PROD     | PRODJOB1 | 090900  | RESTART DECISION DELETED
| 090900 | 084825 | PROD     | PRODJOB1 | 090900  | RESTART CONFIRMED
| 090900 | 084947 | GENERAL  | PRODJOB1 | 090900  | MODIFIED FROM S030. TO.   (ORDERID=0000L)
| 090900 | 085202 | PROD     | PRODJOB2 | 090900  | RESTART CONFIRMED
```

Restart detail report (REPLOGRS)

The Restart Detail report is a list of restart jobs executed over a particular period (for example, daily, weekly). The listing displays restart job, restart step, use of Control-M/Restart CONFIRM option, and so on.

Figure 12 Restart Detail Report (REPLOGRS)

```
BMC SOFTWARE, INC.             IOA KEY-STROKE REPORTING LANGUAGE   (VER 6.1.0)
DATE 09/09/00 TIME 08.53 PAGE 000001

                    I  O  A  -  L  O  G - RESTART DETAIL REPORT                FROM 090900  TO 090900

+--------+--------+----------+----------+---------+-------------------------------------------------
-
| DATE   | TIME   | USERID   | JOBNAME  |  ODATE  | MESSAGE
+--------+--------+----------+----------+---------+-------------------------------------------------
-
| 090900 | 084646 | PROD     | PRODJOB1 | 090900  | RESTART CONFIRMED
| 090900 | 084705 | PROD     | PRODJOB1 | 090900  | RESTARTING FROM STEP S020. TO STEP S050.
| 090900 | 084824 | PROD     | PRODJOB1 | 090900  | RESTART DECISION DELETED
| 090900 | 084825 | PROD     | PRODJOB1 | 090900  | RESTART CONFIRMED
| 090900 | 084947 | GENERAL  | PRODJOB1 | 090900  | MODIFIED FROM S030. TO.   (ORDERID=0000L)
| 090900 | 085006 | PROD     | PRODJOB1 | 090900  | RESTARTING FROM STEP S030. TO STEP S050.
| 090900 | 085202 | PROD     | PRODJOB2 | 090900  | RESTART CONFIRMED
| 090900 | 085217 | PROD     | PRODJOB2 | 090900  | RESTARTING FROM STEP S030. TO STEP S070.
```

Last Night Restart History report (REPJOBRS)

The Last Night Restart History report provides a complete execution history of all jobs that were restarted during the previous night. The report displays all successful and unsuccessful restarts of the job. Job start time, end time and termination condition codes are displayed. For each restart, the restart step of the job is also displayed.

Figure 13 Last Night Restart History Report (REPJOBRS)

```
BMC SOFTWARE, INC.         IOA KEY-STROKE REPORTING LANGUAGE          (VER 6.1.0)          DATE
09/09/00 TIME 08.53 PAGE 000001

                 LAST NIGHT RESTART HISTORY REPORT
                 =================================
 NAME      USERID   ODATE   JOBNAME  JOBID  TYP --------- STATUS -----------
 PRODJOB1  PROD     090900  PRODJOB1/01242  JOB ENDED "OK" (RESTARTED) (RUN 4)

 DATE   TIME   ODATE  USERID   CODE    ------ M E S S A G E --------------------

 090900 084505 090900 PROD     JOB511I JOB PRODJOB1 ODATE 090900 ID=0000L TASK= M34 /FDSF - PLACED ON
AJF - PRODUCTION JOB NUMBER ONE
 090900 084509 090900 PROD     SEL203I JOB PRODJOB1 ELIGIBLE FOR RUN
 090900 084511 090900 PROD     SUB133I JOB PRODJOB1 PRODJOB1/01238 SUBMITTED
 090900 084520 090900 PROD     SPY281I JOB PRODJOB1 PRODJOB1/01238 START 00253.0845 STOP 00253.0845
CPU 0MIN 00.05SEC SRB 0MIN 00.00SEC 0.02 1AFDSF
 090900 084521 090900 PROD     SPY254I JOB PRODJOB1 PRODJOB1/01238 SCANNED
 090900 084521 090900 PROD     SEL206W JOB PRODJOB1 PRODJOB1/01238 ABENDED CC S0C4 STEP S020
 090900 084521 090900 PROD     SEL214I JOB PRODJOB1 PRODJOB1/01238 RERUN NEEDED
 090900 084521 090900 PROD     SEL205I JOB PRODJOB1 PRODJOB1/01238 RERUN IN PROCESS USING MEM PRODJOB1
 090900 084521 090900 PROD     SEL286I JOB PRODJOB1 PRODJOB1/01238 WAITING FOR CONFIRMATION
 090900 084651 090900 PROD     SEL203I JOB PRODJOB1 ELIGIBLE FOR RUN
 090900 084652 090900 PROD     SUB133I JOB PRODJOB1 PRODJOB1/01240 SUBMITTED
 090900 084705 090900 PROD     CTR082I JOB PRODJOB1 PRODJOB1/01240 RESTARTING FROM STEP S020. TO STEP
S050.
 090900 084705 090996 PROD     SPY281I JOB PRODJOB1 PRODJOB1/01240 START 00253.0846 STOP 00253.0846
CPU 0MIN 00.84SEC SRB 0MIN 00.03SEC 0.08 1AFDSF
 090900 084706 090900 PROD     SPY254I JOB PRODJOB1 PRODJOB1/01240 SCANNED
 090900 084706 090900 PROD     SEL206W JOB PRODJOB1 PRODJOB1/01240 ABENDED CC S0C7 STEP S040
 090900 084706 090900 PROD     SEL214I JOB PRODJOB1 PRODJOB1/01240 RERUN NEEDED
 090900 084706 090900 PROD     SEL205I JOB PRODJOB1 PRODJOB1/01240 RERUN IN PROCESS USING MEM PRODJOB1
 090900 084706 090900 PROD     SEL286I JOB PRODJOB1 PRODJOB1/01240 WAITING FOR CONFIRMATION
 090900 084827 090900 PROD     SEL203I JOB PRODJOB1 ELIGIBLE FOR RUN
 090900 084828 090900 PROD     SUB133I JOB PRODJOB1 PRODJOB1/01241 SUBMITTED
 090900 084835 090900 PROD     SPY281I JOB PRODJOB1 PRODJOB1/01241 START 00253.0848 STOP 00253.0848
CPU 0MIN 00.09SEC SRB 0MIN 00.00SEC 0.03 1AFDSF
 090900 084835 090900 PROD     SPY254I JOB PRODJOB1 PRODJOB1/01241 SCANNED
 090900 084835 090900 PROD     SEL206W JOB PRODJOB1 PRODJOB1/01241 ABENDED CC S0C7 STEP S040
 090900 084835 090900 PROD     SEL214I JOB PRODJOB1 PRODJOB1/01241 RERUN NEEDED
 090900 084835 090900 PROD     SEL215W JOB PRODJOB1 PRODJOB1/01241 NO (MORE) RERUNS
 090900 084951 090900 PROD     SEL220I JOB PRODJOB1 WILL BE RERUN
 090900 084951 090900 PROD     SEL203I JOB PRODJOB1 ELIGIBLE FOR RUN
 090900 084952 090900 PROD     SUB133I JOB PRODJOB1 PRODJOB1/01242 SUBMITTED
 090900 085006 090900 PROD     CTR082I JOB PRODJOB1 PRODJOB1/01242 RESTARTING FROM STEP S030. TO STEP
S050.
 090900 085006 090900 PROD     SPY281I JOB PRODJOB1 PRODJOB1/01242 START 00253.0849 STOP 00253.0849
CPU 0MIN 02.05SEC SRB 0MIN  00.04SEC 0.10 1AFDSF
 090900 085006 090900 PROD     SPY254I JOB PRODJOB1 PRODJOB1/01242 SCANNED
 090900 085006 060600 PROD     SEL208I JOB PRODJOB1 PRODJOB1/01242 ENDED "OK"
```

Restart Time Savings report (RPRSV)

The Restart Time Savings report lists job restarts by Control-M/Restart during the specified period. For each listed job restart, the report provides general information about the job and summary information about the execution time saved as a result of using a restart under Control-M/Restart instead of a job rerun. For each restart, the report displays the number of steps skipped, the elapsed time saved, and the CPU time saved.

Figure 14 Restart Time Savings report (RPRSV)

```
BMC SOFTWARE, INC.            IOA KEY-STROKE REPORTING LANGUAGE   (VER 6.1.0)
DATE  09/09/00 TIME 10.20 PAGE 000001

                    *          RESTART TIME SAVINGS REPORT         *

            ***-------------------------------------------------------------***
            *** FROM DATE:      090800        TO DATE:      090900        ***
            ***-------------------------------------------------------------***

+========+========+==========+=======+===========+=========+=========+==========+
|  DATE  |  TIME  |    JOB   |  JOB  |    STEP   |  # OF   | ELAPSED |   CPU    |
|        |        |   NAME   |   ID  |    NAME   | SKIPPED |  TIME   |   TIME   |
|        |        |          |       |           |  STEPS  |  SAVED  |   SAVED  |
|        |        |          |       |           |         | (HH:MM) | (M:SS:HS)|
+========+========+==========+=======+===========+=========+=========+==========+
| 090800 | 091657 | R0014T01 | 02186 | R0014T01  |    0    |  00:00  |  0:00:00 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090800 | 121834 | R0006T01 | 02464 | R0006T01  |    1    |  00:03  |  0:00:20 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090800 | 122656 | R0007T01 | 02475 | R0007T01  |    1    |  00:00  |  0:00:16 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090800 | 123215 | R0008T01 | 02487 | R0008T01  |    7    |  01:07  |  0:30:14 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090900 | 003818 | R0009T01 | 03297 | R0009T01  |    3    |  00:39  |  0:11:20 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090900 | 014309 | R0010T01 | 03565 | R0010T01  |    1    |  00:00  |  0:00:19 |
+--------+--------+----------+-------+-----------+---------+---------+----------+
| 090900 | 024911 | R0011T01 | 03910 | R0011T01  |    9    |  01:54  |  0:52:14 |
+--------+--------+----------+-------+-----------+---------+---------+----------+

                              ******** END OF REPORT *********
```

Last Night SYSOUT Scan Summary report (REPJOBSY)

The Last Night SYSOUT Scan Summary report provides an execution history for jobs with archived SYSOUTs that ran the previous night. Either the first archived SYSOUT, or all archived SYSOUTs, can be displayed in the report for specified jobs (or all jobs).

Figure 15 Last Night SYSOUT Scan Summary Report (REPJOBSY)

```
 BMC SOFTWARE, INC.                    IOA KEY-STROKE REPORTING LANGUAGE   (VER 6.1.0) DATE 09/09/00
TIME 10.06 PAGE 000001

                      LAST NIGHT SYSOUT SCAN SUMMARY REPORT
                      =====================================

  ------------------------ CONTROL-M ACTIVE ENVIRONMENT ----------------------(3)
  COMMAND ===>                                                  SCROLL===> CRSR
  O NAME     OWNER    ODATE  JOBNAME  JOBID TYP --------- STATUS ----------- UP
    PRODJOB1 N15A     090900 PRODJOB1/29944 JOB ENDED "OK" (RESTARTED) (RUN 2)
                                                PRIOR RUN: ENDED- NOT "OK" - ABENDED
   ====== >>>>>>>>>>>>>>>>>> BOTTOM OF ACTIVE JOBS LIST <<<<<<<<<<<<<<<<<< ======

  ------------------------ JOB ORDER EXECUTION HISTORY ---------------------(3.V)
  COMMAND ===>                                                  SCROLL===> CRSR
  MEMNAME PRODJOB1   OWNER N15A                          ORDERID 001UA  ODATE 090900
  O JOBNAME  JOBID  DATE  START ELAPSED PAGES MAX RC ---------- STATUS ----------
    PRODJOB1 29941 090900 10:04    0:03 00009 S0C4   ENDED- NOT "OK" - ABENDED
    PRODJOB1 29944 090900 10:05    0:16 00011        ENDED "OK" (RESTARTED)

  ======= >>>>>>>>>>> BOTTOM OF ACTIVE JOB ORDER HISTORY LIST <<<<<<<<<<< =======

  ------------- CONTROL-M/RESTART   SYSOUT VIEWING ------   PAGE      1 OF      9
  COMMAND ===>                                                  SCROLL===> CRSR
  MEMNAME PRODJOB1  OWNER N15A        JOBNAME PRODJOB1      ODATE 090900
 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----
10
1                     J E S 2  J O B  L O G  --  S Y S T E M  E S A 1  --  N O D E  N O D E 7
 0
  10.04.40 JOB29941  $HASP373 PRODJOB1 STARTED - INIT  3 - CLASS A - SYS ESA1
  10.04.40 JOB29941  IEF403I PRODJOB1 - STARTED - TIME=10.04.40
  10.04.42 JOB29941  PRODJOB1.IOATEST .#01; - COMPLETION CODE=0000
  10.04.43 JOB29941  IEF450I PRODJOB1 IOATEST S020 - ABEND=S0C4 U0000 REASON=00000000
                             TIME=10.04.43
  10.04.43 JOB29941  PRODJOB1.IOATEST .#02; - COMPLETION CODE=S00C4 - ABENDED######
  10.04.43 JOB29941  PRODJOB1.IOATEST .#03; - COMPLETION CODE=NOT RUN  10.04.43 JOB29941
PRODJOB1.IOATEST .#04; - COMPLETION CODE=NOT RUN
  10.04.43 JOB29941  PRODJOB1.IOATEST .#05; - COMPLETION CODE=NOT RUN
  10.04.43 JOB29941  IEF404I PRODJOB1 - ENDED - TIME=10.04.43
  10.04.43 JOB29941  $HASP395 PRODJOB1 ENDED
 0------ JES2 JOB STATISTICS ------
 -    09 SEP 00 JOB EXECUTION DATE
 -            9 CARDS READ
 -          399 SYSOUT PRINT RECORDS
 -            0 SYSOUT PUNCH RECORDS
 -           33 SYSOUT SPOOL KBYTES
 -         0.05 MINUTES EXECUTION TIME
```

Appendix A

AutoEdit Facility in KSL

The AutoEdit facility provides additional data manipulation capabilities. It is composed of the following types of AutoEdit symbols and instructions:

- System variables
- user-defined variables
- operators
- functions

NOTE

KSL and Control-M use different AutoEdit processors. Therefore, if a KSL script containing KSL AutoEdit terms is submitted under Control-M, the Control-M AutoEdit %%RANGE statement must be used in the JCL to ensure that the Control-M AutoEdit processor skips, that is, that it does not process, the KSL script.

System Variables

System variables are predefined, commonly used variables whose values are automatically updated and maintained by the AutoEdit facility.

The System variable format is:

```
%%$var
```

where *var* represents the name of the System variable.

Each AutoEdit variable begins with "%%". Each variable resolves to (is replaced by) the corresponding system value. AutoEdit System variables are described on the following pages.

Example

```
TYPE '%%$DATE'
```

resolves on the 12th of December 2001 to

```
TYPE '011010'
```

AutoEdit System Variables:

NOTE

In the following table, the ◊ symbol following an AutoEdit System variable indicates that if the variable is specified without the $ in the prefix, the variable is still supported.

Table 30 AutoEdit System Variables (part 1 of 3)

Variable	Description
%%. ◊	Concatenation character
%%$BLANK ◊	Resolves to one blank
%%$BLANK*n* ◊	Resolves to *n* blanks, where *n* is a number between 1 and 99.

Table 30 AutoEdit System Variables (part 2 of 3)

Variable	Description
%%$D2X *num*	Hexadecimal number resulting from the conversion of the decimal number *num*. The largest number that can be converted is 2147483647 (that is, 231 – 1). For example: %%$D2X 4095 converts to 'FFF'.
%%$DATE ◊	Current system date (format yymmdd).
%%$DAY ◊	Current system day (format dd).
%%$JULDAY ◊	Current system day (Julian format jjj).
%%$LENGTH *varname* ◊	Length of variable *varname*.
%%$MONTH ◊	Current system month (format mm).
%%$MVSLEVEL	MVS product version (eight characters) under which IOA is running. Examples: SP3.1.1, SP4.2.2
%%$NULL ◊	Indicates a null variable (a variable with length 0).
%%$PARSRC	Return code from a %%$PARSE function. Indicated whether the parsed string matched all string patterns in the template. Possible values are: ■ 0 – The parsed string fully matched the string patterns in the template. ■ 4 – At least one string pattern in the template was not matched.
%%$RDATE ◊	Current working date (format yymmdd).
%%$RDAY ◊	Current working day (format dd).
%%$RJULDAY ◊	Current working day of the year (Julian format jjj).
%%$RMONTH ◊	Current working month (format mm).
%%$RWDAY ◊	Current working day of the week. Format is *d*, where *d* is 1 through 6 or 0 (for example, 1=Sunday, 2=Monday, ... 6=Friday, 0=Saturday).
	Note: Start of the week depends on an IOA installation parameter specifying whether 1=Sunday or 1=Monday. For your site standard, see your INCONTROL administrator.
%%$RYEAR ◊	Current working year (format yy).
%%$SMFID ◊	The SMF ID of the CPU running the KSL script.
%%$SSNAME ◊	Name of the IOA subsystem.
%%$SUBSTR v*arname pos len* ◊	Substring of variable *varname* starting at position *pos* with length *len*.
%%$TIME ◊	Time of day (format hhmmss).

Table 30 AutoEdit System Variables (part 3 of 3)

Variable	Description

%%$UNDEF ◊	Indicates an undefined variable. This variable can be used:
	■ To test whether a variable is defined:
	`IF %%variable EQ %%$UNDEF`
	■ To delete a variable:
	`SETOLOC = %%variable = %%$UNDEF`

%%$WDAY ◊	Current system day of the week. Format is d, where d is 1 through 6 or 0 (for example, 1=Sunday, 2=Monday, ... 6=Friday, 0=Saturday).
	Note: Start of the week depends on an IOA installation parameter specifying whether 1=Sunday or 1=Monday. For your site standard, see your INCONTROL administrator.
%%$W*n varname* ◊	The *n*th word (a comma or a blank can serve as a delimiter) of variable *varname*. *n* can be a value from 1 through 99. For example, %%$W3 %%MESSAGE represents the third word in the original user-defined message text.
%%$WORDS *varname* ◊	Number of words in variable *varname*. Delimiters are commas and/or blanks within the variable.
%%$X2D *string*	Numeric decimal string resulting from the conversion of the hexadecimal number string. The maximum number that can be converted is 7FFFFFFF. Example: %%$X2D FFF converts to '4095'.
%%$YEAR ◊	Current system year (format yy).

User-Defined Variables

The user-defined variables capability is designed to provide additional flexibility. You can define your own symbols using the KSL command SETOLOC and use them in other KSL commands. They are automatically resolved when the KSL is executed.

A user-defined variable can be any alphanumeric string starting with %%. The characters @ # $ _ are also valid. Lowercase characters are resolved, but upon resolution they remain lowercase and are not translated to uppercase characters.

When the AutoEdit facility identifies a string that starts with %%, the string is assumed to be an AutoEdit variable or instruction. If the string is a reserved AutoEdit symbol or a System variable, it is interpreted as such. Otherwise the string is assumed to be a user-defined variable.

Rules of Variable Substitution

A KSL command can contain expressions including both KSL and AutoEdit variables. Variable substitution is performed in the following order:

1. All KSL variables (variables preceded by a single % character) are substituted sequentially from left to right.

   Example

   ```
   TYPE '%A %%$PLUS 1'
   ```

   Assuming that the value of %A is 1, variable substitution begins with the KSL variable substituted as follows:

   ```
   TYPE '1 %%$PLUS 1'
   ```

2. If the resulting expression contains AutoEdit symbols (in this example, %%$PLUS), variables are substituted sequentially from right to left until the symbol is assigned a value.

   In the above example, TYPE ‘1 %%$PLUS 1’ resolves to TYPE ‘2’.

The largest number that can be handled by mathematical AutoEdit operations is 2^{31} - 1, that is, 2147483647.

Examples

The following are additional examples of AutoEdit variable substitution.

Example 1

```
%%SMF_TAPE_%%DAY,
```

resolves on the third of the month to:

```
%%SMF_TAPE_03,
```

The AutoEdit facility then tries to resolve the symbol %%SMF_TAPE_03. Assuming the value of the symbol in the Global environment is EE1022, the result is:

EE1022

To concatenate two symbols, separate them with a period. Before AutoEdit variables are concatenated, trailing blanks are eliminated.

Example 2

```
%%DAY.%%MONTH
```

resolves on the 4th of December to:

0412

NOTE

Specification of %%DAY%%MONTH would result in an attempt to resolve %%DAY12 (a user-defined variable).

In order to put a period between two symbols, use two consecutive periods.

Example 3

```
%%DAY..%%MONTH
```

resolves on the 4th of December to:

04.12

To concatenate a symbol and a constant, use %%. (concatenation symbol).

Example 4

```
A91%%DAY%%.UP
```

resolves on the 4th of December to:

A9104UP

NOTE

Specification of A91%%DAYUP would result in an attempt to resolve %%DAYUP (a user-defined variable).

AutoEdit Operators

AutoEdit operators can be used in conjunction with AutoEdit symbols. Valid AutoEdit operators are:

Table 31 AutoEdit Operators

Operator	Description
%%$PLUS	Add two operands.
%%$MINUS	Subtract the second operand from the first operand.
%%$TIMES	Multiply one operand by another operand.
%%$DIV	Divide the first operand by the second operand.

The format for use of AutoEdit symbols and operators is:

```
operand operator operand
```

Only one operator can be used in an expression.

Operands must resolve into positive numeric constants. The final result is translated into a character string. For example:

```
SETOLOC %%x = %%x %%$PLUS 1
```

User-defined variable %%x is incremented by one.

%%$CALCDATE Function

The %%$CALCDATE function performs date calculations based on a specified date. Format:

```
%%$CALCDATE date ± quantity
```

where:

- *date* is the date in yymmdd format.
- *quantity* is the number (or numeric AutoEdit expression) of days to add to or subtract from the date (from 1 to 366).

NOTE

%%$CALCDATE operates on Gregorian dates only; Julian dates, such as %%JULDAY, cannot be specified.

Example

```
SETOLOC %%A = %%$CALCDATE %%$RDATE -1
```

On February 1, 2000:

```
SETOLOC %%A = 000131
```

%%$SUBSTR Function

The %%$SUBSTR function extracts a substring from the input string, and returns the attached substring. Format:

```
%%$SUBSTR strng startpos len
```

where:

- *strng* is the input string from which the substring is extracted.
- *startpos* is the first character of the input string to extract.
- *len* is the number of characters to extract.

startpos and *len* must be numbers (or numeric AutoEdit expressions) and greater than zero.

If (*startpos* + *len* – 1) is greater than the *strng* length, the function is not executed and the value returned is null.

Example

```
SETOLOC %%A = %%$CALCDATE %%$RDATE -1
SETOLOC %%AMON = %%$SUBSTR %%A 3 2
```

On December 1, 2000:

```
SETOLOC %%A = 001130
SETOLOC %%AMON = 11
```

%%$TIMEINT Function

The %%$TIMEINT function calculates the time interval between two given times, specified in any order.

Format:

```
%%$TIMEINT time1 time2
```

time1 and *time2* are constants or variables in yydddhhmmss format.

where:

- *yy* is a 2-digit year
- *ddd* is a Julian day
- *hh* is the number of hours
- *mm* is the number of minutes
- *ss* is the number of seconds

The resulting time interval is in format:

- *ddddd* is the number of days
- *hh* is the number of hours
- *mm* is the number of minutes
- *ss* is the number of seconds

Example

```
SETOLOC %%A = %%$TIMEINT 01120070000 01119060000
```

The result is: 00001010000.

%%$PARSE Function

The %%$PARSE function is a powerful tool that offers extensive string manipulation capabilities in the AutoEdit environment. This function, which is similar to the REXX PARSE command in the TSO/E environment, can be used to analyze and extract information from various AutoEdit strings.

The %%$PARSE function parses a specified string, that is, the %%$PARSE function splits the specified string into substrings, according to a specified template. A template consists of variables and "patterns" that determine the parsing process.

Format:

```
SETOLOC %%$PARSE strng template
```

where:

- *strng* is the AutoEdit variable that contains the string to be parsed.
- *template* is the AutoEdit variable or constant that contains the template.

Example

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 A3 A4 A5
SETOLOC %%$PARSE %%S %%T
```

The %%$PARSE function assigns substrings of the specified string to the specified variables according to the specified template.

The SETOLOC statements in the above example provide the same result as the following statements:

```
SETOLOC %%A1 = THIS
SETOLOC %%A2 = IS
SETOLOC %%A3 = A
SETOLOC %%A4 = SAMPLE
SETOLOC %%A5 = STRING
```

The parsing process involves the following stages:

1. The string is broken into substrings, from left to right, using the patterns in the template.

2. Each substring is parsed into words, from left to right, using the variable names in the template.

   Template elements are:

   - string patterns
   - position patterns
   - variables
   - place holders (dummy variables)

The rules of parsing are detailed below.

Parsing Words

Scanning is performed from left to right and words in the string, leading and trailing blanks excluded, are matched one by one with the variables named in the template. The last variable named in the template contains the remaining part of the string, including leading and trailing blanks.

Up to 30 variable names can be specified in a parsing template.

The following situations may be encountered:

- The number of words in the string matches the number of variables in the template: Each of those variables contains one word of the string. The last variable contains the last word in the string including leading and/or trailing blanks.
- The number of words in the string is smaller than the number of variables named in the template: The first variables each contain one word of the string and the extra variables receive a value of NULL (a string of 0 character length).
- The number of words in the string is greater than the number of variables in the template. With the exception of the last, all variables contain one word of the string. The last variable named in the template contains the remaining part of the string, including leading and trailing blanks.

Example

The statements below, which include a %%$PARSE function:

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 A3
SETOLOC %%$PARSE %%S %%T
```

have the same result as the following statements:

```
SETOLOC %%A1 = THIS
SETOLOC %%A2 = IS
SETOLOC %%A3 = A SAMPLE STRING
```

Using Dummy Variables (Place Holders)

A single period can be used as a dummy variable in the template. This is useful when the corresponding word in the string does not need to be stored in a named variable.

Example

The following statements, which include a %%$PARSE function:

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = . . . A4.
SETOLOC %%$PARSE %%S %%T
```

have the same result as the following statement:

```
SETOLOC %%A4 = SAMPLE
```

Using Patterns in Parsing

Patterns can be included in the template. Their purpose is to divide the string into substrings prior to the process of parsing into words. Parsing is then performed, as previously described, on the substrings and not on the original string.

The following types of patterns are available:

- string pattern, a character string delimited by quotes, to distinguish it from a variable name
- number, signed or unsigned

Using String Patterns

The string is scanned from left to right for a substring that matches the string pattern.

The following situations may occur:

1. A match is found, that is, a substring within the string is identical to the given string pattern:

   The original string is divided into two substrings. The first substring, up to, but not including, the string pattern, is parsed into words using the variables named before the string pattern on the template. Parsing continues from the character following the matched string.

Example

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 'SAMPLE' A3 A4 A5
SETOLOC %%$PARSE %%S %%T
```

A match is found since the string SAMPLE is part of the original string.

System variable %%$PARSRC can be used to check if all strings specified in the template were matched during the parsing process, which was described in "Parsing Words" on page 93.

The original string is divided into two substrings while the matched part of the string is excluded. Parsing of the first substring uses the variables listed before the match on the template while parsing of the second substring uses the variables listed after the match:

First substring: THIS IS A

As a result of parsing:

```
A1=THIS
A2=IS A
```

Second substring: STRING

As a result of parsing:

```
A3=STRING
A4=NULL
A5=NULL
```

2. A match is not found, there is no substring identical to the given string pattern within the string:

   It is assumed that a match is found at the end of the string. The first substring consists of the entire string and it is parsed using only the variables named before the string pattern on the template. Parsing continues from the character following the matched string, the end of the string, in this case.

Example

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 A3 'EASY' A4 A5
SETOLOC %%$PARSE %%S %%T
```

A match is not found since the string EASY does not exist in the original string.

First substring: THIS IS A SAMPLE STRING

As a result of parsing:

```
A1=THIS
A2=IS
A3=A SAMPLE STRING
```

Second substring: NULL

As a result of parsing:

```
A4=NULL
A5=NULL
```

Using Numeric Patterns Within the Template

Numeric patterns are numbers that mark positions within the string. They are used to break the original string into substrings at the position indicated by the number.

The position specified can be absolute or relative:

- An absolute position is specified by an unsigned number.
- A relative position is specified by a signed number (positive or negative). It determines a new position within the string, relative to the last position.

Last position refers to one of the following:

- The start of the string (position 1) if last position was not specified previously.
- The starting position of a string pattern if a match was found.
- The end of the string if the string pattern was not matched.
- The last position specified by a numeric pattern.

If the specified position exceeds the length of the string, the numeric pattern is adjusted to the end of the string. Similarly, if the specified position precedes the beginning of the string (negative or zero numeric pattern), the beginning of the string is used as the last position.

Example 1

A parsing template with an absolute numeric pattern:

```
SETOLOC %%S =THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 11 A3 A4 A5
SETOLOC %%$PARSE %%S %%T
```

First substring: THIS IS A (up to, not including, position 11)

As a result of parsing:

```
A1=THIS
A2=IS A
```

Second substring: SAMPLE STRING (from position 11, to the end of the string).

As a result of parsing:

```
A3=SAMPLE
A4=STRING
A5=NULL (0 length string)
```

Example 2

A parsing template with a relative numeric pattern:

```
SETOLOC %%S =THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 +10 A3 A4 A5
SETOLOC %%$PARSE %%S %%T
```

Last position is the beginning of the string (position 1).

Position marked within the string is 1 + 10 = 11.

First substring: THIS IS A (up to, not including, position 11)

As a result of parsing:

```
A1=THIS
A2=IS A
```

Second substring: SAMPLE STRING (from position 11, to the end of the string).

As a result of parsing:

A3=SAMPLE
A4=STRING
A5=NULL (0 length string)

Using More Than One Pattern and Combining Pattern Types in the Template

Both types of patterns (string and numeric) can be intermixed in the same template. Up to 30 patterns and up to 30 variable names can be specified.

Scanning of the string proceeds from the beginning of the string until the first pattern (if any).

1. String pattern – a match was found:

   The substring that precedes the match with the pattern is parsed using the variables named in the template before the pattern, with the last variable receiving the end of the substring, including leading or trailing blanks.

2. String pattern – a match was not found:

   Since no match was found within the string, it is assumed that a match is found at the end of the string. The whole string is parsed using only the variables named in the template before the pattern.

3. Numeric pattern (absolute).

   The absolute numeric pattern points to a position within the string. The beginning of the string is position 1.

   The string is divided into two substrings.

   The first substring extends from the beginning of the string and up to, but not including, the position that corresponds to the numeric pattern. This substring is parsed using the variables named in the template before the pattern.

   If the absolute numeric pattern specifies a position beyond the length of the string, it is readjusted to the first position beyond the length of the string and the entire string is parsed using the variables named in the template before the pattern.

4. Relative numeric pattern:

   The relative numeric pattern (a signed number) specifies a position within the string, relative to the last position.

5. Last position:

   If the relative numeric pattern is the first pattern within the template, the last position is the beginning of the string.

   If the relative numeric pattern is not the first pattern within the template and the previous pattern was numeric, the last position is that specified by the previous numeric pattern.

   If the relative numeric pattern is not the first pattern within the template and the previous pattern was a string, the last position is that of the starting character of the match (if there was a match) or the position following the end of the string (if there was no match).

As a result of what was just explained

- If a pattern was not matched until the end of the string and the following pattern is a string pattern, this new string pattern is ignored since the starting point for the new scan is the end of the string.
- If a pattern was not matched until the end of the string and the following pattern is a numeric pattern, then the scan and subsequent parsing resume from the new position indicated by that numeric pattern.

Example 1

A parsing template with two absolute numeric patterns, with the second position preceding the first:

The following statements:

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 A2 11 A3 6 A4
SETOLOC %%$PARSE %%S %%T
```

have the same result as the following DO SET statements:

```
SETOLOC %%A1 = THIS
SETOLOC %%A2 = IS A
SETOLOC %%A3 = SAMPLE STRING
SETOLOC %%A4 = IS A SAMPLE STRING
```

First substring: THIS IS A (up to, not including, position 11)

As a result of parsing:

```
A1=THIS
A2=IS A
```

Second substring: SAMPLE STRING from position 11 and up to the end of the string. Because the next pattern, position 6, precedes the previous position, it cannot limit this second substring.

As a result of parsing:

```
A3=SAMPLE STRING
```

Third substring: IS A SAMPLE STRING (from position 6 to the end of the string)

As a result of parsing:

```
A4=IS A SAMPLE STRING
```

Example 2

A parsing template with one absolute and one relative numeric pattern:

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 6 A2 +3 A3
SETOLOC %%$PARSE %%S %%T
```

First substring: THIS (beginning of the string up to, not including, position 6).

As a result of parsing:

```
A1=THIS
```

Second substring: IS (from position 6 up to, not including, position 6+3=9)

As a result of parsing:

```
A2=IS
```

Third substring: A SAMPLE STRING (from position 9 to the end of the string)

As a result of parsing:

```
A3=A SAMPLE STRING
```

Example 3

A parsing template with two relative numeric patterns:

The following statements:

```
SETOLOC %%T = A1 A2 +40 A3 -13 A4 A5
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%$PARSE %%S %%T
```

have the same result as the following statements:

```
SETOLOC %%A1 = THIS
SETOLOC %%A2 = IS A SAMPLE STRING
SETOLOC %%A3 = %%NULL
SETOLOC %%A4 = SAMPLE
SETOLOC %%A5 = STRING
```

The first numeric pattern specifies a position at column 40. This is beyond the end of the string so position is reset to column 24 (end of string + 1). As a result, the entire string is parsed to words using variables A1 and A2.

The second numeric pattern specifies a position at column 11 (end of the string + 1 minus 13) that precedes the position (40 readjusted to 24) previously specified. Therefore, the data from the last position, to the end of the string is parsed to words using variable A3 (A3 is set to NULL).

The data (from column 12 to the end of the string) is parsed to words using variables A4 and A5.

Example 4

Combining string pattern and numeric pattern:

The following statements:

```
SETOLOC %%S = THIS IS A SAMPLE STRING
SETOLOC %%T = A1 `A' A2 +3 A3
SETOLOC %%$PARSE %%S %%T
```

have the same result as the following statements:

```
SETOLOC %%A1 = THIS IS
SETOLOC %%A2 = A S
SETOLOC %%A3 = AMPLE STRING
```

The pattern specifies a string (A) that is matched at column 9. The data before column 9 is parsed to words using variable A1. The numeric pattern (+3) specifies a position at column 12 by using relative position. The data from column 9 to column 12 is parsed to words using variable A2. The remaining data (from column 12 to the end of the string) is parsed to words using variable A3.

KOA VTAM Exception Codes

KOA VTAM Exception codes are listed and described below.

NOTE

%VTAMFDBK (if specified) contains information supplied by VTAM applications in the hexadecimal representation of two full words (16 characters). These two values (usually RPL and ACB error information) are determined by the specific error as described below.

For example, whenever RTNCD and FDBK2 are specified, the first 8 characters contain the hexadecimal representation of the RPL RTNCD field, and the next 8 characters contain the representation of the RPL FDBK2 field. If R15 and R0 are specified, the first 8 characters contain the hexadecimal representation of the contents of Register 15, and the next 8 characters contain the representation of the contents of Register 0.

These codes and their descriptions can be found in the IBM VTAM Programming manual.

Table 32 KOA VTAM Exception Codes (part 1 of 4)

%VTAMRC	%VTAMERR	%VTAMFDBK	Description
0	OK		The last operation completed successfully.
4	UNSOLMSG		An unsolicited (unexpected) message was received. The message can either be accepted (by means of the GETSCREEN command) or rejected (by re-sending the last screen).
8	NODATA		The GETSCREEN command did not receive any new messages from the VTAM application.

Table 32 KOA VTAM Exception Codes (part 2 of 4)

%VTAMRC	%VTAMERR	%VTAMFDBK	Description
12	TIMEOUT		Response was not received from the VTAM application for *n* seconds. The maximum wait period is specified by the TIMEOUT command or by the KOATIME installation parameter.
16	ERROR-EXIT	RTNCD, FDBK2	A VTAM error exit (SYNAD or LERAD) was activated because of an exception situation. The RTNCD and FDBK2 RPL fields determine the specific error situation.
20	NSEXIT		The NSEXIT VTAM Exit was activated because of a network services exception situation.
24	SESS-NOT-ACT		A KOA communication command failed because the session was no longer active.
28	VTAMREQ	R15 R0	A VTAM command (for example, SEND, RECEIVE) was rejected by the VTAM application. The values of registers 15 and 0 determine the specific error.
32	NOSTORAGE		KOA processing could not continue due to insufficient storage. The REGION parameter (in the JCL procedure) should be increased, and the KOA script should be reactivated.
36	OPEN	ACB error flag	An error occurred while opening the ACB. The error flag indicates the specific reason for the failure.
40	SLUNOTDEF		The terminal name specified in the LOGON command was not defined to VTAM. No new sessions can be initiated.
44	NOFREESLU		Selection of a KOA terminal (SLU) failed because all defined terminals were currently in use; no new sessions can be initiated.

Table 32 KOA VTAM Exception Codes (part 3 of 4)

%VTAMRC	%VTAMERR	%VTAMFDBK	Description
48	NOTACCEPT		An attempt was made to initiate a session with a non-available VTAM application or a VTAM application which is not ready to accept any new sessions.
52	SETLOGON	RTNCD, FDBK2	The SETLOGON VTAM command (which enables KOA to start sessions) could not be executed. The RTNCD and FDBK2 RPL fields determine the specific error situation.
56	REQSESS	RTNCD, FDBK2	The REQSESS VTAM command (which initiates a session with a VTAM application) could not be executed. The RTNCD and FDBK2 RPL fields determine the specific error situation.
60	BIND-REJECT		The specified parameters prevent session initiation. Check the validity of the supplied LOGMODE.
64	RECEIVE	RTNCD, FDBK2	The RECEIVE VTAM command (which accepts a message from a VTAM application) could not be executed. The RTNCD and FDBK2 RPL fields determine the specific error situation.
68	INVALID-DATA		A message (from the VTAM application) which contains invalid data was received. KOA cannot display the data.
72	UNKNOWN-CMD		A message (from the VTAM application) which contains an unknown 3270 command was received. KOA cannot display the data.
76	SEND	RTNCD, FDBK2	The SEND VTAM command (send a message to the VTAM application) could not be executed. The RTNCD and FDBK2 RPL fields determine the specific error situation.
80	KEYLOCKED		The keyboard was locked. Messages cannot be sent to the VTAM application.

Table 32 KOA VTAM Exception Codes (part 4 of 4)

%VTAMRC	%VTAMERR	%VTAMFDBK	Description
84	QUIESCED		The session was halted (as requested by the application). Messages cannot be sent to the VTAM application.
88	CLOSE	ACB error flag	An error occurred while closing the ACB. The error flag indicates the specific reason for the failure.
92	INVREQ		An invalid request from the KOA session handler caused an internal error.

Appendix C

Sample KeyStroke Reports and Utilities

The IOA KSL and IOA SAMPLE libraries contain BMC Software-supported and customer-contributed examples of KSL scripts, respectively.

Two types of reports are available:

- Reports produced in batch by KSL scripts. These are listed later in this chapter, and samples of supported KSL scripts are located in the IOA KSL library.
- Special reports that cannot readily be produced using the Online facility or KSL. These are produced by utilities that are described in the *INCONTROL for z/OS Utilities Guide*.

Some reports are produced from information in the IOA Log file. Other reports are produced from the Active Jobs file, Jobs Statistics file, Job Network file and from tables.

NOTE

If you choose to modify an existing sample report or utility, BMC Software recommends that you save the changed report under a different name and keep the original report unchanged. This precaution can help in error detection if the altered KSL script does not run as expected.

Sample KSL Report Outputs

The following sample outputs are the result of running job REPJBDEF (in the Control-M JCL library), which invokes sample KSLREPSCHED (in the IOA KSL library).

Output #1 from the sample KSL Script:

Figure 16 Output from KSL Library Sample KSLREPSCHED

```
KEYSTROKE REPORTING LANGUAGE   (REL PROD.)        DATE 06/06/01 TIME 10.08 PAGE 000001
           LIST OF JOBS  TABLE: PRODYH  LIB: CTM.PROD.SCHEDULE

================================================================================
--------------------------------------------------------------------------------
       PRODYIDK   UPDATE # 1
       PRODYHST   UPDATE # 2
       PRODYJCL   CREATE INPUT FILES # 2
       PRODYBTL   REPORTS FOR BRANCHES
       PRODYHTK   PROCESS INPUT DATA FOR PRODYHST
       PRODYHC2   CREATE INPUT FILE # 2
       PRODYBCK   PROCESS INPUT DATA FOR PRODYIDK
       PRODYIZN   REPORTS FOR BRANCH MANAGERS
       PRODYEND   REPORTS FOR MAIN OFFICE
       PROJYFOT   BEGIN OF EVENING PROCESS
       PROJYMRG   EVENING UPDATE PROCEDURE
       PROJYMTI   VERIFICATION PROCESS OF EVENING UPDATE
       PROJYHO1   SPECIAL CALCULATIONS FOR ACCOUNTING DEPARTMENT
       PROJYHO2   REPORTS FOR ACCOUTING DEPARTMENT
       PROJYDPY   UPDATE OF ON-LINE FILES
       PROJYDTK   REPORTS OF ON-LINE FILES
       PROJYDLI   CREATE DUAL ON-LINE FILE
       PROYH11    YH APPLICATION UPDATE
       PROJYFIN   CLEAN-UP FOR YH APPLICATION
       PROJYBNK   VERIFICATION OF BRANCH BALANCES
       PROJEND    FINAL YH APPLICATION PROCEDURE
       PROLYPAR   NIGHT INPUT COLLECTION # 1
       PROLYDOC   BACKUP FILES STATUS REPORTS
       PROLYFMZ   REPORTS FOR MAIN OFFICE
       PROLYDEL   DELETE TEMPORARY "REPORT" FILES
       PROLYBME   CREATE EXTERNAL TAPE
       PROLYDM2   ARCHIVE YH APPLICATION DATA SETS #2
       PROLYDM1   ARCHIVE YH APPLICATION DATA SETS #1
 ======= >>>>>>>>>>>>>>>>>>>    NO MORE JOBS IN TABLE      <<<<<<<<<<<<<<<< =======
```

Output #2 from the sample KSL Script:

```
BMC SOFTWARE, INC. IOA KEY-STROKE REPORTING LANGUAGE (REL PROD.)DATE 06/06/01 TIME
10.12
PAGE 000001

         SCHEDULE DEFINITION OF MEMBER PRODYBCK  IN TABLE PRODYH   LIBRARY
CTM.PROD.SCHEDULE
         ======================================
 +-------------------------------------------------------------------------------+
 | MEMNAME PRODYBCK    MEMLIB   CTM.PROD.SCHEDULE                                |
 | OWNER   M44         TASKTYPE JOB    PREVENT-NCT2 Y DFLT  N                    |
 | APPL    APPL-L                      GROUP BKP-PROD-L                          |
 | DESC    DAILY BACKUP OF SPECIAL FILES FROM APPL-L                             |
 | OVERLIB CTM.OVER.JOBLIB                                                       |
 | SET VAR                                                                       |
 |.CTB STEP AT         NAME            TYPE                                      |
 | DOCMEM  BACKPL02    DOCLIB   CTM.PROD.DOC                                     |
 | ============================================================================= |
 | DAYS                                                          DCAL            |
 |                                                                    AND/OR     |
 | WDAYS                                                         WCAL            |
 | MONTHS  1- Y 2- Y 3- Y 4- Y 5- Y 6- Y 7- Y 8- Y 9- Y 10- Y 11- Y 12- Y        |
 | DATES                                                                         |
 | CONFCAL WORKDAYS SHIFT       RETRO N MAXWAIT 04  D-CAT                        |
 | MINIMUM          PDS                                                          |
 | ============================================================================= |
 | IN       START-DAILY-BACKUP   ODAT                                            |
 | CONTROL                                                                       |
 | RESOURCE INIT                 0001      CART                 0001             |
 | PIPE                                                                          |
 | TIME: FROM       UNTIL       PRIORITY     DUE OUT       SAC      CONFIRM    |
 | ============================================================================= |
 | OUT      BAKCKPL02-ENDED-OK   ODAT +                                          |
 | AUTO-ARCHIVE Y          SYSDB    Y      MAXDAYS      MAXRUNS                 |
 | RETENTION:  # OF DAYS TO KEEP 030  # OF GENERATIONS TO KEEP                   |
 | SYSOUT OP   (C,D,F,N,R)                                             FROM      |
 | MAXRERUN     RERUNMEM                                 INTERVAL       FROM     |
 | STEP RANGE         FR (PGM.PROC)          .          TO          .           |
 | ON PGMST          PROCST          CODES                                  A/O |
 |   DO                                                                          |
 | SHOUT WHEN                                        TO                  URGN    |
 |   MS                                                                          |
 ======= >>>>>>>>>>>>>>>>>>> END OF SCHEDULING PARAMETERS <<<<<<<<<<<<<<< =====
```

KSL Library Scripts

This table lists the KSL library scripts and their associated products, libraries, and JOB members. If an associated job member is not listed for a script, a standard KSL procedure can be used for that script.

You can use these scripts to design scripts for your own report utilities.

NOTE

In addition to the scripts in the IOA KSL library, the IOA SAMPLE library contains many other useful scripts. However, the scripts in the IOA SAMPLE library have been developed and supplied by users. They have been placed in the IOA SAMPLE library as examples. They have not been tested and they are not supported.

Script	Product	Library	JOB member in xxx.JCL	Description
ADDCOND7	IOA	KSL	ADDMNCND	Manual operations to be performed at night (Previously called REP7COND).
ADDMAYBE	CTM	KSL	MAYBEJOB	A general utility routine to add maybe conditions.
ADDMNCND	IOA			Add manual conditions based on prefix.
BJR5ALL	CTB			All Control-M/Analyzer Log messages during a specified period.
BRPBLEFT	CTB	KSL		All Control-M/Analyzer missions that have not yet been scheduled, and a description for the problem.
BRPCOMM	CTB	KSL		Values of variables committed to the database for each rule invocation.
BRPGRP	CTB	KSL		Site-defined application groups and their current mode of implementation.
BRPINV	CTB	KSL		Status of all jobs currently in the Active Balancing file.

Script	Product	Library	JOB member in xxx.JCL	Description
BRPJPRT	CTB	KSL		Printout of the invocation report of a specific rule.
BRPRLBK	CTB	KSL		Roll back a specific Rule Activity display entry.
BRPRLDEF	CTB	KSL		Printout of specified rule definitions.
BRPROLL	CTB	KSL		Roll back a Control-M/Analyzer invocation from a specified job step.
BUTCRGRP	CTB	KSL		Create a group in the Control-M/Analyzer database.
BUTCRVAR	CTB	KSL		Create a variable definition under a specified group in the Control-M/Analyzer database.
BUTMSORD	CTB	KSL		Order a Control-M/Analyzer balancing mission.
CTDSCREN	CTD	SAMPLE		Printout of all Control-V screens (for documentation and training).
CTOALOR1	CTO	SAMPLE		Lists the Automation Log messages for the specified date (Previously called REPOALO).
CTTEXPV	CTT	SAMPLE		Uses Control-M/TAPE API in order to expire (make scratch) volumes. the program reads the input volumes from the SYSIN DD.
DHLDGRP1	CTD	SAMPLE		Hold a group of reports.
HOLDELGR	CTM	KSL	DELGROUP	Hold and delete all entries related to a certain Control-M group.
HOLDGRP	CTM	KSL	HOLDGRUP	Hold a group of jobs in the AJF.
HOLDGRUP	CTM			Hold a group of jobs.
MAYBEJOB	CTM			Add prerequisite conditions for maybe jobs.
ORDERBPR	CTD	SAMPLE		Order or force backup, printing or restore missions.
ORDERREP	CTD	SAMPLE		Order or force report decollating missions.
ORDERRUL	CTO	SAMPLE		Order a specified rule (Previously called RULORDER).
PRINTREP	CTD CTV	SAMPLE		Print specific reports by specifying report name, index value and date.
REP3GRUP	CTM			Status of all the jobs of specified groups.

Script	Product	Library	JOB member in xxx.JCL	Description
REP3LEFT	CTM			All jobs still in the Active Job File that did not run during the previous night (wait schedule, ended NOT OK, executing) and the reasons for the problems.
REP3STAT	CTM			Statistical summary of what must be done tonight, or job status in the morning.
REP3TAPE	CTM			Status of all jobs using tapes.
REP3WHY	CTM			All jobs in the Active Jobs file having a WAIT SCHEDULE status.
REP5ABND	IOA			All abends in a given period.
REP5ALL	IOA			All IOA Log file messages of a specified period.
REP5MSGD	IOA			All IOA Log file messages of specified message codes for a specific period.
REPCALND	IOA	SAMPLE		Printout of calendar definitions (Previously called REP8CAL).
REPJOBAV	CTM	KSL	REP3AVER	This report prints job statistical information for all or selected jobs currently residing on the AJF.
REPJOBGR	CTM	KSL	REP3GRUP	This report prints the status of all jobs on the AJF belonging to specified groups.
REPJOBMO	CTM	KSL	REP3LEFT	Print all the jobs left (WAIT SCHEDULE, ended not ok, executing) on the active jobs file, and a description for the problem.
REPJOBMO	CTM	KSL	CTMPSIMS	Skeleton for simulation job and tape pull list.
REPJOBRS	CTM	KSL	REPJOBRS	Last night restart history report.
REPJOBSA	CTM	KSL	REP3STAV	Print list of jobs that exceeded average elapsed time.
REPJOBSS	CTM	KSL	REP3STAT	Print a statistical summary of what should be done tonight, or job status the next day.
REPJOBSY	IOA	SAMPLE		
REPJOBTP	CTM	KSL	REP3TAPE	Prints the status of all jobs on the AJF using a tape.
REPJOBWH	CTM	KSL	REP3WHY	Print list of jobs which are in WAIT SCHEDULE status.

Script	Product	Library	JOB member in xxx.JCL	Description
REPLGGRD	CTD	SAMPLE		Print all log messages for specific groups.
REPLGMSG	IOA	KSL	REP5MSGD	Print all log messages for specific messages for a specific period of time.
REPLOGAB	IOA	KSL	REP5ABND	Print all ABENDS in a period.
REPLOGAL	IOA	KSL	REP5ALL	All IOA Log file messages of a specified period.
REPLOGAL	IOA	KSL	REP5ALL	Print all the log messages for a specific date.
REPLOGCN	IOA	SAMPLE		Print all log events associated with messages.
REPLOGEM	CTM	SAMPLE		Print all the emergency (recovery) jobs which were activated in a period.
REPLOGEX	IOA	SAMPLE		Log - exception report.
REPLOGGR	CTR	SAMPLE		Print all log messages of specified groups.
REPLOGMN	IOA	SAMPLE		Print all log messages of type user.
REPLOGNC	CTM	SAMPLE		Print all 'NOT CATLGD 2' log events in a period.
REPLOGRS	CTM	SAMPLE		Print all log events associated with messages.
REPLOGST	IOA	SAMPLE		Night exception report (Previously called REP5EXP).
REPLOGST	IOA	SAMPLE		Print job statistics at night.
REPLOGUS	IOA	SAMPLE		Print all log messages of specified users.
REPOJHST	CTO	SAMPLE		List events and important messages for specified jobs and started tasks. Input to the report is the REPJHIST rule table in the Control-O RULES library (Previously called REPJHSJ).
REPRERUN	CTM	KSL	RERUNJOB	Rerun a specified job.
REPRULED	CTO	SAMPLE		Prints specified rule definitions (Previously called REPRLDEF).
REPSCHED	CTM	KSL	REPJBDEF	Report routine to print job schedules.
REPUPRT	CTD	KSL		Print a report via IMMEDIATE/DEFERRED print option in the USER screen.
REPUSRHP	CTD	KSL		Print a list of the user's reports currently available in the history file.

Script	Product	Library	JOB member in xxx.JCL	Description
REPUSRHR	CTD	KSL		Change retention period of reports of specified job from the history file.
REPUSRPP	CTD	KSL		Print a list of user's reports currently available for online viewing. The reports will be requested to PRINT DEFERRED (with a PRINTING MISSION).
REPUTREE	CTD	SAMPLE		Create a recipient tree from a given report.
REPUURUL	CTD	SAMPLE		List of the user's currently defined rulers.
REPUUSRA	CTD	SAMPLE		List of all users who may receive ■ a report from a specified job (including the names of the reports) ■ list of all users who may receive a specific report. ■ list of all user's reports currently available for online viewing, and their status, line count and page count.
REPUUSRP	CTD	SAMPLE		List of all reports a user may receive at any time.
REPVOLES	CTT	SAMPLE		List of all or selected volumes in the Media Database.
RPDAMSCH	CTD	SAMPLE		Status of all missions currently in the Active Missions file (such as tonight's schedule or last night's schedule).
RPDGLBC	CTD	SAMPLE		Perform global changes to the IN condition in report decollating missions.
RPDJOBGR	CTD	SAMPLE		Status of all report decollating missions of specified groups.
RPDJOBMO	CTD	SAMPLE		All missions remaining in the morning (WAIT SCHEDULE, Ended NOTOK), with explanation.
RPDJOBUS	CTD	SAMPLE		Status of all the reports (jobs) to be decollated for specified user IDs.
RPDPSCHD	CTD	SAMPLE		Printout of printing mission definitions.
RPDRERUN	CTD	SAMPLE		Rerun a report decollating mission.

Script	Product	Library	JOB member in xxx.JCL	Description
RPDSCHED	CTD	SAMPLE		Printout of report decollating mission definitions.
RPRSV	CTM	SAMPLE		Print all restart messages for a given period.
RPTABRUL	CTO	SAMPLE		Cross-references all tables and rules for each message (Previously called REPOXRFM).
UPDACTC	CTD	SAMPLE		Update the number of copies of a report.
UPDRPRT	CTD	SAMPLE		This KSL delete oldest duplicate reports those have the same user, jobname and report name.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

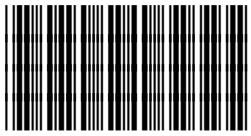
T

U

V

W

Notes



450677