IBM /Technical Newsletter

This Newsletter No.	SN28-1293
Date	February 10, 1989
Base Publication No.	SC28-1883-0
File No.	S370-39
Prerequisite Newsletters/ Supplements	None

TSO Extensions Version 2 REXX Reference

TSO Extensions Version 2, Program Number 5685-025

This newsletter contains replacement pages for *TSO Extensions Version 2 REXX Reference* in support of TSO Extensions Version 2, Program Number 5685-025.

Before inserting any of the attached pages into *TSO Extensions Version 2 REXX Reference* read ***carefully*** the instructions on this cover. They indicate when and how you should insert pages.

Pages to be Removed	Attached Pages to be Inserted*
Cover - Edition Notice	Cover - Edition Notice
iii - xiv	iii - xiv
51 - 52	51 - 52
247 - 248	247 - 248
271 - 272	271 - 272.2
425 - 426	425 - 426
431 - 448	431 - 448

*If you are inserting pages from different Newsletters/Supplements and ***identical*** page numbers are involved, always use the page with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This newsletter documents the following new and changed information for TSO/E Version 2 support of the REXX programming language:

- New information about how to initialize a language processor environment if you use a user-written terminal monitor program (TMP)
- New values returned by the PARSE VERSION instruction, for example, the language level description and language processor release date. The new values support APAR OY17590 and are returned ***only*** if you install the PTF that supports the APAR.

This newsletter also includes minor technical changes.

Note: Please file this cover letter at the back of the publication to provide a record of changes.

IBM Corporation, Information Development, Dept. D58, Building 921-2, P.O. Box 950, Poughkeepsie, New York 12602

 Printed in U.S.A.

TSO Extensions Version 2
REXX Reference

SC28-1883-0

First Edition (December 1988)

This edition with Technical Newsletter SN28-1293 applies to the TSO Extensions (TSO/E) Version 2 Licensed Program, Program Number 5685-025, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product in this publication is not intended to state or imply that only IBM's product may be used. Any functionally equivalent product may be used instead. This statement does not expressly or implicitly waive any intellectual property right IBM may hold in any product mentioned herein.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921, PO Box 950, Poughkeepsie, New York 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Before Using the Information in This Book

Before you use the information in this book, please read "Summary of Changes" on page 425. This topic lists the instructions, functions, and services described in this book that support various APARs.

Contents

PARSE NUMERIC

The current numeric controls (as set by the NUMERIC instruction, see page 47) are made available. These controls are in the order DIGITS FUZZ FORM.

Example:

```
After: Parse Numeric Var1
Var1 would be equal to: 9 0 SCIENTIFIC
```

See Numeric instruction on page 47. Also refer to the built-in functions DIGITS, FORM, and FUZZ found on pages 87, 90, 91, respectively.

PARSE PULL

The next string from the queue is parsed. If the queue is empty, lines will be read from the default input (typically the user's terminal). Data can be added to the head or tail of the queue by using the PUSH and QUEUE instructions respectively. The number of lines currently in the queue can be found by using the QUEUED built-in function, described on page 97. The queue will remain active as long as the language processor is active. The queue can be altered by other programs in the system and can be used as a means of communication between these programs and programs written in REXX.

Note: PULL and PARSE PULL read from the data stack. If that is empty, they read from the terminal (TSO/E address space) or from the data set that represents the input stream (non-TSO/E address space). See the PULL instruction on page 55 for further details.

PARSE SOURCE

The data parsed describes the source of the program being executed.

The source string contains the following tokens:

1. The characters TSO
2. The string COMMAND, FUNCTION, or SUBROUTINE depending on whether the program was invoked as some kind of host command (for example, as an exec from TSO/E READY mode), or from a function call in an expression, or via the CALL instruction.
3. Name of the exec in uppercase. If the name is not known, this token is a question mark (?).
4. Name of the DD from which the exec was loaded. If the name is not known, this token is a question mark (?).
5. Name of the data set from which the exec was loaded. If the name is not known, this token is a question mark (?).
6. Name of the exec as it was invoked, that is, the name is not folded to uppercase. If the name is not known, this token is a question mark (?).
7. Initial (default) host command environment in uppercase. For example, this token may be TSO, MVS, or ISPEXEC.
8. Name of the address space in uppercase. For example, the value may be MVS (non-TSO/E) or TSO/E or ISPF. If the exec was invoked from ISPF, the address space name is ISPF.

   The value is taken from the parameter block (see page 280). Note that the initialization exit routines may change the name specified in the parameters module. If the name of the address space is not known, this token is a question mark (?).

9. Eight character user token. This is the token that is specified in the PARSETOK field in the parameters module (see page 277).

For example, the string parsed might look like one of the following:

```
TSO COMMAND PROGA SYSXR07 EGGERS.ECE.EXEC ? TSO TSO/E ?

TSO SUBROUTINE PROGSUB SYSEXEC ? ? TSO ISPF ?
```

PARSE VALUE

expression is evaluated, and the result is the data that is parsed. Note that WITH is a subkeyword in this context and so cannot be used as a symbol within *expression*.

Thus, for example:

```
PARSE VALUE time() WITH  hours ':' mins ':' secs
```

will get the current time and split it up into its constituent parts.

PARSE VAR *name*

The value of the variable specified by *name* is parsed. *name* must be a symbol that is valid as a variable name (that is, it can not start with a period or a digit). Note that the variable name may be included in the template, so that for example:

```
PARSE VAR string word1 string
```

will remove the first word from *string* and put it in the variable *word1*, and

```
PARSE UPPER VAR string word1 string
```

will also translate the data from *string* to uppercase before it is parsed.

PARSE VERSION

Information describing the language level and the date of the language processor is parsed. This consists of five words:

- A word describing the language, which is the string "REXX370"
- The language level description, for example, "3.45" or "3.46"
- Three tokens describing the language processor release date in the format as the default for the DATE() function (see page 85), for example, "20 Oct 1987" or "30 Jun 1988".

The values returned for the language level description and the language processor release date depend on whether or not your installation has installed the PTF for APAR OY17590. If the PTF is installed, the values returned are "3.46" and "30 Jun 1988". If the PTF is not installed, the values returned are "3.45" and "20 Oct 1987".

Note: PARSE VERSION information should be parsed on a word basis rather than on an absolute column position.

Maintain Entries in the Host Command Environment Table (IRXSUBCM)

Use the IRXSUBCM routine to maintain entries in the host command environment table. The table contains the names of the valid host command environments that REXX execs can use to execute host commands. In an exec, you can use the ADDRESS instruction to direct a host command to a specific environment for execution. The host command environment table also contains the name of the routine that is invoked to handle the execution of commands for each specific environment. "Host Command Environment Table" on page 291 describes the table in more detail.

Note: To permit FORTRAN programs to call IRXSUBCM, TSO/E provides an alternate entry point for the IRXSUBCM routine. The alternate entry point name is IRXSUB.

Using IRXSUBCM, you can add, delete, update, or query entries in the table. You can also use IRXSUBCM to dynamically update the host command environment table while a REXX exec is executing.

A program can access IRXSUBCM using either the CALL or LINK macro instructions, specifying IRXSUBCM as the entry point name. You can obtain the address of the IRXSUBCM routine from the REXX vector of external entry points. "Format of the REXX Vector of External Entry Points" on page 328 describes the vector.

If a program uses IRXSUBCM, it must create a parameter list and pass the address of the parameter list in register 1.

IRXSUBCM changes or queries the host command environment table for the current language processor environment, that is, for the environment in which it executes (see "General Considerations for Calling TSO/E REXX Routines" on page 212 for information). IRXSUBCM affects only the environment in which it executes. Changes to the table take effect immediately and remain in effect until the language processor environment is terminated.

Environment Customization Considerations

If you use the initialization routine to initialize environments, on the call to IRXSUBCM, you can optionally pass the address of an environment block in register 0. If the environment block is valid, IRXSUBCM will execute in the environment represented by that environment block. If register 0 does not point to a valid environment block, IRXSUBCM will locate the current environment.

If the environment in which IRXSUBCM executes is part of a chain of environments and you use IRXSUBCM to change the host command environment table, the following applies:

- The changes do not affect the environments that are higher in the chain or existing environments that are lower in the chain.
- The changes are propagated to any language processor environment that is created on the chain after IRXSUBCM updates the table.

Entry Specifications

For the IRXSUBCM routine, the contents of the registers on entry are:

Register 0 Address of an environment block (optional)

Register 1 Address of the parameter list passed by the caller

Registers 2-12 Unpredictable

Register 13 Address of a register save area

Register 14 Return address

Register 15 Entry point address

Parameters

In register 1, you pass the address of a parameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter. You must pass all parameters on the call. The high order bit of the last address in the parameter list must be set to 1. Figure 29 describes the parameters for IRXSUBCM.

Figure 29. Parameters for IRXSUBCM

Parameter	Number of Bytes	Description
Parameter 1	8	The function to be performed. The name of the function must be left justified and padded to the right with blanks. The valid functions are: • ADD • DELETE • UPDATE • QUERY Each function is described after the table in "Functions."
Parameter 2	4	The address of a string. On both input and output, the string has the same format as an entry in the host command environment table. "Format of a Host Command Environment Table Entry" on page 249 describes the entry in more detail.
Parameter 3	4	The length of the string (entry) that is pointed to by parameter 2.
Parameter 4	8	The name of the host command environment. The name must be left justified and padded to the right with blanks.

Functions

Parameter 1 contains the name of the function IRXSUBCM is to perform. The functions are:

ADD

Adds an entry to the table using the values specified on the call. IRXSUBCM does not check for duplicate entries. If a duplicate entry is added and then IRXSUBCM is called to delete the entry, IRXSUBCM will delete the duplicate entry and leave the original one.

Using the Environment Block

The main control block that is created for a language processor environment is the environment block. The environment block represents the language processor environment and points to other control blocks that contain information about the environment.

The environment block is known as the *anchor* that is used by all callable interfaces to REXX. All REXX routines, except for the IRXINIT initialization routine, cannot execute unless an environment block exists, that is, a language processor environment must exist. When IRXINIT initializes a new language processor environment, it always returns the address of the environment block in register 0. (If you explicitly call IRXINIT, it also returns the address of the environment block in the parameter list.) You can also use the IRXINIT routine to obtain the address of the environment block for the current non-reentrant environment (see page 340). IRXINIT returns the address in register 0 and also in a parameter in the parameter list.

The address of the environment block is useful for calling a REXX routine or for obtaining information from the control blocks that were created for the environment. If you call any of the REXX routines (for example, IRXEXEC to execute an exec or the variable access routine IRXEXCOM), you can optionally pass the address of an environment block to the routine in register 0. By passing the address of an environment block, you can specify in which specific environment you want either the exec or the service to execute. This is particularly useful if you use the IRXINIT routine to initialize several environments on a chain and then want to execute a REXX routine in a specific environment. When you call the routine, you can pass the address of the environment block in register 0.

If you call a REXX routine and do not pass the address of an environment block in register 0, the routine will execute:

- In the last environment on the chain under the current task (non-TSO/E address space)
- In the last environment on the chain under the current task or a parent task (TSO/E address space).

If you call IRXEXEC or IRXJCL and an environment does not exist, IRXINIT is invoked to initialize an environment in which the exec will execute. When the exec completes processing, the newly created environment is terminated.

The environment block points to several other control blocks that contain the parameters used to define the environment and the addresses of REXX routines, such as IRXINIT, IRXEXEC, and IRXTERM, and replaceable routines. You can access these control blocks to obtain this information. The control blocks are described in "Control Blocks Created for a Language Processor Environment" on page 323.

Note About Changing Any Control Blocks

You can obtain information from the control blocks. However, you **must not change** any of the control blocks. If you do, unpredictable results may occur.

When Environments are Automatically Initialized in TSO/E

The initialization routine IRXINIT initializes a language processor environment. The system calls IRXINIT to automatically initialize a default environment when a user logs on to TSO/E and when ISPF is invoked.

When a user logs on to TSO/E, IRXINIT is called as part of the logon process to automatically initialize a language processor environment for the TSO/E session. The initialization of a language processor environment is transparent to the user. After users log on to TSO/E, they can simply invoke a REXX exec without performing any other tasks.

Note: If your installation uses a user-written terminal monitor program (TMP) instead of the TMP provided by TSO/E, a language processor environment is not automatically initialized. See "Initializing Environments for User-Written TMPs" on page 272.1 for information about the tasks you must perform to initialize a language processor environment in order to execute REXX execs.

Similarly, when a user invokes ISPF from TSO/E, the IRXINIT routine is called and automatically initializes a language processor environment for ISPF, that is, for the ISPF screen. The second language processor environment is separate from the environment that was initialized for the TSO/E session. If the user enters split screen in ISPF, IRXINIT initializes a third language processor environment for the second ISPF screen. At this point, three separate language processor environments exist. If the user executes a REXX exec from the second ISPF screen, the exec executes under the third language processor environment, that is, the environment IRXINIT initialized for the second ISPF screen. If the user executes the exec from the first ISPF screen, it runs under the second language processor environment.

The termination routine, IRXTERM, terminates a language processor environment. Continuing the above example, when the user returns to one screen in ISPF, the IRXTERM routine is called. IRXTERM terminates the third language processor environment that was initialized for the second ISPF screen. Similarly, when the user exits from ISPF and returns to TSO/E READY mode, IRXTERM terminates the language processor environment for the first ISPF screen. In TSO/E READY mode, the first language processor environment still exists. At this point, if the user executes a REXX exec from READY mode, the exec executes under the environment that was initialized at TSO/E logon. When the user logs off, IRXTERM terminates the language processor environment for the TSO/E session.

To summarize, the IRXINIT routine automatically initializes a language processor environment when a user logs on to TSO/E and whenever an ISPF screen is initialized. Each environment that is initialized is separate from another environment. The IRXTERM routine automatically terminates the language processor environment for an ISPF screen when the screen session ends and terminates the environment created at TSO/E logon when the user logs off.

You can also call the IRXINIT routine to initialize a language processor environment. On the call to IRXINIT, you specify values you want defined for the new environment. Using IRXINIT gives you the ability to define a language processor environment and *customize* how REXX execs execute and how system services are accessed and used. This is particularly important in non-TSO/E address spaces where you may want to provide replaceable routines to handle system services.

However, you may want to use IRXINIT in TSO/E in order to create an environment that is similar to a non-TSO/E address space to test any replaceable routines or REXX execs you have developed for non-TSO/E.

If you explicitly call IRXINIT to initialize a language processor environment, you must call the IRXTERM routine to terminate the environment. The system does not terminate language processor environments that you initialized by calling IRXINIT. Information about IRXINIT and IRXTERM is described later in this chapter. Chapter 15, "Initialization and Termination Routines" provides reference information about the parameters and return codes for IRXINIT and IRXTERM.

Initializing Environments for User-Written TMPs

If your installation uses a user-written terminal monitor program (TMP) instead of the TMP provided by TSO/E, a language processor environment is not automatically initialized in the TSO/E address space when a user logs on to TSO/E. That is, a language processor environment is not initialized for TSO/E READY mode. A language processor environment is required for executing REXX execs. To allow users to execute REXX execs from TSO/E READY mode, your user-written TMP must invoke the initialization routine IRXINIT to initialize a language processor environment. To initialize the environment, the TMP must do the following:

- Invoke the initialization routine IRXINIT to initialize a language processor environment. The environment must be integrated into TSO/E, that is, the TSOFL flag must be on. On the call to IRXINIT, you can provide parameters that are equivalent to the default values that TSO/E provides in the IRXTSPRM default parameters module.
- The initialization routine IRXINIT returns the address of the environment block for the new language processor environment in register 0. You must store the address of the environment block in the ECTENVBK field of the environment control block (ECT).
- You must ensure that the ECTEXTPR field in the ECT is set to zeroes.
- When all user-written TMP processing is completed, you must invoke the termination routine IRXTERM to terminate the language processor environment that was initialized by IRXINIT. The system does not automatically terminate the environment.

The following topics in this chapter describe the characteristics of a language processor environment, the different types of environments, and the default parameters modules that TSO/E provides. Chapter 15, "Initialization and Termination Routines" describes the initialization and termination routines IRXINIT and IRXTERM.

Summary of Changes

Summary of Changes
for SC28-1883-0
as Updated February 10, 1989
by Technical Newsletter SN28-1293

This Technical Newsletter, which supports TSO Extensions (TSO/E) Version 2, contains the following changes for TSO/E support of the REXX programming language. The newsletter also contains minor technical changes.

- New information about how to initialize a language processor environment if you use a user-written terminal monitor program (TMP)
- New values returned by the PARSE VERSION instruction for the language level description (3.46) and the language processor release date (30 Jun 1988). The new values support APAR OY17590 and are returned if you install the PTF that supports the APAR. If the PTF is not installed, the values returned are "3.45" and "20 Oct 1987."

Summary of Changes
for SC28-1883-0
TSO Extensions Version 2

This book is a new book in the TSO/E Version 2 library. It contains reference information about TSO/E REXX.

APAR Information

The following APARs provide TSO/E REXX instructions, functions, and services that are described in this book. The instructions, functions, and services listed below can be used only if your installation installs the PTF that supports the particular APAR.

- APAR OY17498 provides the TSO/E function MSG, which is described on page 118.
- APAR OY17590 provides the:
  - Ability to enable and disable condition traps using the CALL instruction (CALL ON and CALL OFF). The CALL instruction is described on page 32. Chapter 7, "Conditions and Condition Traps" describes how to enable and disable condition traps.
  - Ability to specify NAME *trapname* using the SIGNAL ON instruction. The SIGNAL instruction is described on page 62. Chapter 7, "Conditions and Condition Traps" describes how to enable and disable condition traps.
  - CONDITION built-in function, which is described on page 82.
  - Ability to specify up to 20 expressions on the CALL instruction and on function calls, such as MAX and MIN. If the PTF for the APAR is not installed, the maximum number of expressions you can specify is 10.
  - Exit routines for exec initialization and exec termination. The exits are described in "REXX Exit Routines" on page 392.
- APAR OY17558 provides the SYS1.SAMPLIB members for coding the parameters modules IRXPARMS, IRXTSPRM, and IRXISPRM. The SAMPLIB members are:
  - TSOREXX1 (for IRXPARMS)
  - TSOREXX2 (for IRXTSPRM)
  - TSOREXX3 (for IRXISPRM)
- APAR OY17979 provides alternate entry point names for the TSO/E REXX external entry points. The alternate entry point names are less than six characters and allow FORTRAN programs to call the TSO/E REXX external entry points.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Special Characters

SN28-1293-00