z/OS

# Stream I/O for TSO/E REXX

*Version 1*

z/OS

# Stream I/O for TSO/E REXX

*Version 1*

**Note**

Before using this information and the product it supports, be sure to read the general information under "Appendix B. Notices" on page 53.

**First Edition, February 2002**

This edition applies to Version 1 Release 3 of the z/OS TSO/E REXX Stream I/O function package (applicable for z/OS (5694-A01)), and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

# Contents

# About this book

This book describes the z/OS TSO/E REXX Stream I/O function package and its usage. This function package is a collection of I/O functions that follow the stream I/O concept. It extends and enhances the I/O capabilities of TSO/E REXX and shields the complexity of z/OS data set I/O to some degree. Further, the use of stream I/O functions provides for easier coding syntax and leads to better portability of REXX programs among different operating system platforms.

The function package can be used with TSO/E REXX on z/OS™, OS/390®, and MVS™ systems that provide the MVS Name/Token Services, which are required to hook the function package into an existing TSO/E REXX installation.

This book is intended for application programmers who want to apply the described functions to new or modified REXX programs. It is assumed that the reader is familiar with the REXX language, the TSO/E environment, and the logical organization of data sets in the z/OS environment. The stream I/O concept is introduced in "Chapter 2. Understanding the stream I/O concept" on page 5. If required, see also "Glossary of z/OS terms" on page 55.

## Related information

The reader should be familiar with the following books, or the equivalent books for the predecessors of z/OS:

- *z/OS V1R1.0 TSO/E REXX User's Guide*
- *z/OS V1R1.0 TSO/E REXX Reference*

## How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other REXX documentation:

- Visit our home page at:
  `http://www.ibm.com/software/ad/obj-rexx/support.html#Buy or get support`
  There you can access the Internet Online Form where you can enter comments and send them.
- Send your comments by e-mail to `swsdid@de.ibm.com`. Be sure to include the name of the book, the part number of the book, the version of REXX,

and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative. The mailing address is on the back of the Readers' Comments form. The fax number is +49-(0)7031-16-4892.

## How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ▸▸— symbol indicates the beginning of a statement.

  The —▸ symbol indicates that the statement syntax is continued on the next line.

  The ▸— symbol indicates that a statement is continued from the previous line.

  The —▸◂ symbol indicates the end of a statement.

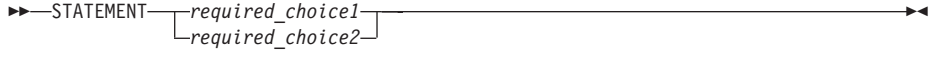
  Diagrams of syntactical units other than complete statements start with the ▸— symbol and end with the —▸ symbol.

- Required items appear on the horizontal line (the main path).

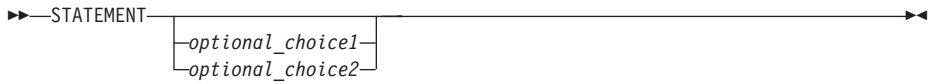

- Optional items appear below the main path.


- If you can choose from two or more items, they appear vertically, in a stack.

  If you *must* choose one of the items, one item of the stack appears on the main path.



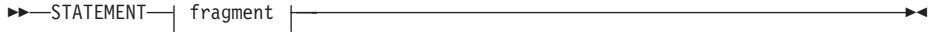

- If choosing one of the items is optional, the entire stack appears below the main path.




- If one of the items is the default, it appears above the main path and the remaining choices are shown below.


- An arrow returning to the left above the main line indicates an item that can be repeated.


  A repeat arrow above a stack indicates that you can repeat the items in the stack.
- A set of vertical bars around an item indicates that the item is a *fragment*, a part of the syntax diagram that appears in greater detail below the main diagram.




**fragment:**


- Keywords appear in uppercase (for example, PARM1). They must be spelled exactly as shown, but you can type them in uppercase, lowercase, or mixed case. Variables appear in all lowercase letters (for example, *parmx*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, you must enter them as part of the syntax.

The following example shows how the syntax is described.


,
MAX(
number
)

# Chapter 1. Installing the function package

The z/OS TSO/E REXX Stream I/O function package is a loadable file that contains multiple object files bound together. Before its functions can be accessed and executed, the function package must be properly integrated into TSO/E REXX. Perform the following steps to install the package.

## Preparation

1. Unpack the zipped file that contains the function package on a workstation. It contains the following files:
   - Object modules, as listed in step 3
   - JCL job `MAKESIO` to build the load modules
   - Parameter modules `IRXPARMS` and `IRXTSPRM`, modified for the needs of the REXX Stream I/O function package
   - Documentation, as file `STREAMIO.PDF`
   - Probably a `README` file with latest information
2. Allocate the following data sets:
   - For the object library: *<uid>*`.REXX.SIO130.OBJECT`, record format FB, record length 80
   - For the load library: *<uid>*`.REXX.SIO130.LOAD`, record format U, record length 0, block size 23200
3. Upload the following object modules to the object library *<uid>*`.REXX.SIO130.OBJECT`:

```
EAGIOCHI
EAGIOCHO
EAGIOCLS
EAGIODYN
EAGIOGET
EAGIOGFE
EAGIOGNM
EAGIOHKP  (the function package termination)
EAGIOLNI
EAGIOLNO
EAGIOLNS
EAGIOMSG
EAGIOOPE
EAGIOPUT
EAGIORET
EAGIOSTR
EAGEFSIO  (the function package directory)
```

4. Upload the parameter modules `IRXPARMS` and `IRXTSPRM`.
5. Upload the JCL job `MAKESIO`.

6. Customize the JCL job MAKESIO (listed also in "Appendix A. JCL job MAKESIO" on page 37). It contains predefined steps to automate the installation and verification. It does the necessary assemble and link-edit steps to create the load library *<uid>*.REXX.SIO130.LOAD. You need to customize the PROC section as follows:
   - Specify the data sets you have already allocated. Replace the first qualifier *<uid>* by the appropriate user ID.
   - Ensure that SYS1.MACLIB and SYS1.CSSLIB is referenced in your SYSLIB concatenation. SYS1.CSSLIB must contain the modules IEANTRT, IEANTCR, and IEANTDL.

     SYS1.CSSLIB contains the stubs for MVS Name/Token Services that the stream I/O functions require to share data with TSO/E REXX.
7. The parameter modules IRXPARMS and IRXTSPRM provided with this function package are modified exclusively for the needs of the REXX Stream I/O function package. Do not modify them. They are used by the MAKESIO job.

## Assembly, link-edit, and verification

1. Submit the MAKESIO job to assemble and link-edit the modules, and place the load module into a load library that is accessible by your system.

   Upon completion the load library *<uid>*.REXX.SIO130.LOAD should contain these load modules:

   ```
   EAGEFSIO EAGIOHKP IRXPARMS IRXTSPRM
   ```
2. At the end of the installation job MAKESIO runs a small REXX exec that issues several stream I/O function calls. You should see the appropriate output in the job output.

Note that load modules IRXPARMS and IRXTSPRM provided with this function package can only be used if the REXX Stream I/O function package is the only function package to be used on your system.

## Installations with multiple function packages

Your installation might already use other function packages. These are defined in the parameter modules IRXPARMS and IRXTSPRM installed on your system. You need to add the definitions for the REXX Stream I/O function package to these modules to make all function packages work.

1. Inspect the parameter modules IRXPARMS and IRXTSPRM provided with this function package. They contain the TSO/E default definitions and the definitions for the REXX Stream I/O function package.
2. Incorporate the modifications for the REXX Stream I/O function package into the modules IRXPARMS and IRXTSPRM that are installed on your system.

3. Assemble and link-edit the updated parameter modules `IRXPARMS` and `IRXTSPRM`.
4. Copy the resulting load modules `IRXPARMS` and `IRXTSPRM` to the load library. (The load modules now contain the definitions for all function packages.)
5. Copy the load modules `EAGEFSIO` and `EAGIOHKP` (created with the `MAKESIO` job) to the load library.

The new load library should now contain the load modules for all function packages.

More detailed information about function packages is described in *z/OS V1R1.0 TSO/E REXX Reference*.

## Usage considerations

Your TSO/E REXX installation might use the EXECTERM exec termination exit to customize the processing after REXX execs complete their processing. This customized processing can include closing of data sets, and freeing of resources that were allocated during the exec initialization step. If exec termination is used, a REXX exec does not necessarily need to close data sets it has opened.

On the other hand, the stream I/O function package provides the STREAM function, which can issue a CLOSE ALL stream command. If CLOSE ALL is used in a REXX exec, it also closes the data sets and frees the resources that were allocated with the first use of stream functions.

If you prefer relying on exec termination functionality (without using CLOSE ALL in your REXX exec), ensure that exec termination is active and APF-authorized, otherwise you might receive abend 066D.

To avoid these dependencies, use CLOSE ALL in your REXX execs regardless of the use of exec termination.

# Chapter 2. Understanding the stream I/O concept

This chapter introduces the stream I/O concept and the implementation for TSO/E REXX. The terminology, the functions, and the common elements are described. Further, attention is given to the aspects of TSO/E and z/OS data set handling from the view of the stream I/O functions.

This knowledge lets you effectively use the information in "Chapter 3. Stream I/O functions" on page 17.

## The basic elements of stream I/O

A stream is a popular concept for how to perform input/output to and from a program. Basically, a stream is a *sequence of characters* with functions to take characters out of one end, and put characters into the other end. In the case of input/output streams, one end of the stream is connected to a physical or logical I/O device, such as a keyboard, display, file, or queue. If it is an *output stream*, your program puts characters into one end of the stream, and an output device takes characters out of the other end. If it is an *input stream*, an input device puts characters into one end of the stream, and your program takes characters out of the other end.

The purpose of stream I/O is to simplify a programmer's view of input and output devices. The physical characteristics of I/O devices and the organization of data remain hidden. The data organization of devices is reduced to two simple forms:

- A sequence of characters that can be read or written character by character
- A sequence of lines that can be read or written line by line. A line in this context is defined as a sequence of characters that are terminated by means of any special character, or by means of the organizational form of the storage media.

A simple set of functions performs stream I/O operations from within a program.

- Housekeeping functions *declare streams* as input or output streams, *open* and *close* streams before and after using them, and allow to query their existence and characteristics.
- *Character input* and *character output* functions let the program read and write data character by character from input streams or to output streams.
- *Line input* and *line output* functions let the program read and write data line by line from input streams or to output streams.

- Further functions let the program check for the availability of input data from input streams.

During stream I/O operations a pointer is maintained for each stream. The pointer references the *current position* in a stream where a read operation or a write operation takes place. The position in a stream is relative to its beginning, counted as number of characters. Position 1 is always the first character. Pointers increase automatically during read operations and write operations. This eases the sequential reading from or writing to streams. The stream I/O functions can modify the position pointers by specifying explicit character or line positions to be read or written.

When streams are declared, they are given *names*. The input and output functions refer to these names to distinguish among multiple streams in a REXX program.

Generally, a stream can be any source or destination of external data that a program uses. Typical streams are files and data sets, and consoles for interactive input and output. The stream I/O concept also allows to view other sources and destinations as streams, for example, a reader, puncher, printer, program stack, queue, or a communication path. Programming environments that support stream I/O usually provide a *default input stream*, which is often the terminal input buffer, and a *default output stream*, often the display.

Data streams have two distinctive traits; they are either finite or conceptually unbound. An input stream from a file is finite because of the known quantity of characters; an input stream from a keyboard or communication path is unbound because of the unknown quantity. Stream I/O functions generally provide a mechanism of determining that an input stream is exhausted – that all data was read, and no more data is available. For finite streams they can detect the end of a file, for example. For unbound streams they might interpret special characters of the stream as delimiters.

## The TSO/E REXX Stream I/O implementation

Stream I/O is a concept already implemented in REXX for various operating system platforms. However, on z/OS and its predecessors, programmers needed to use the EXECIO command to access z/OS data sets. EXECIO requires programmers to consider many parameters, and to care about the allocation and deallocation of data sets. The TSO/E REXX Stream I/O functions hide this complexity. Programmers can use these easy-to-use functions to access z/OS data. Further, the use of stream I/O functions makes REXX programs more portable among platforms that support stream I/O.

The following sections describe the implementation of stream I/O for TSO/E REXX. A good understanding of this information is necessary to effectively use the individual functions described in "Chapter 3. Stream I/O functions" on page 17.

## The stream I/O functions

The function package provides the following functions:

- The STREAM function controls streams and their status. It opens and closes streams, declares the type of operation (either read or write), and queries the existence and details of streams.
- The CHARIN and LINEIN functions are the stream input functions. They perform character input or line input.
- The CHAROUT and LINEOUT functions are the stream output functions. They perform character output and line output.
- The CHARS and LINES functions determine whether data exists in input streams for further read operations.

In this context, the following terms require definitions:

- The term "character" is any single byte in the range of X'00'...X'FF', respectively 0...255. So, a "character stream" is synonymous with a "binary stream" or a "byte stream".
- The term "line" is defined as a sequence of characters that makes up the smallest unit that can be processed by the LINEIN and LINEOUT functions. A line read by LINEIN or written by LINEOUT does not process any additional line-terminating characters (such as new-line character or carriage return character) if they are not part of the string to be read or written. You might think of a line as a record of an MVS data set.

The function calling mechanism for the stream I/O functions is identical to the REXX built-in functions. Thus, they are called in REXX programs as functions, with the result being assigned to a variable, like in `rexx_variable = CHARS()`.

## Naming streams

TSO/E REXX provides a *default input stream* and a *default output stream*, which are used implicitly whenever a stream I/O function does not name a stream. In z/OS these default streams are associated with the console:

- For TSO/E background and MVS:
  - ddname SYSTSIN represents the default input stream.
  - ddname SYSTSPRT represents the default output stream.
- For TSO/E foreground:
  - ddname SYSIN represents the default input stream.
  - ddname SYSIN represents the default output stream.

If functions are to be performed on other streams, the streams must be named explicitly. The naming follows the rules and conventions for z/OS data sets as follows:

- A stream name can be the name of a data set, or the name of a data set member, for example:
  - A fully qualified data set name, for example `bill.january.data`.
  - A partially qualified data set name, for example `january.data`, where TSO/E prepends a system-defined prefix to the data set name, as in `<user_id>.january.data`.
  - A fully qualified or partially qualified name of a data set member, for example `bill.year2001.data(january)`.

  Data set names should be enclosed in single quotation marks to avoid a modification by TSO/E, such as `'year2001.data(january)'`.
- A stream name can also be a ddname that is known to TSO/E and has the required data sets or resources allocated to it, for example, SYSPRINT or SYSOUT.
- A stream name can be a ddname that is generated from a data set name through the STREAM function. Each time the STREAM function opens a stream that is specified as a data set name, it automatically generates enumerated ddnames of the form `&SYS`*xxxxx*. The leading ampersand distinguishes them from data set names, and *xxxxx* is an enumeration. These unique ddnames can be used in a REXX program to explicitly name a stream with the stream I/O functions.

  The following example shows how the STREAM function opens the data set member `SYS1.MACLIB(PARM)`, generates a ddname, assigns this ddname to the variable *infile*, and uses this variable in the following CHARIN function call to name the stream. To recognize the generated ddname you could add `SAY infile`, which displays something similar to `&SYS00004`.

  ```
  /* Open the file. */
  infile = STREAM("'SYS1.MACLIB(PARM)'",'C','OPEN')
  if infile ¬= "ERROR" then
     parm = CHARIN(infile,,20)
  ```

  Note the specification of the data set member name; the inner single quotation marks avoid a modification by TSO/E, the outer double quotation marks are the REXX convention for literal strings that include single quotation marks.

  A second use of this side effect is more sophisticated. You can open the same data set multiple times with this method, and the STREAM function will provide a respective number of unique ddnames. Using these ddnames with the stream I/O functions lets you maintain multiple position pointers in the same data set. See “Multiple read operations” on page 14 for a detailed description.

After streams are given names, the stream I/O functions use these names to specify on which stream an operation is to be performed.

## Transient and persistent streams

Streams might have a variety of sources and destinations, but they are either transient or persistent. Both types have certain characteristics that should be known when using the stream I/O functions.

- Transient streams usually communicate with the human user. The default input stream and the default output stream, if they represent the keyboard and the display, are typical examples. A communication path in a network is another example of a transient stream because of its similar behavior.

  The distinctive feature of a transient stream is that after a specific character or line was read from or written to a stream this process cannot be repeated. For example, if your REXX program reads user input from the default input stream, the characters are read as they are typed. You cannot change the position in a stream and read again the same character or line without the character being typed again.
- Persistent streams are usually files or data sets or equivalent media.

  The distinctive feature of a persistent stream is that you can repeatedly change the position in a persistent stream and read or write from and to different positions, within the boundaries of the stream.

When you use the stream I/O functions you will find that several parameters, such as the *start* position for the CHARIN function, are applicable only for persistent streams. In transient streams, read positions and write positions always default to the *next* character or line in a stream. In persistent streams, read positions and write positions can generally be changed within the boundaries of a stream.

**Note:** The current implementation of the TSO/E REXX stream I/O functions is limited with respect to randomly changing the positions in persistent streams. See the description of the individual functions for these capabilities. The LINEIN function might provide the most flexibility.

## Opening and closing streams

A stream needs to be opened before it can be used, as a means to make the stream known to a REXX program, and to gain access to this stream for read and write operations.

The default input stream and the default output stream are opened when TSO/E REXX is started. Any stream I/O function that does not specify a stream by name performs its read operation or write operation on a default stream.

### Implicit versus explicit opening of streams

Streams are opened either implicitly or explicitly. All stream I/O functions open a named stream *implicitly* upon their first use within a REXX program. The named stream remains open for further function calls.

Streams can also be opened *explicitly* with the STREAM function. Explicit opening (as well as closing) of streams has some advantages. For the sake of a few lines, your program is more understandable, and you can easily recognize the type of operation (read or write) allowed on a stream.

You must explicitly open a stream to perform multiple read operations on the same data set. See also "Naming streams" on page 7 and "Multiple read operations" on page 14.

### Opening streams for read or write operations

A stream is opened for either read operations or write operations. It is not recommended to have a stream concurrently open for both types of operations.

If a stream is opened *implicitly*, the stream I/O function that is used at first decides the type of operation. A CHARIN, CHARS, LINEIN, or LINES function call opens a stream for read operations. A CHAROUT or LINEOUT function call opens a stream for write operations.

If a stream is opened *explicitly* through the STREAM function, the type of operation is specified as a parameter of the STREAM function.

After the type of operation is determined for a specific stream, you can use only the corresponding stream I/O functions, otherwise an error occurs.

To change the type of operation allowed for a stream, you first need to close the stream, then open it again for a different type of operation.

Note that opening a stream with the stream I/O functions in a TSO/E REXX program implies an allocation of the corresponding resource. You do not need to allocate a resource with the TSO/E ALLOCATE command, or by any other means.

### Opening nonexistent streams

Persistent streams like data sets or files might not exist at the time they are opened. An attempt to open such a stream for read operations, either implicitly or explicitly, will fail. An attempt to open such a stream for write operations, either implicitly or explicitly, allocates an empty data set with VB 255, or whatever the operating system has defined as default. If you require a different record format, allocate the data set through the TSO/E ALLOCATE command, ISPF option 3.2, or a DD statement in batch.

If you name a nonexisting member (directly as data set member, or indirectly through a ddname) with a stream output function, the member is created and receives all subsequent output data.

You can use the STREAM function to query the existence of a stream before it is opened for read operations or write operations.

### Closing streams

All opened streams are closed implicitly when the REXX program ends. You can also use the STREAM function to explicitly close all or specific streams. You might want to do so for clarity, to free dynamically allocated working storage, or to change the type of operations on a stream (from read to write, or vice versa).

Closing a stream causes all pending write operations on this stream to be executed first. Pending write operations can be, for example, partially written lines on fixed block data sets.

## Stream formats

The z/OS TSO/E REXX stream I/O functions can work with the following files and data sets:

- QSAM (queued sequential access method) files
- z/OS sequential data sets and single members of partitioned data sets with the following record formats:
  - Fixed block formats (FB), and fixed length with ASA control characters (FBA)
  - Variable length (VB), and variable length with ASA control characters (VBA)

As a rule, the stream I/O functions do not write any *additional* formatting or control characters (other than what is specified as *string* with the stream I/O function) to a data set. Vice versa, the stream I/O functions read whatever is considered data from a data set. The read functions do not hide or remove anything.

Note that the record formats of data sets influence how the output stream functions succeed:

- The LINEOUT function attempts to write a specified string to a data set as a single line.

  If the length of the string fits in to the LRECL of the data set, the line is written. For data sets with a fixed record length the line is padded with blanks up to the logical record length.

  If the length of the string exceeds the LRECL of the data set, the line is *truncated*. The function returns a 1 as an indication that data remains to be written to the stream.

Note that, if the LINEOUT function writes a null string, the stream is closed. Nothing is written to a data set.

- The CHAROUT function attempts to write a specified string to a data set character by character.

  No truncation takes place. Subsequent strings of characters are concatenated to previously written strings. If a fixed or maximum LRECL is exceeded, the characters wrap around to the next record. Thus, a large string can cause several records to be written. A partially filled record is retained internally until it is filled by subsequent CHAROUT (or LINEOUT) function calls, or until the output stream is closed. For data sets with fixed record length a partial record is padded with blanks.

Contrary, the LINEIN function and the CHARIN function attempt to read a line, respectively a number of characters, from a persistent stream. For data sets with a fixed record length the string returned includes the padded blanks.

You can combine the use of the CHARIN and LINEIN, respectively the CHAROUT and LINEOUT, functions for whatever purpose. For example, you can write a few characters with CHAROUT, followed by a line written with LINEOUT. The basic rule is that the line starts at the position where the character string ended. To use these combinations, understand how the position pointers in stream work, as described in "Position pointer details".

## Position pointer details

Each persistent stream maintains a position pointer to mark the position where a read operation or write operation takes place. By definition, position 1 marks the first character in a stream, and the positions are counted in number of characters relative to the beginning of a stream. When a stream is opened, the position pointer is set to position 1 of the stream.

The general use of position pointers is to ease the sequential reading and writing of streams. By default the first read operation starts at position 1, reads a number of characters or a line, and automatically increments the read position to the next unread character or line. A subsequent read operation starts at the incremented read position (the *current read position*). Similarly, a first write operation starts at position 1, writes a number of characters or a line, and automatically increments the write position behind the last character written. A subsequent write operation starts at the incremented write position (the *current write position*). The current position is maintained automatically. Thus, for sequential processing of a persistent stream, the stream I/O functions do not require the specification of a stream position.

The position pointer in a persistent stream can be manipulated to a certain degree to set it to a specific position where the next read operation or write operation should take place.

- A CHARIN or CHAROUT *start* value of 1 sets the current position to the beginning of a stream.
- A LINEIN *line* value can be set to any line number within a stream, which sets the current position to the beginning of this line.
- A LINEOUT *line* value of 1 sets the current position to the beginning of the stream (the beginning of the first line).

Note that the *line* parameter specifies a line, not the position of a character. Lines are counted from 1 to *n*, where line 1 is the first line in a stream.

Each open stream has its own position pointer. If a stream is opened for read operations, the pointer is either automatically set by any sequence of CHARIN and LINEIN function calls, or it is explicitly manipulated as described. Likewise, if a stream is opened for write operations, the pointer is either automatically set by any sequence of CHAROUT and LINEOUT function calls, or it is explicitly manipulated as described.

The CHARIN and LINEIN functions manipulate the same read position in a stream; respectively the CHAROUT and LINEOUT functions manipulate the same write position in a stream. For example, if two lines of 80 characters each where written to a fixed length data set by LINEOUT, followed by a CHAROUT of five characters, the current write position is 166 (the position where the next write operation would start). A subsequent LINEOUT with 80 characters would not succeed because only 75 characters would fit in the record. The line would be truncated. Conversely, if a line of 50 characters was written by LINEOUT to a fixed length (80) data set, the line is padded with blanks, and the current write position is 81 (the position were the next write operation would start). A subsequent CHAROUT or LINEOUT function starts at position 81.

## End-of-stream treatment

For transient and persistent input streams use the CHARS function or the LINES function to detect the end of an input stream. These functions return 0 if no more characters or lines are available for reading, or they return 1 if at least one character, respectively line, is available for reading.

For transient streams, 0 means that the user has terminated the input to the stream by means of the two-character sequence `/*`, followed by the Enter key.

For persistent streams, 0 means that the input stream is either empty, or a previous read operation has already read the last character or line, or repeated read operations have triggered an end-of-file condition.

An attempt to read beyond the end of a stream returns a null string and triggers an error message. If this happens, the stream should be closed and reopened. Do not try to manipulate the position pointer after the end-of-file condition was triggered.

## Error treatments

### Stream I/O processing errors

The current implementation of the z/OS TSO/E REXX Stream I/O function package supports only the `SIGNAL ON SYNTAX` condition trap. This means that a `SYNTAX` condition is raised if a language processing error, a syntax error, or a run-time error occurs during the execution of a stream I/O function call.

Note that it is not possible to trap `NOTREADY` conditions. Therefore, before using a stream, query its existence with the `STREAM ... QUERY EXISTS` function call.

If a syntax condition is raised because of a stream I/O function call, it is recommended to exit the REXX program. The recovering from such a syntax condition might cause unpredictable results.

### Messages

The z/OS TSO/E REXX Stream I/O function package adds its own set of messages to TSO/E REXX. Similar to TSO/E REXX messages, each message consists of a message identifier and a message text. The message identifier is EAGSIO.

See "Chapter 4. Stream I/O messages" on page 31, if required.

## Multiple read operations

As already described, each open stream maintains its own position pointer. This is sufficient for most sequential operations on a persistent stream. However, if you have a need to perform multiple read operations on the same stream, for example if you work with an indexed data set, you can use the following method. (Multiple write operations as well as concurrent read and write operations on the same stream are not supported.)

Use the STREAM function to open a stream explicitly for read operations. Name the data set to work with by its fully qualified or partially qualified data set name. The STREAM function returns a ddname, for example `&SYS00001`. You have now a stream open with its own position pointer.

Repeat this step with the *same* data set name. The next ddname might be `&SYS00002`. You have now a second stream open with its own position pointer.

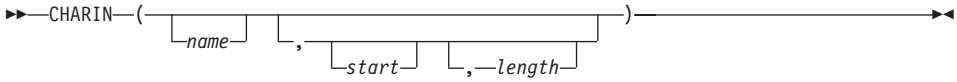
Both streams represent the same data set. Both streams have their position pointers, each set to position 1 at the beginning.

You can now perform various CHARIN and LINEIN function calls on the streams `&SYS00001` and `&SYS00002` in any combination, and each stream pointer is maintained independently.

# Chapter 3. Stream I/O functions

This chapter lists the stream I/O functions and shows their syntax elements. For each function, the basic function, the boundary conditions, the parameters, and the results are described. Examples show possible uses and return values.

## CHARIN (Character Input)




Returns a string of up to *length* characters read from the character input stream *name*.

For persistent streams, a read position is maintained for each stream. Any read operation from the stream will by default start at the current read position. When the read operation is completed, the read position is increased by the number of characters read.

A *start* value of 1 can be given, together with a *length* of 0, to refer to the first character in a persistent stream. The read position is set to the beginning of the stream, no characters are read, and the null string is returned.

For transient streams (SYSIN in TSO/E foreground) only: If there are fewer than *length* characters available, then the execution of the program will normally stop until sufficient characters become available.

**name**
Specifies the name of the character input stream. If it is not specified, the default input stream is assumed.

**start**
For a persistent stream, specify a value of 1 (and a *length* of 0) to set the read position to the first character in the stream. No other value is supported.

For a transient stream do not specify a read position.

**length**
Specifies the number of characters to be returned. The default is 1.

If *length* is 0, no characters are read, a null string is returned, and the read position is set to the value specified by *start*.

### Comments

If a *length* of 0 is given (to specify an explicit read position, without reading from the stream) you must also specify *start* or let *start* default to 1 (the first character in a stream). This combination is only applicable to persistent streams, because for transient streams you cannot specify an explicit read position.

### Results

A string of characters, or a null string.

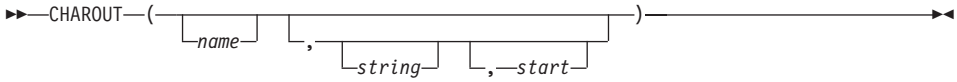
## Examples

```
CHARIN(myfile,1,3)  -> 'MFC'   /* First 3 characters are read.           */


CHARIN(myfile,1,0)  -> ''      /* Read position set to start position.   */
CHARIN(myfile)      -> 'M'     /* 1 character read from start position.  */
CHARIN(myfile,,2)   -> 'FC'    /* Next 2 characters read.                */


/* Reading from default input stream (here, the keyboard).               */
/* The user types 'abcd efg'.                                            */
CHARIN()            -> 'a'     /* Default is one character.              */
CHARIN(,,5)         -> 'bcd e' /* Next 5 characters.                     */
```

## CHAROUT (Character Output)




Returns the result (0 or 1) of the write operation after attempting to write *string* to the character output stream *name*. *string* can be the null string, then no characters are written to the stream and 0 is returned.

For persistent streams, a write position is maintained for each stream. Any write operation to the stream will by default start at the current write position. When the write operation is completed, the write position is increased by the number of characters that are written. The initial write position is the beginning of the stream, so that calls to CHAROUT will append characters to the beginning of the stream.

A *start* value of 1 can be given, together with *string* being omitted (or specified as a null string), to refer to the first character in a persistent stream. The write position is set to the beginning of the stream, no characters are written to the stream, and 0 is returned.

If neither *start* nor *string* is given, the output stream is closed, and 0 is returned.

The execution of the CHAROUT function will normally stop until the output operation is effectively complete. If it is impossible for a character to be written, CHAROUT returns with a result of 1, and a corresponding error message is shown.

**name**
Specifies the name of the output stream. If it is not specified, the default output stream is assumed.

**string**
specifies the string to write.

For transient streams, the length of the string is limited by the capabilities of your input device, usually 80 characters.

For persistent strings, the length is limited to a maximum of 32760 characters.

**start**
For a persistent stream, specify a value of 1 and omit *string* to set the write position to the first character in the stream. No other value is supported.

For a transient stream do not specify a write position. (If a value is specified, it is ignored.)

## Results

Returns 0 after the specified characters are successfully written, or 1 if the specified characters could not be written.

## Examples

```
CHAROUT(myfile,'Hi') -> 0      /* */
CHAROUT(myfile)      -> 0
CHAROUT(,'Hi')       -> 0
CHAROUT(V90,'29 BYTES FOR a V90 FILE LRECL')  -> 0    /* Variable format */
CHAROUT(V20,'29 BYTES FOR a V20 FILE LRECL')  -> 9    /* Variable format */
```

If a string of 29 characters is written to a data set with RECFM=F and LRECL=20, 20 bytes are written to record *n*, and nine bytes are written to record *n*+1.

If a string of 29 characters is written to a data set with RECFM=V or VB and LRECL=20, four bytes are reserved for the RDW, 16 bytes are written to record *n*, and 13 bytes are written to record *n*+1.

In both cases CHAROUT returns 0, as no truncation takes place.

## CHARS (Characters Remaining)


Returns 0, or 1 if characters are remaining in the character input stream *name*.

**name**
Specifies the name of the input stream. If it is not specified, the default input stream is assumed.

### Results

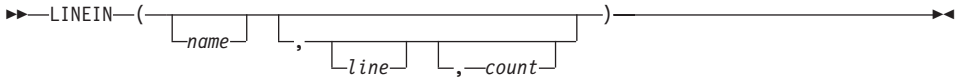
Returns 1, if one or more characters are available.

Returns 0, if no character is available. The data set or data set member is empty, or a previous read operation has already read the last character, or a previous read operation has triggered an EOF condition.

### Examples

```
CHARS(myfile)  -> 1      /* EOF not reached. */
CHARS(empty)   -> 0      /* Empty data set.  */
CHARS()        -> 1      /* TSO/E console.   */
```

## LINEIN (Line Input)




Returns *count* (0 or 1) lines read from the character input stream *name*.

For persistent streams, a read position is maintained for each stream. Any read operation from the stream will by default start at the current read position.[1] When the read operation is completed, the read position is increased by the number of characters read.

A *line* number can be given to set the read position to the start of a specified line. This line number must be positive and within the boundaries of the stream, and it must not be specified for a transient stream. A value of 1 for *line* refers to the first line in the stream.

If a *count* of 0 is given, then the read position is set to the start of the specified *line*, but no characters are read, and the null string is returned.

For transient streams (SYSIN in TSO/E foreground) only: If a complete line is not available in the stream, then the execution of the program will normally stop until the line becomes available.

**name**
Specifies the name of the input stream. If it is not specified, the default input stream is assumed.

**line**
For a persistent stream, it specifies an explicit read position. The default is 1, or the position set by a previous read operation.

For a transient stream do not specify a read position.

**count**
Specifies the number of lines to be returned. Only 0 or 1 is allowed. The default is 1.

If *count* is 0, no lines are read, a null string is returned, and the read position is set to the value specified by *line*.

1. Under certain circumstances, therefore, a call to LINEIN will return a partial line if the stream has already been read with the CHARIN function, and part but not all of the line has been read.

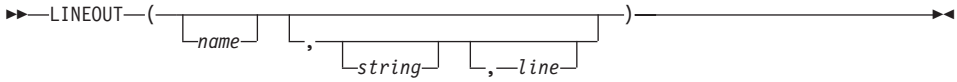
## Comments

If a *count* of 0 is given (to specify an explicit read position, without reading from the stream), you must also specify *line*. This combination is only applicable to persistent streams, because for transient streams you cannot specify an explicit read position.

## Results

A line, or a null string.

## LINEOUT (Line Output)




Returns the result (0 or 1) of the write operation after attempting to write *string* as a line to the character output stream *name*. The result is either 0 (the line was successfully written) or 1 (an error occurred while writing the line). *string* can be the null string, then no characters are written to the stream and 0 is returned.

For persistent streams, a write position is maintained for each stream. Any write operation will by default start at the current write position.[2] When the write operation is completed, the write position is increased by the length of the line written. The initial write position is the beginning of the stream, so that calls to LINEOUT will append lines to the beginning of the stream.

**Note:** The *line* parameter is provided for compatibility reasons, but does not allow to set the write position in this implementation.

If neither *line* nor *string* is given, the output stream is closed, and 0 is returned.

The execution of the LINEOUT function will normally stop until the output operation is effectively complete. If it is impossible for a line to be written, LINEOUT returns with a result of 1, and a corresponding error message is shown.

**name**
Specifies the name of the output stream. If it is not specified, the default output stream is assumed.

**string**
Specifies the string to write as a line.

**line**
Specify value of 1, or specify no value. See the previous note.

### Results

Returns 0 after the specified line is successfully written, or 1 if the line could not be written or is only partially written.

2. Under certain circumstances, therefore, the characters written by a call to LINEOUT might be added to a partial line previously written to the stream with the CHAROUT routine. LINEOUT conceptually terminates a line at the end of each call.

## Examples

```
LINEOUT(myfile,'Hi')                 -> 0    /* Writes the string.          */
LINEOUT(myfile,,                     -> 0    /* No action.                  */
LINEOUT(myfile)                      -> 0    /* Output stream is closed.    */
LINEOUT(myfile,'String longer than lrecl')   -> 1   /* Truncated.          */
```

## LINES (Lines Remaining)


Returns 0, or 1 if lines are remaining in the character input stream *name*. If the stream has already been read with the CHARIN function, this might include an initial partial line.

**name**
Specifies the name of the input stream. If it is not specified, the default input stream is assumed.

### Results

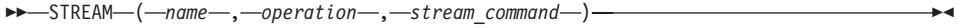
Returns 1, if one or more lines are available.

Returns 0, if no line is available. The data set or data set member is empty, or a previous read operation has already read the last line, or a previous read operation has triggered an EOF condition.

### Examples

```
LINES(myfile)  -> 0      /* EOF encountered. */
LINES(empty)   -> 0      /* Empty data set.  */
LINES()        -> 1      /* TSO/E console.   */
```

## STREAM (Operations)




Returns a string describing the state of the character stream *name*, or the result of an operation upon the character stream *name*.

This function is used to request information on the state of an input or output stream, or to carry out some particular operation on the stream.

**name**
Specifies the name of the stream. Use a fully or partially qualified data set name (with or without a member specification), or a ddname known to TSO/E.

Note that the STREAM function returns enumerated ddnames of type `&SYS`*xxxxx* when it performs an OPEN, OPEN READ, or OPEN WRITE command. You can use these ddnames to name a stream in the stream I/O function calls. For more information see "Naming streams" on page 7.

**operation**
Specifies the type of operation. This parameter must be the string **Command**, or its leading character **C**. The first character must be uppercase, and subsequent characters are ignored.

**stream_command**
Specifies one of the following commands (in capital letters) to be performed on the named stream:

**CLOSE**
Closes the named stream.

**CLOSE ALL**
Closes all streams that have been opened so far in this REXX exec, and frees resources that are bound to opened streams. For this stream command the first parameter *name* is ignored and can be omitted, such as in `STREAM(,'Command','CLOSE ALL')`.

It is recommended that you use this stream command in your REXX execs, regardless of external exec termination exits providing similar functions. See "Usage considerations" on page 3 for a detailed description.

**OPEN** Is identical with **OPEN READ**.

**OPEN READ**
Opens the named stream for input, respectively read operations. The input stream must already exist.

Note that input functions (CHARIN, CHARS, LINEIN, LINES) implicitly open streams for input at their first usage. You can explicitly open a stream for clarity reasons. You need to explicitly open a stream if you want to maintain multiple position pointers. See "Multiple read operations" on page 14, if required.

**OPEN WRITE**
Opens a named stream for output, respectively write operations. If the output stream does not exist, a data set is allocated with the system defaults. See "Opening nonexistent streams" on page 10, if required.

Note that output functions (CHAROUT, LINEOUT) implicitly open streams for output at their first usage. You can explicitly open a stream for clarity reasons.

When the output stream is opened, the position pointer is initially set to position 1. Thus, the contents are overwritten. See "Position pointer details" on page 12, if required).

**QUERY EXISTS**
Queries the existence of a named stream and returns the fully qualified data set name that is allocated to this stream.

**QUERY REFDATE**
Queries the date when the named stream was last referenced. The date is returned in julian form.

**Note:** You can also use the STREAM function to query the level of the installed function package. This might be required if you need to report problems. If required, type `STREAM(,'COMMAND','QUERY SERVICELEVEL')`. The function returns the service level of the installed function package in the form `REXXSIO <v><r><m> FIX<nnnn> <yyyy><mm><dd>`, for example `REXXSIO 130 FIX0000 20011207`.

## Results

- **CLOSE** returns 0. If unsuccessful, RC = 4 is returned, and a message is issued. The named stream might not exist, or it has already been closed.
- **OPEN** returns a ddname as a string `&SYS`*xxxxx*, with *xxxxx* being an enumeration. If unsuccessful, the string `ERROR` is returned.
- **QUERY REFDATE** returns the date in julian form (like `1999/360`), or a null string if the stream does not exist or the date cannot be determined.

## Examples

```
STREAM('MYDATA FILE','C','CLOSE') /* Closes the named data set. */

STREAM(strinp,'C','OPEN')         /* Opens an input stream.     */

STREAM(strout,'C','OPEN WRITE')   /* Opens an output stream.    */
```

```
STREAM('YOURDATA.FILE','C','QUERY EXISTS')
                                  /* Request the fully qualified */
                                  /* data set name.              */

STREAM('MY.DATA.FILE','C','QUERY REFDATE')
                                  /* Requests the date when last */
                                  /* referenced.                 */
```

# Chapter 4. Stream I/O messages

One or more of the following messages might occur in response to a problem during a stream I/O function call. If the problem cause is likely to be with your REXX program, detailed information is given with each message. If the problem cause is outside your REXX program, you need to see the appropriate TSO/E or z/OS documentation or to contact the system administrator for further help.

**EAGSIO0001 Invalid numeric parameter for REXXIO.**

**Explanation:** A function call contains a numeric parameter that is not allowed. A *line*, *count*, or *start* parameter value might be negative or outside the boundaries of a stream.

**EAGSIO0002 Invalid 'count' value for LINEIN.**

**Explanation:** The *count* parameter contains a value other than 0 or 1.

**EAGSIO0003 Invalid number of parameters for LINEIN.**

**Explanation:** Possible syntax problem with the LINEIN function call, or more than three parameter values specified. Check also the use of commas in the function call. See "LINEIN (Line Input)" on page 23.

**EAGSIO0004 Invalid file specification for REXXIO.**

**Explanation:** A stream name is not properly specified. The name might contain invalid characters, or a qualifier might be more than eight characters long. See also "Naming streams" on page 7.

**EAGSIO0005 Invalid number of parameters for LINES.**

**Explanation:** Possible syntax problem with the LINES function call, or more than one parameter value specified (no commas in the function call). See "LINES (Lines Remaining)" on page 27.

**EAGSIO0007 Invalid number of parameters for LINEOUT.**

**Explanation:** Possible syntax problem with the LINEOUT function call, or more than three parameter values specified. Check also the use of commas in the function call. See "LINEOUT (Line Output)" on page 25.

**EAGSIO0008 Invalid line number for LINEOUT.**

**Explanation:** The *line* parameter contains a value for a transient stream, or a value other than 1 for a persistent stream.

**EAGSIO0009 The file failed to open for REXXIO.**

**Explanation:** The specified stream failed to open. The ddname might not be allocated.

**EAGSIO0010 The file is a partitioned data set, but no member name was specified.**

**Explanation:** You need to specify the stream name with an explicit member name because the data set is partitioned. See "Naming streams" on page 7.

**EAGSIO0011 The file is a sequential data set, but a member name was specified.**

**Explanation:** Do not specify the stream name with an explicit member name; the data set is not partitioned. See "Naming streams" on page 7.

**EAGSIO0012 Record format of file is not supported.**

**Explanation:** An attempt was made to open a persistent stream with an unsupported record format. See "Stream formats" on page 11 for supported formats.

**EAGSIO0013 Data set organization of file is not supported.**

**Explanation:** An attempt was made to open a persistent stream with an unsupported data set organization. See "Stream formats" on page 11 for supported formats.

**EAGSIO0014 Warning: Output record truncated.**

**Explanation:** A LINEOUT function call attempted to write a string that exceeds the LRECL of the data set. A preceding CHAROUT function call might have already written several characters, or the string length exceeds the LRECL of the data set. See "Stream formats" on page 11 for details.

**EAGSIO0015 Cannot allocate data set. Insufficient storage.**

**Explanation:** Increase the Region size (MVS).

**EAGSIO0016 Cannot allocate data set. Data set cannot be accessed exclusively.**

**Explanation:** Opening a stream for write operations requires exclusive allocation. Someone else has already allocated the data set.

**EAGSIO0017 Cannot allocate data set. Data set in use by another user or job.**

**Explanation:** Someone else has already allocated the data set exclusively. You cannot open the stream for read operations or write operations yet.

**EAGSIO0018 Cannot allocate data set. No unit available.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0019 Cannot allocate data set. Volume cannot be mounted.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0020 Volume allocated to another user or job.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0021 Cannot allocate data set. Number of required devices unavailable.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0022 Cannot allocate data set. Volume or unit in use by system.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (exclusively used by someone else). Ask the system administrator for help.

**EAGSIO0023 Cannot allocate data set. Volume mounted on an ineligible device.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0024 Cannot allocate data set. Specified device in use.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0025 Cannot allocate data set. Specified Volume is on another device.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0026 Cannot allocate data set. Maximum number of allocations reached.**

**Explanation:** You have already allocated too many data sets. The maximum number of allocations is specified in TSO/E and MVS. Either try to free some other allocations, or ask the system administrator for help.

**EAGSIO0027 Cannot allocate data set. Maximum number of allocations exceeded.**

**Explanation:** The maximum number of concurrent allocations is reached. Either try to free some ddnames, or ask the system administrator for help.

**EAGSIO0028 Cannot allocate data set. Job Entry Subsystem unavailable.**

**Explanation:** The JES is not available to verify an allocation request. Ask the system administrator for help.

**EAGSIO0029 Cannot allocate data set. Number of volumes exceeds limit.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0030 Cannot allocate data set. Request cancelled by the operator.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system operator for help.

**EAGSIO0031 Cannot allocate data set. MSS volume not accessible from unit.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0032 Cannot allocate data set. MSS volume does not exist.**

**Explanation:** You specified a stream name as a data set that cannot be allocated. Ask the system administrator for help.

**EAGSIO0033 Cannot allocate data set. Data set name not found.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (the data set name is not cataloged). Ask the system administrator for help.

**EAGSIO0034 Cannot allocate data set. Locate I/O error.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (probably a system problem). Ask the system administrator for help.

**EAGSIO0035 Cannot allocate data set. DADSM I/O error.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (probably a system problem). Ask the system administrator for help.

**EAGSIO0036 Cannot allocate data set. Data set not on volume as denoted by catalog.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (probably a system problem). Ask the system administrator for help.

**EAGSIO0037 Cannot allocate data set. OBTAIN I/O error.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (probably a system problem). Ask the system administrator for help.

**EAGSIO0038 Cannot allocate data set. Required catalog not mounted.**

**Explanation:** You specified a stream name as a data set that cannot be allocated (probably a system problem). Ask the system administrator for help.

**EAGSIO0039 The requested member is not in the specified data set.**

**Explanation:** An attempt was made to open a partitioned data set for a read operation, but the data set member does not exist.

**EAGSIO0040 The STOW failed during the close of a data set.**

**Explanation:** Probably a system problem. Ask the system administrator for help.

**EAGSIO0041 Invalid number of parameters for CHARIN.**

**Explanation:** Possible syntax problem with the CHARIN function call, or more than three parameter values specified. Check also the use of commas in the function call. See "LINEIN (Line Input)" on page 23.

**EAGSIO0042 Invalid number of parameters for CHAROUT.**

**Explanation:** Possible syntax problem with the CHAROUT function call, or more than three parameter values specified. Check also the use of commas in the function call. See "LINEIN (Line Input)" on page 23.

**EAGSIO0043 Invalid number of parameters for CHARS.**

**Explanation:** Possible syntax problem with the CHARS function call, or more than one parameter value specified (no commas in the function call). See "LINEIN (Line Input)" on page 23.

**EAGSIO0044 Invalid 'start' value for CHAROUT.**

**Explanation:** The *start* parameter contains a value other than 1.

**EAGSIO0045 Invalid 'start' value for CHARIN.**

**Explanation:** The *start* parameter contains a value other than 1.

**EAGSIO0046 Invalid 'line' value for LINEIN.**

**Explanation:** The *line* parameter contains a value that is negative or not within the boundaries of the stream (the specified line might not exist).

**EAGSIO0047 Read error in REXXIO.**

**Explanation:** TSO/E returned with a Read error. See the subsequent messages for further information. If required, ask the TSO/E support for help.

**EAGSIO0048 CLOSE ignored. File already closed, or not found.**

**Explanation:** The stream was already closed by a preceding STREAM CLOSE command or by an

implicit close, or the named stream does not exist.

**EAGSIO0049 Logic error IEANTRT token retrieval.**

**Explanation:** Should never occur. Contact IBM service.

**EAGSIO0050 Invalid command for STREAM specified.**

**Explanation:** The *stream_command* parameter of the STREAM function call contains an invalid value. See "STREAM (Operations)" on page 28.

**EAGSIO0051 Invalid parameter for STREAM QUERY specified.**

**Explanation:** The *stream_command* parameter of the STREAM function call contains an invalid value. Only `QUERY EXISTS`, `QUERY REFDATE`, and `QUERY SERVICELEVEL` are supported. See "STREAM (Operations)" on page 28.

**EAGSIO0052 I/O RC=12, DCB already opened for a different type of I/O operation.**

**Explanation:** Error returned by TSO/E. A data set operation was tried in a mode different from the initial mode.

**EAGSIO0053 I/O RC=16, Output data was truncated for WRITE option.**

**Explanation:** A LINEOUT function call failed to write a string. The LRECL value of the data set in question might be too small.

**EAGSIO0054 I/O RC=20, unsuccessful processing, function not performed.**

**Explanation:** Error returned by TSO/E. Subsequent messages might provide further information. If required, ask the TSO/E support for help.

**EAGSIO0055 I/O RC=24, unsuccessful processing, file cannot be opened.**

**Explanation:** Error returned by TSO/E. Subsequent messages might provide further information. If required, ask the TSO/E support for help.

**EAGSIO0056 I/O RC=28, unsuccessful processing. Language processor cannot be located.**

**Explanation:** Error returned by TSO/E. Subsequent messages might provide further information. If required, ask the TSO/E support for help.

**EAGSIO0057 I/O RC=32, unsuccessful processing. Internal error in REXXIO.**

**Explanation:** Error returned by TSO/E. Subsequent messages might provide further information. If required, ask the TSO/E support for help.

**EAGSIO0058 Record for console must be Fixed 80.**

**Explanation:** An attempt to read from SYSTSIN with IRXJCL failed. The DSN must be F 80.

**EAGSIO0059 SVC99 allocation error, errorcode not known in EAGIODYN.**

**Explanation:** Should never occur. Contact IBM service.

**EAGSIO0060 Allocation failed. Too many attempts.**

**Explanation:** Too many unsuccessful allocation attempts from within the REXX program have used up the available storage.

# Appendix A. JCL job MAKESIO

**Note:** For pre-reading and walkthrough purposes only. The actual job might be slightly different.

```
//* -->uidCSIO  JOB - Specify your Job card here
//*--------------------------------------------------------------------*
//*   IBM z/OS TSO/E REXX Stream I/O function package. VRM 130         *
//*                                                                    *
//*   OCO Source Materials Program Property of IBM                     *
//*   5695-014 IBM Library for SAA REXX/370                            *
//*   (C) Copyright IBM Corp. 1989, 1994.                              *
//*                                                                    *
//*   Purpose:                                                         *
//*    JCL to assemble and linkedit the REXX Stream I/O function       *
//*    package. After successful creation, a job step to verify the    *
//*    functionality is executed.                                      *
//*                                                                    *
//*   Setup:                                                           *
//*    If you already have function packages and if you want to        *
//*    add this one, you must modify the IRXPARMS, IRXTSPRM            *
//*    according to your needs. The setup here is exclusively for      *
//*    REXX Stream I/O. EXECTERM is not used to avoid APF need.        *
//*                                                                    *
//*   Necessary Modifications:                                         *
//*    The object and load libraries, defined in the PROC section,     *
//*    have to be allocated.                                           *
//*    For the object library it is recommended: PDS FB 80,            *
//*    for the load library: PDS U 0 23200 etc.                        *
//*    Verify the DS Names listed in the PROC section, especially      *
//*    the first DSN qualifier which defines the user ID.              *
//*                                                                    *
//*   Note:                                                            *
//*    The Job inputs are defined in section:                          *
//*      Source inputs for all Job steps                               *
//*                                                                    *
//*                                                                    *
//*   Job Steps:                                                       *
//*     ALLOLOAD    Allocate load library - job step is commented,     *
//*                 activate if desired, UNIT=SYSDA used.              *
//*     COMPRESS    Compress load library                              *
//*                 Discard COMPRESS and COMPRESS.SYSIN DD *           *
//*                 if not desired, or comment out both job steps.     *
//*     LKEDSIO     Link the function package                          *
//*     LKEDHKP     Link Termination Exit EAGIOHKP                     *
//*     ASSMIRP     Assemble IRXPARMS REXX Parameter Module            *
//*     LKEDIRP     Link     IRXPARMS                                  *
//*     ASSMIRT     Assemble IRXTSPRM REXX Parameter Module            *
//*     LKEDIRT     Link     IRXTSPRM                                  *
//*     VERTSYS     Create temporary SYSEXEC                           *
//*     VERTUSE     Create temporary DSN for verification              *
//*     COPYEXEC    Copy REXX exec to SYSEXEC                          *
```

```
//*     GOREXMVS    Execute verification under IRXJCL                *
//*     GOREXTSO    Execute verification under IKJEFT01              *
//*                 Note:                                            *
//*                   This setup does not require APF authorization. *
//*                   If you define EXECTERM together with a non-APF *
//*                   load library, you receive a system 66D abend.  *
//*                                                                  *
//*   Change Activity: New 020230 - otm                              *
//*                                                                  *
//*------------------------------------------------------------------*
//* PROC section, make your updates here:                            *
//*------------------------------------------------------------------*
//MAKESIO PROC OBJLIB='uid.REXX.SIO130.OBJECT',  Object library
//            LOADLB='uid.REXX.SIO130.LOAD',    Load library
//            SMACLB='SYS1.MACLIB',             Sample library
//            CSSLIB='SYS1.CSSLIB',             Systems Maclib
//            REXPRT='X'                        EXEC output
//*------------------------------------------------------------------*
//*       Allocate load library - Discard * to activate Job step     *
//*------------------------------------------------------------------*
//*ALLOLOAD EXEC PGM=IEFBR14
//*ALLODD   DD DSN=&LOADLB,DISP=(NEW,CATLG,DELETE),
//*         UNIT=SYSDA,SPACE=(TRK,(5,1,10)),
//*         DCB=(RECFM=U,LRECL=0,BLKSIZE=23200,DSORG=PO)
//*------------------------------------------------------------------*
//*       Compress load library - Comment out if not desired         *
//*------------------------------------------------------------------*
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//IN       DD DISP=SHR,DSN=&LOADLB
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,2))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(1,2))
//SYSIN    DD DUMMY
//*------------------------------------------------------------------*
//*       Link Directory                                             *
//*------------------------------------------------------------------*
//LKEDSIO  EXEC PGM=IEWL,
//             PARM='LIST,XREF,LET,RENT',
//             COND=(4,LT)
//SYSLIB   DD DISP=SHR,DSN=&OBJLIB
//         DD DISP=SHR,DSN=&CSSLIB
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DISP=SHR,DSN=&LOADLB
//SYSLIN   DD DUMMY
//*------------------------------------------------------------------*
//*       Link Exec Termination Exit                                 *
//*------------------------------------------------------------------*
//LKEDHKP  EXEC PGM=IEWL,
//             PARM='LIST,XREF,LET,RENT',
//             COND=(4,LT)
//SYSLIB   DD DISP=SHR,DSN=&OBJLIB
//         DD DISP=SHR,DSN=&CSSLIB
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DISP=SHR,DSN=&LOADLB
//SYSLIN   DD DUMMY
```

```
//*-------------------------------------------------------------------*
//*   Assemble and linkedit the IRXPARMS REXX Parameter Module        *
//*   Parameter Module consists of:                                   *
//*     PARMBLOCK                                                     *
//*     MODULE NAME TABLE                                             *
//*     HOST COMMAND ENVIRONMENT TABLE                                *
//*     FUNCTION PACKAGE TABLE                                        *
//*-------------------------------------------------------------------*
//ASSMIRP  EXEC PGM=ASMA90,REGION=0K,
//             PARM=(NODECK,OBJECT,SIZE(200K)),
//             COND=(4,LT)
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//SYSPUNCH DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLIN   DD DISP=SHR,DSN=&OBJLIB(IRXPARMS)
//SYSIN    DD DUMMY
//*-------------------------------------------------------------------*
//*       LKED Module IRXPARMS                                        *
//*-------------------------------------------------------------------*
//LKEDIRP  EXEC PGM=IEWL,
//             PARM='LIST,XREF,LET,RENT',
//             COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DISP=SHR,DSN=&LOADLB(IRXPARMS)
//SYSLIN   DD DISP=SHR,DSN=&OBJLIB(IRXPARMS)
//*-------------------------------------------------------------------*
//*   Assemble and linkedit the IRXTSPRM REXX Parameter Module        *
//*   Parameter Module consists of:                                   *
//*     PARMBLOCK                                                     *
//*     MODULE NAME TABLE                                             *
//*     HOST COMMAND ENVIRONMENT TABLE                                *
//*     FUNCTION PACKAGE TABLE                                        *
//*-------------------------------------------------------------------*
//ASSMIRT  EXEC PGM=ASMA90,REGION=0K,
//             PARM=(NODECK,OBJECT,SIZE(200K)),
//             COND=(4,LT)
//SYSLIB   DD DISP=SHR,DSN=&SMACLB
//SYSPUNCH DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLIN   DD DISP=SHR,DSN=&OBJLIB(IRXTSPRM)
//SYSIN    DD DUMMY
//*-------------------------------------------------------------------*
//*       LKED IRXTSPRM                                               *
//*-------------------------------------------------------------------*
//LKEDIRT  EXEC PGM=IEWL,
//             PARM='LIST,XREF,LET,RENT',
//             COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DISP=SHR,DSN=&LOADLB(IRXTSPRM)
//SYSLIN   DD DISP=SHR,DSN=&OBJLIB(IRXTSPRM)
//*-------------------------------------------------------------------*
```

```
//*       Verification Step:                                          *
//*         It allocates two temporary data sets:                     *
//*          1. SYSEXEC                                               *
//*          2. Used to write data to it and to read data from it     *
//*-------------------------------------------------------------------*
//ALLOCTMP EXEC PGM=IEFBR14,
//             COND=(4,LT)
//VERTSYS  DD DSN=&&SYSEXEC,
//         DISP=(NEW,PASS,DELETE),
//         UNIT=SYSDA,SPACE=(TRK,(1,1,1)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000,DSORG=PO)
//VERTUSE  DD DSN=&&USE2VER,
//         DISP=(NEW,PASS,DELETE),
//         UNIT=SYSDA,SPACE=(TRK,(1,1)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS)
//*-------------------------------------------------------------------*
//*  COPY Source Exec to SYSEXEC                                      *
//*-------------------------------------------------------------------*
//COPYEXEC EXEC PGM=IEBGENER,
//             COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD DISP=(SHR,PASS),DSN=&&SYSEXEC(GOVERIFY)
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD DUMMY
//*-------------------------------------------------------------------*
//*  Execute GOVERIFY EXEC under IRXJCL                               *
//*-------------------------------------------------------------------*
//GOREXMVS EXEC PGM=IRXJCL,PARM='GOVERIFY VERDSN (IRXJCL',
//             COND=(4,LT)
//STEPLIB  DD DISP=SHR,DSN=&LOADLB
//SYSEXEC  DD DISP=(SHR,PASS),DSN=&&SYSEXEC
//VERDSN   DD DISP=(SHR,PASS),DSN=&&USE2VER
//SYSTSPRT DD SYSOUT=&REXPRT,DCB=(RECFM=FB,LRECL=120,BLKSIZE=1200)
//SYSPRINT DD SYSOUT=&REXPRT
//SYSTSIN  DD DUMMY
//*-------------------------------------------------------------------*
//*  Execute GOVERIFY EXEC under IKJEFT01                             *
//*-------------------------------------------------------------------*
//GOREXTSO EXEC PGM=IKJEFT01,PARM='GOVERIFY VERDSN (IKJEFT01',
//             COND=(4,LT)
//STEPLIB  DD DISP=SHR,DSN=&LOADLB
//SYSEXEC  DD DISP=(SHR,PASS),DSN=&&SYSEXEC
//VERDSN   DD DISP=(SHR,PASS),DSN=&&USE2VER
//SYSTSPRT DD SYSOUT=&REXPRT,DCB=(RECFM=FB,LRECL=120,BLKSIZE=1200)
//SYSPRINT DD SYSOUT=&REXPRT
//SYSTSIN  DD DUMMY
//*-------------------------------------------------------------------*
//*  Job Procedure End                                                *
//*-------------------------------------------------------------------*
//         PEND
//*-------------------------------------------------------------------*
//MAKEIT EXEC MAKESIO
//*-------------------------------------------------------------------*
//*                                                                   *
```

```
//* Source inputs for all Job steps                                    *
//*                                                                    *
//*--------------------------------------------------------------------*
//*        Compress SYSIN                                              *
//*--------------------------------------------------------------------*
//COMPRESS.SYSIN DD *
  COPY INDD=IN,OUTDD=IN
/*
//*--------------------------------------------------------------------*
//*        Link EAGEFSIO Function Package                              *
//*--------------------------------------------------------------------*
//LKEDSIO.SYSLIN DD *
   INCLUDE SYSLIB(EAGEFSIO)
   INCLUDE SYSLIB(EAGIOCHI)
   INCLUDE SYSLIB(EAGIOCHO)
   INCLUDE SYSLIB(EAGIOCLS)
   INCLUDE SYSLIB(EAGIODYN)
   INCLUDE SYSLIB(EAGIOGET)
   INCLUDE SYSLIB(EAGIOGFE)
   INCLUDE SYSLIB(EAGIOGNM)
   INCLUDE SYSLIB(EAGIOHKP)
   INCLUDE SYSLIB(EAGIOLNI)
   INCLUDE SYSLIB(EAGIOLNO)
   INCLUDE SYSLIB(EAGIOLNS)
   INCLUDE SYSLIB(EAGIOMSG)
   INCLUDE SYSLIB(EAGIOOPE)
   INCLUDE SYSLIB(EAGIOPUT)
   INCLUDE SYSLIB(EAGIORET)
   INCLUDE SYSLIB(EAGIOSTR)
   MODE AMODE(31),RMODE(ANY)
   NAME EAGEFSIO(R)
/*
//*--------------------------------------------------------------------*
//*       Link Exec Termination Exit                                   *
//*--------------------------------------------------------------------*
//LKEDHKP.SYSLIN DD *
   ENTRY EAGIOHKP
   INCLUDE SYSLIB(EAGIOHKP)
   MODE AMODE(31),RMODE(ANY)
   NAME EAGIOHKP(R)
/*
//*--------------------------------------------------------------------*
//*       IRXPARMS                                                     *
//*--------------------------------------------------------------------*
//ASSMIRP.SYSIN  DD *
         TITLE 'IRXPARMS REXX System RXAPI Parameter Module for MVS'
************************************************************************
*
* Module name : IRXPARMS
*
*   LICENSED MATERIALS - PROPERTY OF IBM
*   THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
*   5685-025 (C) COPYRIGHT IBM CORP. 1988, 1992
*   SEE COPYRIGHT INSTRUCTIONS
*
```

```
* Description : Parameter Module for MVS.
*
* Function    : Declare the system
*               - PARMBLOCK  = parameter block
*               - MODNAMET   = module name table
*               - SUBCOMTB   = subcommand handler table
*               - PACKTB     = function package table
*
* Notes       : This module does not contain any executable code,
*               only data.
*
* Setup       : for REXX Stream I/O
*
* Created     : otm 12/05/2001
*
*********************************************************************
                                                                EJECT
IRXPARMS AMODE 31
IRXPARMS RMODE ANY
IRXPARMS CSECT
                                                              SPACE 1
*********************************************************************
* Parmblock
*-------------------------------------------------------------------
* Flags / Masks:
*-------------------------------------------------------------------
*  Byte 1 of PARMBLOCK Flags
* TSOFL           1... ....   Integrate with TSO flag
*                 .1.. ....   Reserved
* CMDSOFL         ..1. ....   Command search order flag
* FUNCSOFL        ...1 ....   Function/subroutine search order flag
* NOSTKFL         .... 1...   No data stack flag
* NOREADFL        .... .1..   No read flag
* NOWRTFL         .... ..1.   No write flag
* NEWSTKFL        .... ...1   New data stack flag
*-------------------------------------------------------------------
*  Byte 2 of PARMBLOCK Flags
* USERPKFL        1... ....   User external function package flag
* LOCPKFL         .1.. ....   Local external function package flag
* SYSPKFL         ..1. ....   System external function package flag
* NEWSCFL         ...1 ....   New subcommand table flag
* CLOSEXFL        .... 1...   Close exec data set flag
* NOESTAE         .... .1..   No recovery ESTAE flag
* RENTRANT        .... ..1.   Reentrant REXX environment flag
* NOPMSGS         .... ...1   No primary messages flag
*-------------------------------------------------------------------
*  Byte 3 of PARMBLOCK Flags
* ALTMSGS         1... ....   Issue alternate messages flag
* SPSHARE         .1.. ....   Subpool storage is shared flag
* STORFL          ..1. ....   STORAGE function flag
* NOLOADDD        ...1 ....   Do not load from the sys-lvl EXEC DDNAME.
* NOMSGWTO_MASK   .... 1...   MVS, do not issue err msgs with WTO srv
* NOMSGIO_MASK    .... .1..   MVS, do not issue err msgs to OUTDD
*                 .... ..1.   Reserved
*                 .... ...1   Reserved
```

```
*--------------------------------------------------------------------
*  Byte 4 of PARMBLOCK Flags
*                 1111 1111   Reserved
**********************************************************************
                                                                SPACE 1
PARMBLOCK           EQU *
PARMBLOCK_ID        DC CL8'IRXPARMS'    Initialize the ID field
PARMBLOCK_VERSION   DC CL4'0200'        Initialize the Version field
PARMBLOCK_LANGUAGE  DC CL3'ENU'         Language is American English
RSVD001             DC CL1' '           Reserved
PARMBLOCK_MODNAMET  DC A(MODNAMET)      Addr of Module Names Table
PARMBLOCK_SUBCOMTB  DC A(SUBCOMTB)      Addr of Subcom Table Header
PARMBLOCK_PACKTB    DC A(PACKTB_HEADER) Addr of Package Table Header
PARMBLOCK_PARSETOK  DC CL8' '           Parse Source Token
PARMBLOCK_FLAGS     DC X'0000C000'      Set the flags
PARMBLOCK_MASKS     DC X'FFFFFFFF'      Set all masks
PARMBLOCK_SUBPOOL   DC AL4(0)           Subpool to use
PARMBLOCK_ADDRSPN   DC CL8'MVS'         Address space name
PARMBLOCK_FFFF      DC X'FFFFFFFFFFFFFFFF'
                                                                SPACE 1
**********************************************************************
* Module Name Table
**********************************************************************
                                                                SPACE 1
MODNAMET            EQU *
MODNAMET_DDS        EQU *               DDs used by REXX:
MODNAMET_INDD       DC CL8'SYSTSIN '    - Specify the input DD
MODNAMET_OUTDD      DC CL8'SYSTSPRT'    - Specify the output DD
MODNAMET_LOADDD     DC CL8'SYSEXEC '    - Specify the load exec DD
*
MODNAMET_ROUTINES   EQU *               Routines used by REXX
*                                        Blank indicates default to
*                                        previous environment's settings
MODNAMET_IOROUT     DC CL8'         '   The I/O Routine
MODNAMET_EXROUT     DC CL8'         '   The exec load Routine
MODNAMET_GETFREER   DC CL8'         '   The GETMAIN/FREEMAIN Routine
MODNAMET_EXECINIT   DC CL8'         '   The EXEC initialization rtn.
MODNAMET_ATTNROUT   DC CL8'         '   The Attention Routine
MODNAMET_STACKRT    DC CL8'         '   The Data Stack Routine
MODNAMET_IRXEXECX   DC CL8'         '   The IRXEXEC exit Routine
MODNAMET_IDROUT     DC CL8'         '   The userid Routine
MODNAMET_MSGIDRT    DC CL8'         '   The message id Routine
***MODNAMET_EXECTERM  DC CL8'EAGIOHKP'    EXEC Termination Stream I/O
MODNAMET_EXECTERM   DC CL8'         '   Name of the EXEC Termination RTN
MODNAMET_FFFF       DC X'FFFFFFFFFFFFFFFF'
                                                                SPACE 1
**********************************************************************
* Subcommand Table
**********************************************************************
                                                                SPACE 1
SUBCOMTB            EQU *
SUBCOMTB_HEADER     EQU *               Set up the SUBCOMTB Header
SUBCOMTB_FIRST      DC A(SUBCOMTB_ENTRIES)  Addr of first SUBCOMTB entry
SUBCOMTB_TOTAL      DC F'10'            Total number of entries
SUBCOMTB_USED       DC F'10'            Number of entries used
```

```
SUBCOMTB_LENGTH    DC F'32'            Length of each entry
SUBCOMTB_INITIAL   DC CL8'MVS'         Initial Subcommand environment
RSVD007            DC X'0000000000000000' Reserved
SUBCOMTB_FFFF      DC X'FFFFFFFFFFFFFFFF'
*
SUBCOMTB_ENTRIES   EQU *               Start of SUBCOMTB entries
*
                   DC CL8'MVS'         subcommand environment name
                   DC CL8'IRXSTAM'     handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'LINK'        subcommand environment name
                   DC CL8'IRXSTAM'     handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'ATTACH'      subcommand environment name
                   DC CL8'IRXSTAM'     handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'CPICOMM'     subcommand environment name
                   DC CL8'IRXAPPC'     handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'LU62'        subcommand environment name
                   DC CL8'IRXAPPC'     handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'LINKMVS'     subcommand environment name
                   DC CL8'IRXSTAMP'    handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'LINKPGM'     subcommand environment name
                   DC CL8'IRXSTAMP'    handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'ATTCHMVS'    subcommand environment name
                   DC CL8'IRXSTAMP'    handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'ATTCHPGM'    subcommand environment name
                   DC CL8'IRXSTAMP'    handler routine name
                   DC CL16' '          subcommand token
*
                   DC CL8'APPCMVS'     subcommand environment name
                   DC CL8'IRXAPPC'     handler routine name
                   DC CL16' '          subcommand token
                                                              SPACE 1
*********************************************************************
* Function Package Table
*********************************************************************
                                                              SPACE 1
PACKTB             EQU *
PACKTB_HEADER      EQU *               Set up the PACKTB Header
*
PACKTB_USER_FIRST  DC A(USER_PACKTB_ENTRIES) Addr of first User entry
```

```
PACKTB_USER_TOTAL  DC F'1'             Total number of User entries
PACKTB_USER_USED   DC F'1'             Number of User entries in use
*
PACKTB_LOCAL_FIRST DC A(LOCAL_PACKTB_ENTRIES) Addr of first Local entry
PACKTB_LOCAL_TOTAL DC F'1'             Total number of Local entries
PACKTB_LOCAL_USED  DC F'1'             Number of Local entries in use
*
PACKTB_SYSTEM_FIRST DC A(SYSTEM_PACKTB_ENTRIES) Addr of 1st System entry
PACKTB_SYSTEM_TOTAL DC F'2'            Total number of System entries
PACKTB_SYSTEM_USED  DC F'2'            Number of System entries in use
*
PACKTB_LENGTH      DC F'8'             Length of each PACKTB entry
PACKTB_FFFF        DC X'FFFFFFFFFFFFFFFF'
*
SYSTEM_PACKTB_ENTRIES EQU *
                   DC CL8'IRXEFMVS'    external MVS functions
                   DC CL8'EAGEFSIO'    REXX Stream IO
                                                              SPACE 1
LOCAL_PACKTB_ENTRIES EQU *
                   DC CL8'IRXFLOC'     Default Local Function Package
*
USER_PACKTB_ENTRIES EQU *
                   DC CL8'IRXFUSER'    Default User Function Package
                                                              SPACE 1
*********************************************************************
* Patch area
*********************************************************************
                                                              SPACE 1
                   DC    C'PATCH AREA - IRXPARMS'
                   DS    8F            Patch area
                                                              SPACE 1
IRXPARMS CSECT
         END IRXPARMS
/*
//*-------------------------------------------------------------------*
//*       IRXTSPRM                                                    *
//*-------------------------------------------------------------------*
//ASSMIRT.SYSIN  DD *
         TITLE 'IRXTSPRM REXX System RXAPI Parameter Module for TSO'
*********************************************************************
*
* Module name : IRXTSPRM
*
*   LICENSED MATERIALS - PROPERTY OF IBM
*   THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
*   5685-025 (C) COPYRIGHT IBM CORP. 1988, 1992
*   SEE COPYRIGHT INSTRUCTIONS
*
* Description : Parameter Module for TSO.
*
* Function    : Declare the system
*               - PARMBLOCK  = parameter block
*               - MODNAMET   = module name table
*               - SUBCOMTB   = subcommand handler table
*               - PACKTB     = function package table
```

```
*
* Notes       : This module does not contain any executable code,
*               only data.
*
* Setup       : for REXX Stream I/O
*
* Created     : otm 12/05/2001
*
*********************************************************************
                                                                EJECT
IRXTSPRM AMODE 31
IRXTSPRM RMODE ANY
IRXTSPRM CSECT
                                                              SPACE 1
*********************************************************************
* Parmblock
*-------------------------------------------------------------------
* Flags / Masks:
*-------------------------------------------------------------------
*  Byte 1 of PARMBLOCK Flags
* TSOFL           1... ....   Integrate with TSO flag
*                 .1.. ....   Reserved
* CMDSOFL         ..1. ....   Command search order flag
* FUNCSOFL        ...1 ....   Function/subroutine search order flag
* NOSTKFL         .... 1...   No data stack flag
* NOREADFL        .... .1..   No read flag
* NOWRTFL         .... ..1.   No write flag
* NEWSTKFL        .... ...1   New data stack flag
*-------------------------------------------------------------------
*  Byte 2 of PARMBLOCK Flags
* USERPKFL        1... ....   User external function package flag
* LOCPKFL         .1.. ....   Local external function package flag
* SYSPKFL         ..1. ....   System external function package flag
* NEWSCFL         ...1 ....   New subcommand table flag
* CLOSEXFL        .... 1...   Close exec data set flag
* NOESTAE         .... .1..   No recovery ESTAE flag
* RENTRANT        .... ..1.   Reentrant REXX environment flag
* NOPMSGS         .... ...1   No primary messages flag
*-------------------------------------------------------------------
*  Byte 3 of PARMBLOCK Flags
* ALTMSGS         1... ....   Issue alternate messages flag
* SPSHARE         .1.. ....   Subpool storage is shared flag
* STORFL          ..1. ....   STORAGE function flag
* NOLOADDD        ...1 ....   Do not load from the sys-lvl EXEC DDNAME.
* NOMSGWTO_MASK   .... 1...   MVS, do not issue err msgs with WTO srv
* NOMSGIO_MASK    .... .1..   MVS, do not issue err msgs to OUTDD
*                 .... ..1.   Reserved
*                 .... ...1   Reserved
*-------------------------------------------------------------------
*  Byte 4 of PARMBLOCK Flags
*                 1111 1111   Reserved
*********************************************************************
                                                              SPACE 1
PARMBLOCK          EQU *
PARMBLOCK_ID       DC CL8'IRXPARMS'    Initialize the ID field
```

```
PARMBLOCK_VERSION  DC CL4'0200'        Initialize the Version field
PARMBLOCK_LANGUAGE DC CL3'ENU'         Language is American English
RSVD001            DC CL1' '           Reserved
PARMBLOCK_MODNAMET DC A(MODNAMET)      Addr of Module Names Table
PARMBLOCK_SUBCOMTB DC A(SUBCOMTB)      Addr of Subcom Table Header
PARMBLOCK_PACKTB   DC A(PACKTB_HEADER) Addr of Package Table Header
PARMBLOCK_PARSETOK DC CL8' '           Parse Source Token
PARMBLOCK_FLAGS    DC X'8000C000'      Set the flags
PARMBLOCK_MASKS    DC X'FFFFFFFF'      Set all masks
PARMBLOCK_SUBPOOL  DC AL4(78)          Subpool to use
PARMBLOCK_ADDRSPN  DC CL8'TSO/E'       Address space name
PARMBLOCK_FFFF     DC X'FFFFFFFFFFFFFFFF'
                                                                    SPACE 1
*********************************************************************
* Module Name Table
*********************************************************************
                                                                    SPACE 1
MODNAMET           EQU *
MODNAMET_DDS       EQU *               DDs used by REXX:
MODNAMET_INDD      DC CL8'SYSTSIN '    - Specify the input DD
MODNAMET_OUTDD     DC CL8'SYSTSPRT'    - Specify the output DD
MODNAMET_LOADDD    DC CL8'SYSEXEC '    - Specify the load exec DD
*
MODNAMET_ROUTINES  EQU *               Routines used by REXX
*                                       Blank indicates default to
*                                       previous environment's settings
MODNAMET_IOROUT    DC CL8'        '    The I/O Routine
MODNAMET_EXROUT    DC CL8'        '    The exec load Routine
MODNAMET_GETFREER  DC CL8'        '    The GETMAIN/FREEMAIN Routine
MODNAMET_EXECINIT  DC CL8'        '    The EXEC initialization rtn.
MODNAMET_ATTNROUT  DC CL8'        '    The Attention Routine
MODNAMET_STACKRT   DC CL8'        '    The Data Stack Routine
MODNAMET_IRXEXECX  DC CL8'        '    The IRXEXEC exit Routine
MODNAMET_IDROUT    DC CL8'        '    The userid Routine
MODNAMET_MSGIDRT   DC CL8'        '    The message id Routine
***MODNAMET_EXECTERM  DC CL8'EAGIOHKP'    EXEC Termination Stream I/O
MODNAMET_EXECTERM  DC CL8'        '    Name of the EXEC Termination RTN
MODNAMET_FFFF      DC X'FFFFFFFFFFFFFFFF'
                                                                    SPACE 1
*********************************************************************
* Subcommand Table
*********************************************************************
                                                                    SPACE 1
SUBCOMTB           EQU *
SUBCOMTB_HEADER    EQU *               Set up the SUBCOMTB Header
SUBCOMTB_FIRST     DC A(SUBCOMTB_ENTRIES)  Addr of first SUBCOMTB entry
SUBCOMTB_TOTAL     DC F'12'            Total number of entries
SUBCOMTB_USED      DC F'12'            Number of entries used
SUBCOMTB_LENGTH    DC F'32'            Length of each entry
SUBCOMTB_INITIAL   DC CL8'TSO'         Initial Subcommand environment
RSVD007            DC X'0000000000000000' Reserved
SUBCOMTB_FFFF      DC X'FFFFFFFFFFFFFFFF'
*
SUBCOMTB_ENTRIES   EQU *               Start of SUBCOMTB entries
*
```

```
                  DC CL8'MVS'         subcommand environment name
                  DC CL8'IRXSTAM'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'TSO'         subcommand environment name
                  DC CL8'IRXSTAM'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'LINK'        subcommand environment name
                  DC CL8'IRXSTAM'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'ATTACH'      subcommand environment name
                  DC CL8'IRXSTAM'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'CONSOLE'     subcommand environment name
                  DC CL8'IRXSTAM'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'CPICOMM'     subcommand environment name
                  DC CL8'IRXAPPC'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'LU62'        subcommand environment name
                  DC CL8'IRXAPPC'     handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'LINKMVS'     subcommand environment name
                  DC CL8'IRXSTAMP'    handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'LINKPGM'     subcommand environment name
                  DC CL8'IRXSTAMP'    handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'ATTCHMVS'    subcommand environment name
                  DC CL8'IRXSTAMP'    handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'ATTCHPGM'    subcommand environment name
                  DC CL8'IRXSTAMP'    handler routine name
                  DC CL16' '          subcommand token
*
                  DC CL8'APPCMVS'     subcommand environment name
                  DC CL8'IRXAPPC'     handler routine name
                  DC CL16' '          subcommand token
                                                              SPACE 1
*********************************************************************
* Function Package Table
*********************************************************************
                                                              SPACE 1
PACKTB            EQU *
PACKTB_HEADER     EQU *               Set up the PACKTB Header
*
```

```
PACKTB_USER_FIRST  DC A(USER_PACKTB_ENTRIES) Addr of first User entry
PACKTB_USER_TOTAL  DC F'1'             Total number of User entries
PACKTB_USER_USED   DC F'1'             Number of User entries in use
*
PACKTB_LOCAL_FIRST DC A(LOCAL_PACKTB_ENTRIES) Addr of first Local entry
PACKTB_LOCAL_TOTAL DC F'1'             Total number of Local entries
PACKTB_LOCAL_USED  DC F'1'             Number of Local entries in use
*
PACKTB_SYSTEM_FIRST DC A(SYSTEM_PACKTB_ENTRIES) Addr of first System ent
PACKTB_SYSTEM_TOTAL DC F'3'            Total number of System entries
PACKTB_SYSTEM_USED  DC F'3'            Number of System entries in use
*
PACKTB_LENGTH      DC F'8'             Length of each PACKTB entry
PACKTB_FFFF        DC X'FFFFFFFFFFFFFFFF'
*
SYSTEM_PACKTB_ENTRIES EQU *
                   DC CL8'IRXEFMVS'    external MVS functions
                   DC CL8'IRXEFPCK'    external MVS functions
                   DC CL8'EAGEFSIO'    REXX Stream I/O
                                                              SPACE 1
LOCAL_PACKTB_ENTRIES EQU *
                   DC CL8'IRXFLOC'     Default Local Function Package
*
USER_PACKTB_ENTRIES EQU *
                   DC CL8'IRXFUSER'    Default User Function Package
                                                              SPACE 1
********************************************************************
* Patch area
********************************************************************
                                                              SPACE 1
                   DC    C'PATCH AREA - IRXTSPRM'
                   DS    8F            Patch area
                                                              SPACE 1
IRXTSPRM CSECT
         END IRXTSPRM
/*
//*-------------------------------------------------------------------*
//*       REXX EXEC Device under Test                                 *
//*-------------------------------------------------------------------*
//COPYEXEC.SYSUT1 DD DATA,DLM=§§
/*-------------------------------------------------------------------*/
/* REXX sample exec for REXX STREAM I/O Verfication Test             */
/*      All Stream I/O commands are used to Read, Write and detect   */
/*      EOF. Closing files is performed implicit and by STREAM.      */
/*-------------------------------------------------------------------*/

Signal on Syntax;
Arg use_ddname . '('host .;                   /* Pre-alloc DDN (host*/
  Parse version ver;                          /* Get REXX Version   */
  Parse source  src;                          /* Get REXX Source ID */
  Say '';
  /*-----------------------------------------------------------------*/
  /* We need the pre-allocated DD Name by parm                       */
  /*-----------------------------------------------------------------*/
  Say 'Running Verification EXEC' word(src,3) 'under' host!!':';
```

```
If use_ddname='' then do;
  Say 'Error, you did not specify a DD Name as argument.';
  Exit 4;
End;
Else use_ddname='&'!!use_ddname;             /* Prefix DDN sign    */

/*--------------------------------------------------------------------*/
/* Write out 2 records with LINEOUT                                   */
/*--------------------------------------------------------------------*/
Say '  REXX Stream I/O uses DD Name' use_ddname;
Say '  LINEOUT  the following strings to file:';
Say '   ' ver;
Say '   ' src;
rc=lineout(use_ddname,ver);                  /* Write out to DDN   */
rc=lineout(use_ddname,src);
rc=lineout(use_ddname);                      /* Close the file     */

/*--------------------------------------------------------------------*/
/* Read in the 2 records with LINEIN until EOF                        */
/*--------------------------------------------------------------------*/
Say '  LINEIN   the created file up to EOF:';
i=0;
Do While LINES(use_ddname)=1;                /* Up to EOF          */
  i=i+1;
  Say '   Record' right(i,3)':' linein(use_ddname);
End;
rc=STREAM(use_ddname,"C","CLOSE");           /* Close the file     */
Say '';

/*--------------------------------------------------------------------*/
/* Write out 50 bytes with CHAROUT                                    */
/*--------------------------------------------------------------------*/
string ='<>';                                /* Write string       */
times  =120;                                 /* Multiple of above  */
portion=50;                                  /* CharIn portion     */
Say '  CHAROUT ' times 'times "'string '" to file:';
rc=charout(use_ddname,copies(string,times)); /* Write out to DDN   */
rc=lineout(use_ddname);                      /* Close the file     */

/*--------------------------------------------------------------------*/
/* Read in 50 bytes portions with CHARIN until EOF                    */
/*--------------------------------------------------------------------*/
Say '  CHARIN  ' portion 'bytes portions up to EOF:';
i=0;
Do While CHARS(use_ddname)=1;                /* Up to EOF          */
  i=i+1;
  Say '  Portion' right(i,3)':' charin(use_ddname,,portion);
End;
rc=STREAM(use_ddname,"C","CLOSE");           /* Close the file     */

/*--------------------------------------------------------------------*/
/* End of Verification                                                */
/*--------------------------------------------------------------------*/
Say '';
Say 'End of Verification EXEC' word(src,3);
```

```
  rc=STREAM(use_ddname,"C","CLOSE ALL");       /* Terminate I/O      */
Exit;

/*--------------------------------------------------------------------*/
/* Syntax routine                                                     */
/*--------------------------------------------------------------------*/
Syntax:;
  Say 'Syntax received in sourceline' sigl;
  Say '->' sourceline(sigl);
Exit rc;
§§
//
```

# Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100
70569 Stuttgart
Germany

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

## Trademarks

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

MVS
S/390
z/OS

Other company, product, and service names may be trademarks or service marks of others.

# Glossary of z/OS terms

**allocate.** To assign a resource, such as a disk file, to a specific task. Contrast with deallocate.

**authorized program facility (APF).** Allows to identify system programs and user programs that can use sensitive system functions, and restricts the use of such functions to APF-authorized programs.

**data set.** The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

**ddname.** Data definition name.

**DD statement.** Data definition statement.

**HFS data set.** A hierarchical file system data set, which is used to store, and is essentially identified with, a file system.

**partitioned data set (PDS).** A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. A partitioned data set has a directory that contains information about each member. Each member can be accessed individually by its unique 1- to 8-character name.

**partitioned data set extended (PDSE).** A partitioned data set managed by the Storage Management Subsystem (SMS). Similar to PDS, but with a number of enhancements.

**sequential data set.** A data set in which the contents are arranged in successive physical order and are stored as an entity. The data set can contain data, text, a program, or part of a program. Contrast with partitioned data set (PDS).

# Index

## P



## Q



## R



## S



## T



## V



## W



## Z

# Readers' Comments — We'd Like to Hear from You

**z/OS**
**Stream I/O for TSO/E REXX**
**Version 1**

**Overall, how satisfied are you with the information in this book?**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Overall satisfaction | ☐ | ☐ | ☐ | ☐ | ☐ |

**How satisfied are you that the information in this book is:**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Accurate | ☐ | ☐ | ☐ | ☐ | ☐ |
| Complete | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to find | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to understand | ☐ | ☐ | ☐ | ☐ | ☐ |
| Well organized | ☐ | ☐ | ☐ | ☐ | ☐ |
| Applicable to your tasks | ☐ | ☐ | ☐ | ☐ | ☐ |

**Please tell us how we can improve this book:**

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

## Readers' Comments — We'd Like to Hear from You

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Str. 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line