REXX/WAIT
VM REXX Function Package for Event Handling and ASCII/EBCDIC Conversion

Reference Manual

Document Number REXX/WAIT 2.12

7 April 1994

Rainer F. Hauser

IBM Research Laboratory
Zurich, Switzerland
RFH at ZURLVM1
rfh@zurich.ibm.com

Contents

Figures

1.0 Introduction

REXX/WAIT is an external REXX function package that, when loaded as a nucleus extension, provides a central wait and a test function, three central functions to set, query and reset values and four functions related to ASCII/EBCDIC translation.

Functions for waiting are needed by many programs. It is not a satisfying solution that each program provides its own wait function because this allows waiting for events from this program only but not for events from any other program. Therefore, waiting for events as well as polling for pending events must be offered through a central function to which also any other program can register their events. In a multi-processing environment, this statement is still true although the lack of such a function is less of a problem, because different processes can wait for different events. However, a single process may still need to wait for multiple events. REXX/WAIT provides such a central wait function and a test function for programming in REXX for CMS.

The function WAIT for waiting and the function TEST for polling alone are not sufficient, because a REXX program may need to set the default waiting parameters or other values provided by a program that registered its events. REXX only supports functions with up to ten arguments. If a REXX program needs to wait for more than ten events, it needs a mechanism to let the WAIT function wait for all events with default waiting parameters. Even if a REXX program needs to wait for ten events or less, default waiting parameters are very convenient. Therefore, the REXX/WAIT program offers also three functions to set, query and reset such values.

The REXX/WAIT program contains also functions for ASCII/EBCDIC translation. These functions could have been provided by another function package. However, since waiting is most important to communication protocols where heterogeneous computers exchange information, the functions to translate ASCII strings to EBCDIC and vice versa were included into the same package.

Since REXX is a programming language available on many platforms, REXX/WAIT has been designed with other platforms in mind. Certain events such as SMSG's are obviously only available in a VM environment. However, other events such as MAIL and TIME are of a more general nature. In the long run, the REXX/WAIT functions should become part of the REXX language definition, as we believe.

The REXX/WAIT program is described in the next three chapters from the point of view of a REXX programmer using the REXX functions WAIT, TEST, SETVALUE, QUERYVALUE and RESETVALUE. The two chapters following them discuss the REXX/WAIT program from the point of view of an Assembler programmer writing a program (e.g., a REXX extension program) which registers its events such that the WAIT function allows waiting for them. The last chapter introduces the four REXX functions AC2EC, EC2AC, CTYPE and CTABLE used for translation from ASCII to EBCDIC and vice versa.

The REXX/WAIT package is available from the VMTOOLS conferencing disk as REXXWAIT PACKAGE. It consists of the REXXWAIT MODULE, CMS help files, this documentation, a REXX sample program and an Assembler sample program. (RXWAITFN MODULE, the older version of the module, should no longer be used since it will disappear as soon as possible.)

Before a REXX program can use any of the five functions related to waiting or the four functions related to translation, it must execute a command that will cause these functions to be installed as a nucleus extension. To install the functions WAIT, TEST, SETVALUE, QUERYVALUE and RESETVALUE as well as the functions AC2EC, EC2AC, CTYPE and CTABLE, enter:

```
'REXXWAIT LOAD'
```

When another program registers its events, REXX/WAIT gets automatically loaded, and explicit installation is not needed. (The command `'NUCXDROP REXXWAIT'` unloads (de-installs) the functions, but should only be used with care because other programs which registered their events get not informed that REXX/WAIT disappeared and their registrations got lost.)

To find out whether the functions have already been installed, the return code of `'REXXWAIT TEST'` tells whether the package has already been loaded (RC=0) or is not yet loaded (RC=1). However, this is usually not needed, because if the functions are already loaded, calling `'REXXWAIT LOAD'` has no effect. Therefore, a REXX program can always load them to make sure that they are available.

Note: Some pieces of code and most of the ideas for the basic event handlers provided by REXX/WAIT have directly been taken from or indirectly been influenced by WAKEUP 5.5 with the kind permission of its author C. R. Berghorn (BERGHORN at PKSMRVM), Poughkeepsie, NY. I would like to thank him for his co-operation and support.

2.0 Syntax of the Functions Related to Waiting

The package contains a first group of five REXX functions called WAIT, TEST, SETVALUE, QUERYVALUE and RESETVALUE. They are related to waiting for events, and their syntax is summarized in Figure 1. The WAIT function allows waiting for any of a set of events specified by the arguments. When the arguments are correct, the function does not return before at least one of the events has happened. The TEST function allows a REXX program to determine whether an event is already pending, but it never waits. The SETVALUE, QUERYVALUE and RESETVALUE allow communication with an event handler or other programs to set, query or reset internal values.

WAIT([**name1** [**args1**][**,...,namen** [**argsn**]]]**)**
TEST([**name1** [**args1**][**,...,namen** [**argsn**]]]**)**
SETVALUE(name [**args**]**)**
QUERYVALUE(name [**args**]**)**
RESETVALUE(name)

Figure 1. Summary of waiting related REXX functions in REXX/WAIT

The 'name' or 'namei' part of the arguments passed to these functions determines to which program the function call is directed. This program is called an event handler and it has to tell REXX/WAIT which of the functions it allows. (Some event handlers, called the basic event handlers, are registered by REXX/WAIT itself.) The rest of the arguments depends on the corresponding event handler program.

All functions return a result string with the different items separated by a blank such that it can be parsed by REXX in the usual way. The first item is always a return code indicating whether the function call was successful or not. For the WAIT and the TEST function, the second item of the result string is the name of the event handler that reported an event and either returned a string describing the event or a return code indicating the problem. The rest of the result string is defined by the event handler. The RESETVALUE function only returns the return code.

When called at 22:25:02 on June 3rd of the year 2002, and when no console interrupt is pending, the WAIT function call

```
WAIT('Cons','Time 5Sec') == '0 TIME 2002/06/03 22:25:07'
```

will return the shown result. The two arguments in the WAIT function call indicate the possible events that could terminate waiting. The result shows the return code zero indicating success, the keyword 'TIME' telling the REXX program which event happened, and the date and time. (If the console event would have happened first, the function would have returned 'CONS' after the return code.)

The general behavior of the five REXX functions is described in the following sections. For details about the basic event handlers, see the following chapter. It also contains examples of calls in a REXX program together with possible results.

2.1 The REXX Function WAIT

WAIT([**name1** [**args1**][**,...,namen** [**argsn**]]]**)**

Result: rc [namei [string]]

The function returns a return code, the name of the first event handler that reports an event and a description of the event as returned by the event handler. (Instead of 'namei', the special keyword 'ALL' without further arguments can be specified. In this case, the WAIT function waits for events from all event handlers. If WAIT is called without any names and arguments, the keyword 'ALL' is inserted as 'name1'.)

Examples

```
WAIT('Time 13:25:07') == '0 TIME 1992/06/03 13:25:07'
```

2.2 The REXX Function TEST

> **TEST(**[**name1** [**args1**][**,...,namen** [**argsn**]]]**)**
>
> Result: rc [namei [string]]

The function returns a return code, the name of the first event handler that reports an event and a description of the event as returned by the event handler, if an event was pending. Otherwise, it only returns the return code zero. (Instead of 'namei', the special keyword 'ALL' without further arguments can be specified. In this case, the TEST function tests for events from all event handlers. If TEST is called without any names and arguments, the keyword 'ALL' is inserted as 'name1'.)

Examples

```
TEST('Time 13:25:07') == '0'
TEST('Time 5 Secs')   == '0'
TEST('Time 0 Secs')   == '0 TIME 1993/09/14 10:15:27'
```

2.3 The REXX Function SETVALUE

> **SETVALUE(name** [**args**]**)**
>
> Result: rc [string]

The function passes the arguments in 'args' to the event handler and returns the return code and whatever the event handler returns.

Examples

```
SETVALUE('Time 13:25:07') == '0 FOREVER'   /* Perhaps */
```

2.4 The REXX Function QUERYVALUE

> **QUERYVALUE(name** [**args**]**)**
>
> Result: rc [string]

The function passes the arguments in 'args' to the event handler and returns the return code and whatever the event handler returns.

Examples

```
QUERYVALUE('Time Defaults') == '0 FOREVER'   /* Perhaps */
```

2.5 The REXX Function RESETVALUE

RESETVALUE(name)

Result: rc

The function does not allow any arguments in 'args' to be passed to the event handler and does not return a result other than the return code. Instead of 'name', the keyword 'ALL' can be used to call all event handlers.

Examples

```
RESETVALUE('Time') == '0'
```

2.6 Return Codes of the Functions

The first item in the result string of the five functions WAIT, TEST, SETVALUE, QUERYVALUE and RESETVALUE is the return code 'rc'. The return code is an integer, and values larger than zero mean a problem. The common return code are:

0	Normal return
1	Unknown or invalid event handler name
2	Event handler does not support function
3	Event handler does not support multiple calls
4	Reserved for future use
5	Parameter not supported on this platform
6	No more space available
7	Invalid argument string
8	Invalid result string
9	Unspecified error from event handler

All other return codes are equal or greater than 10 and come from the event handler directly.

2.7 General Remarks

This section presents some general remarks for REXX programmers using the functions related to waiting:

1. All parameters strings are usually translated to uppercase before being passed to an event handler. For example, `WAIT('CONS','TIME 10S')` is equivalent to `WAIT('cons','time 10s')`, and the event handler will only see the uppercase version when it did not specify that it wants to see the real argument string. The event handler FILE for example, accepts also lowercase file names.
2. The sequence of parameters entered to the WAIT and TEST functions determines their priorities. Thus, the sequence `WAIT('CONS','TIME 10S')` is better than the sequence `WAIT('TIME 10S','CONS')` if the REXX programmer is more interested in the console event than the timer event. If a console and a timer interrupt occur nearly at the same time, the first function call may return the console interrupt, and the second the timer interrupt.
3. The WAIT function and the TEST function allow only up to 200 characters of arguments to be passed to the event handlers. Longer arguments result in a non-zero return code.
4. The function calls `WAIT()`, `WAIT('')` and `WAIT('ALL')` (and similarly for the TEST function) are equivalent. However, also calls such as `WAIT('CONS READ','TIME 10S','ALL')` (where the keyword 'ALL' appears somewhere in the parameters) and `WAIT('CONS READ','ALL','TIME 10S')` are allowed.
5. Some of the registered event handlers allow setting defaults for waiting. These values are used by the WAIT and the TEST function when no explicitly specified parameters are entered. Be aware that also a call of the WAIT or TEST function with the keyword 'ALL' uses these values.

6. Some events must be consumed before the WAIT function is called again. The call `WAIT('CONS NOREAD')` leaves a line entered from the console in the stack, and the REXX program must read it. Otherwise, a loop calling the WAIT function becomes a busy loop.
7. The call `SETVALUE('MSG OFF')` (and similarly for WNG, SMSG, and OMSG) does not mean that no further events will be reported by the corresponding event handler. There may be pending messages still waiting in the queue. Use the RESETVALUE function to flush the queue.
8. The call `SETVALUE('MSG IUCV')` (and similarly for WNG and SMSG) issues `CP SET MSG IUCV` internally. Therefore, use the CP command SET in a REXX program using REXX/WAIT with care.
9. The call `QUERYVALUE('MSG LOST')` (and similarly for WNG, SMSG, and OMSG) returns the number of messages lost due to memory shortage and other reasons such as problems with the IUCV RECEIVE function. Whenever the interrupt handler for the message queue cannot obtain memory for receiving a message or cannot receive the message because of other problems, it discards the message and increases the corresponding counter for lost messages.
10. The event handler OMSG reports all messages received on the IUCV path to the CP service *MSG which are not handled by one of the other event handlers. To receive all possible messages via this event handler, call `SETVALUE('OMSG IUCV')` and use the CP command SET to set all or some of MSG, WNG, SMSG, EMSG, IMSG, VMCONIO, and CPCONIO to IUCV. The corresponding messages will be reported via OMSG in unmodified form together with the associated message class.
11. At most one event file can be specified in the WAIT and TEST functions. Thus, a call with more than one file such as `WAIT('FILE FIRST TIMEFILE A','FILE SECOND TIMEFILE A')` will terminate with a non-zero return code.
12. More than one time event can be specified in the WAIT function. Thus, the call with the two time events `WAIT('TIME ==:00:00','TIME ==:30:00')` will terminate when the next full or half hour is reached. Whenever more than one time event is specified, the first event in time that occurs will terminate the WAIT function. Thus, keep the parameters specifying time events together.

3.0 Reserved Names and Basic Event Names

Figure 2 summarizes all names of event handlers as registered by REXX/WAIT itself. Other programs (such as REXX/IUCV and REXX/SOCKETS) support it as well and export names of event handlers (such as 'IUCV' is exported by REXX/IUCV and 'SOCKET' by REXX/SOCKETS). For names of event handlers not registered by REXX/WAIT itself, see the respective documentation of the program exporting the name.

WAIT	Function package
ALL	Reserved name for all events
IMMCMD	Immediate Command events
CONS	Console events
WNG	Warning events
MSG	Message events
SMSG	Special message events
OMSG	Other message events
MAIL	Mail (virtual reader) events
FILE	Events specified in a file
TIME	Timer events
HOLIDAY	Holiday support

Figure 2. Summary of registered event handler names in REXX/WAIT

3.1 The Reserved Names WAIT and ALL

The names 'WAIT' and 'ALL' are reserved event handler names. The name 'WAIT' is used by the REXX/WAIT program, and the name 'ALL' is used for all event handler names.

QUERYVALUE('WAIT VERSION')

Result: rc 'REXX/WAIT' version date

The function returns the name ('REXX/WAIT'), the version number (currently 2.12) and the version date (currently 7 April 1994) of the REXX/WAIT function package.

QUERYVALUE('WAIT IDENTIFY')

Result: rc rscsid nodeid

The function returns the rscsid and nodeid obtained using the IDENTIFY command (i.e. the information in the SYSTEM NETID file.)

SETVALUE('WAIT DEBUG'|'NODEBUG')

Result: rc olddefaults

The function call sets the debugging defaults for the function package and returns the default settings active before the new values are set. If debugging is on, REXX/WAIT shows when internal waiting starts and terminates. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'NODEBUG'.

QUERYVALUE('WAIT DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the function package.

Note: To avoid future problems, do not use the result string directly but only the first word. Later versions may return a result string with more parameter values. However, 'DEBUG' or 'NODEBUG' will always be the first word in the result string.

RESETVALUE('WAIT')

Result: rc

The function call resets the defaults for debugging and returns no data.

WAIT(...,'ALL',...)
TEST(...,'ALL',...)

Result: rc name string

The function waits for any event with the default waiting parameters and returns an event.

QUERYVALUE('ALL NAMES')

Result: rc list

The function returns a list of all currently registered event handler names. The list changes as other programs register and de-register their names.

QUERYVALUE('ALL EVENTNAMES')

Result: rc list

The function returns a list of all currently registered event handler names which allow waiting. The list changes as other programs register and de-register their names.

RESETVALUE('ALL')

Result: rc

The function issues a call RESETVALUE(name) to all event handlers which enabled the RESETVALUE function.

Examples

```
QUERYVALUE('All EventNames') == '0 CONS WNG MSG SMSG OMSG MAIL FILE TIME'
```

3.2 The Name IMMCMD for Immediate Command Events

The name 'IMMCMD' is used for the basic event handler provided by REXX/WAIT for immediate command events. Immediate command events are lines entered from the keyboard in CMS for which an IMMCMD is registered. (CMS provides some immediate commands such as HX, HI, HT and RT in the system. They can be replaced by immediate commands through REXX/WAIT.) note: The main purpose of the IMMCMD event handler is to provide the means necessary to overwrite the system immediate commands, and not to handle any kind of console input when a REXX program is running. In most other cases, the CONS event handler is better suited to handle console input.

WAIT(...,'IMMCMD' [**name1** [**name2 ...**]]**,...)**
TEST(...,'IMMCMD' [**name1** [**name2 ...**]]**,...)**

Result: rc 'IMMCMD' [string]

The function call waits for an immediate command interrupt and returns 'IMMCMD' to indicate the event and the complete string as entered on the console. The names of immediate commands or '*' can be entered as 'namei'.

SETVALUE('IMMCMD' [**name1** [**name2 ...**]]**)**

Result: rc olddefaults

The function call sets the waiting defaults for the immediate commands and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.)

SETVALUE('IMMCMD COMMANDNAME' name 'ON|OFF')

Result: rc oldsettings

The function call activates or deactivates an immediate command with the given name and returns the settings for the immediate command before the new value is set. (In a REXX program, the result string without 'rc' can directly be used to reset the settings before termination.)

QUERYVALUE('IMMCMD DEFAULTS')

Result: rc defaults

The function call returns the current waiting defaults for the immediate commands.

QUERYVALUE('IMMCMD COMMANDNAME' name)

Result: rc settings

The function call returns the current settings for the given immediate command.

RESETVALUE('IMMCMD')

Result: rc

The function call resets the waiting defaults, deactivates all immediate commands activated using REXX/WAIT and returns no data.

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10 Error during NUCEXT
11 No more immediate commands allowed

Examples

```
SETVALUE('ImmCmd CommandName HX On') == '0 HX OFF' /* Perhaps */
SETVALUE('ImmCmd HX')                == '0 *'      /* Perhaps */
WAIT('ImmCmd')                       == '0 hx abc' /* if 'hx abc' was entered */
```

3.3 The Name CONS for Console Events

The name 'CONS' is used for the basic event handler provided by REXX/WAIT for console events. Console events are lines entered from the keyboard in CMS. (Keyboard events entered in full-screen applications like XEDIT are not handled by the console event handler.)

WAIT(...,'CONS' [**'READ'**|**'NOREAD'**][**'LINE'**|**'CHAR'**],**...)**
TEST(...,'CONS' [**'READ'**|**'NOREAD'**][**'LINE'**|**'CHAR'**],**...)**

Result: rc 'CONS' [string]

The function call waits for a console interrupt and returns 'CONS' to indicate the event. When 'READ' is specified, or when no further parameter is specified and the default setting is 'READ', it also returns the character string read from the console. Otherwise, it is the responsibility of the REXX program the read the console input. (On the VM platform, only 'LINE' but not 'CHAR' is supported.)

SETVALUE('CONS' [**'READ'**|**'NOREAD'**][**'LINE'**|**'CHAR'**]**)**

Result: rc olddefaults

The function call sets the defaults for the console and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'READ LINE' (On the VM platform, only 'LINE' but not 'CHAR' is supported).

QUERYVALUE('CONS DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the console.

Note: To avoid future problems, do not use the result string directly but only the first two words. Later versions may return a result string with more parameter values. However, 'READ' or 'NOREAD' will always be the first word and 'LINE' or 'CHAR' will be the second word in the result string.

RESETVALUE('CONS')

Result: rc

The function call resets the defaults for the console and returns no data.

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10 Error during line read

Examples

```
SETVALUE('Cons Read') == '0 READ LINE'     /* Perhaps */
WAIT('Cons')          == '0 CONS abc'      /* If 'abc' was entered */
```

3.4 The Names WNG, MSG, SMSG and OMSG for Message Events

The names 'WNG', 'MSG', 'SMSG' and 'OMSG' are used for the basic event handler provided by REXX/WAIT for message type events. CP messages issued by the CP commands WNG, MSG and SMSG and console redirection to IUCV with the CP command SET VMCONIO IUCV and SET CPCONIO IUCV are also handled by these event handlers.

VM allows reception of warnings, messages, special messages and other screen I/O via IUCV from *MSG. (Special messages can also be received via VMCF.) The event handler OMSG accepts any message on this IUCV path not handled by another message event handler. Thus, it can happen that new messages get added to the message queue for OMSG despite the fact that it was never turned on or after it was turned off, because the IUCV path to *MSG may still exist for the event handlers WNG, MSG or SMSG, and the REXX program has used the CP command SET to direct other types of messages to IUCV.

WAIT(...,'WNG',...)
TEST(...,'WNG',...)

Result: rc 'WNG' date time origin string

The function call waits for a warning event and returns 'WNG', the date and time when the warning arrived, the network address and warning text.

WAIT(...,'MSG',...)
TEST(...,'MSG',...)

Result: rc 'MSG' date time origin string

The function call waits for a message event and returns 'MSG', the date and time when the message arrived, the network address and the message text.

WAIT(...,'SMSG',...)
TEST(...,'SMSG',...)

Result: rc 'SMSG' date time origin string

The function call waits for a special message event and returns 'SMSG', the date and time when the special message arrived, the network address and the special message text.

WAIT(...,'OMSG' [**'READ'**|**'NOREAD'**],**...)**
TEST(...,'OMSG' [**'READ'**|**'NOREAD'**],**...)**

Result: rc 'OMSG' [date time msgclass string]

The function call waits for any message event from *MSG and returns 'OMSG'. If 'READ' is specified or is used as the waiting default, also the date and time when the message arrived, the original IUCV message class from *MSG, and the message text (including the origin userid in the first eight bytes) are returned. If 'NOREAD' is used, the message is not removed from the message queue and must be processed by other means.

The network address for 'WNG', 'MSG' and 'SMSG' is in the form 'nodeid(userid):' or 'nodeid():' for RSCS addresses. (So far, only 'MSG' allows nodeids other than your local RSCS nodeid, but for consistency all addresses are reported in the same form.) The form 'nodeid():' with empty userid indicate messages from the local or a foreign RSCS machine. For 'OMSG', the messages received from *MSG via IUCV are presented in unmodified form.

SETVALUE('WNG' 'ON'|'IUCV'|'OFF')

Result: rc oldsettings

The function directs warnings to IUCV or to the screen, and returns the setting active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'OFF'. The keywords 'ON' and 'IUCV' are synonyms.

SETVALUE('MSG' 'ON'|'IUCV'|'OFF')

Result: rc oldsettings

The function directs messages to IUCV or to the screen, and returns the setting active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'OFF'. The keywords 'ON' and 'IUCV' are synonyms.

SETVALUE('SMSG' 'ON'|'IUCV'|'VMCF'|'OFF')

Result: rc oldsettings

The function directs special messages to IUCV, VMCF or disables them, and returns the setting active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'OFF'. The keywords 'ON' and 'IUCV' are synonyms. (Setting SMSG to VMCF is only supported on versions of CMS which allow the use of the

HNDEXT MACRO to handle VMCF external interrupts 4001. Otherwise, a return code reporting VMCF problems is returned.)

SETVALUE('OMSG' ['ON'|'IUCV'|'OFF']['READ'|'NOREAD'])

Result: rc oldsettings

The function directs messages from *MSG to IUCV or lets CP handle them. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'OFF'. The keywords 'ON' and 'IUCV' are synonyms. If 'READ' is specified (the initial default), the WAIT function will remove the message from the queue as a default. Otherwise, the message will not be removed.

Note: Setting a message event handler off does not mean that the event handler will no longer return messages with the WAIT function because there may still be messages pending in the queue. It only means that no new messages get added to the queues except for OMSG. Messages may still get added to the OMSG queue after it got turned off when the IUCV path to *MSG is established and when messages (not handled by another message event handler) get redirected to IUCV. However, if all message event handlers get turned off, no new messages get added to any queue.

Note: Setting special messages to IUCV is the preferred method. Setting them to VMCF should only be used with care because other programs using VMCF will terminate it such that no more special messages get queued.

SETVALUE('OMSG CREATE HIDEQUEUE')

Result: rc queue-id

The function creates a new empty queue for hiding messages and adds it to the top of the stack of hidden message queues.

SETVALUE('OMSG RELEASE HIDEQUEUE' [queue-id])

Result: rc queue-id

The function moves all hidden messages in the queue to the normal message queue and destroys the corresponding hidden message queue. If no queue-id is specified, the hidden message queue created last will be released. If a queue-id is specified, all hidden message queues before and including the hidden message queue with the given id will be released.

SETVALUE('OMSG HIDE' [number])

Result: rc number

The function moves all (or as many as specified in the parameter number) messages from the normal message queue to the hidden message queue which is at the top of the stack of hidden message queues.

QUERYVALUE('WNG DEFAULTS')

Result: rc settings

The function call returns the current settings for handling of warnings.

QUERYVALUE('MSG DEFAULTS')

Result: rc settings

The function call returns the current settings for handling of messages.

QUERYVALUE('SMSG DEFAULTS')

Result: rc settings

The function call returns the current settings for handling of special messages.

QUERYVALUE('OMSG DEFAULTS')

Result: rc settings

The function call returns the current settings for handling of other messages from *MSG.

QUERYVALUE('WNG PENDING')

Result: rc number

The function call returns the number of warnings in the warning queue.

QUERYVALUE('MSG PENDING')

Result: rc number

The function call returns the number of messages in the message queue.

QUERYVALUE('SMSG PENDING')

Result: rc number

The function call returns the number of smsgs in the smsg queue.

QUERYVALUE('OMSG PENDING')

Result: rc number

The function call returns the number of other messages from *MSG in the queue. (Messages moved to hidden queues are not included.)

QUERYVALUE('WNG LOST')

Result: rc number

The function call returns the number of warnings lost due to memory shortage or other reasons.

QUERYVALUE('MSG LOST')

Result: rc number

The function call returns the number of messages lost due to memory shortage or other reasons

QUERYVALUE('SMSG LOST')

Result: rc number

The function call returns the number of special messages lost due to memory shortage or other reasons.

QUERYVALUE('OMSG LOST')

Result: rc number

The function call returns the number of other messages from *MSG lost due to memory shortage or other reasons.

QUERYVALUE('OMSG 'READ'|'PEEK'|'DISCARD' [**omsg-id**]**)**

Result: rc [date time msgclass string]

The function call displays and/or removes the first message (or the message with the given position) in the normal message queue. The message is displayed and removed for 'READ', only displayed for 'PEEK' and only removed for 'DISCARD'.

RESETVALUE('WNG')

Result: rc

The function call resets IUCV, calls 'CP SET WNG ON', clears the queue of pending warnings and returns no data.

RESETVALUE('MSG')

Result: rc

The function call resets IUCV, calls 'CP SET MSG ON', clears the queue of pending messages and returns no data.

RESETVALUE('SMSG')

Result: rc

The function call resets IUCV or VMCF, calls 'CP SET SMSG OFF', clears the queue of pending special messages and returns no data.

RESETVALUE('OMSG')

Result: rc

The function call releases all hidden message queues, resets IUCV, clears the queue of pending other messages from *MSG and returns no data.

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10	HNDIUCV problem
11	CMSIUCV problem
12	VMCF problem
13	Hidden message queue not found
14	No more hidden message queues allowed
15	Message(s) not found

Examples

```
SETVALUE('Msg On') == '0 OFF'              /* Perhaps */
WAIT('Msg')        == '0 MSG 1992/06/03 17:45:15 ZURLVM1(RFH): Regards, Rainer'
```

3.5 The Name MAIL for Reader Events

The name 'MAIL' is used for the basic event handler provided by REXX/WAIT for virtual reader events. Virtual reader events are spool files pending in the virtual reader or arriving there.

WAIT(...,'MAIL' [**'CLASS' classlist**][**'ALL'**|**'NOHOLD'**],...**)**
TEST(...,'MAIL' [**'CLASS' classlist**][**'ALL'**|**'NOHOLD'**],...**)**

Result: rc 'MAIL' mail-id

The function call waits for a virtual reader interrupt if at least one class or a hold setting is specified or if waiting defaults are set for mail, and returns 'MAIL' with the id (spool id) of the mail item.

Note: Waiting for mail leaves the virtual reader as it is and does neither reorder the files in the virtual reader nor open a reader file.

SETVALUE('MAIL' 'ON'|['CLASS' classlist]['ALL'|'NOHOLD']|'OFF')

Result: rc olddefaults

The function call sets the defaults for the reader classes or whether waiting is enabled for all or only not hold mail, or disables mail events, and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) The keyword 'ON' sets the default to 'CLASS * ALL'. Initially, the default is 'OFF'.

SETVALUE('MAIL' mail-id 'HOLD'|'NOHOLD')

Result: rc 'HOLD'|'NOHOLD'

The function call sets the mail item with given id to the hold or nohold status, and returns the old setting for the mail item.

QUERYVALUE('MAIL DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the reader classes and hold setting or 'OFF' when mail events are disabled.

Note: To avoid future problems, do not use the result string directly but only the first two words. Later versions may return a result string with more parameter values. However, 'CLASS' and the class followed by either the keyword 'ALL' or 'NOHOLD', or the keyword 'OFF' will always be the first words in the result string.

QUERYVALUE('MAIL' mail-id)

Result: rc date time origin status class secure xfer [filename [filetype]]

The function call returns the date, time, the network address for RSCS mail in form 'nodeid(userid)', the status ('HOLD' or 'NOHOLD'), the class, the secure address flag ('0' or '1'), the transfer bit ('0' or '1'), and the filename and filetype (if available) of the mail item. (The secure address flag shows whether diagnose 'F8'x has been used for obtaining the mail-address or the cp tag for mail from RSCS. For files not from RSCS, the flag is always true.)

Note: This function closes the virtual reader, reorders the reader files, and reads the SFBLOK of the reader file.

QUERYVALUE('MAIL' mail-id keyword [number])

Result: rc information

The function call returns further information about the mail item depending on the keyword entered. The following keywords are supported:

- 'TAG' without a number returns the number of tag lines found in the spool file and the type of the file. The type can be one of the following keywords:

'Punch' (an unidentified punch file)
'NetData' (a punch file in NETDATA format)
'Profs' (a punch file in PROFS format)
'Print' (a print file)
'Console' (a console file)
'Dump' (a system dump file)
'Unknown' (an unidentified file)

- 'TAG' with a number returns the corresponding tag line without trailing blanks.
- 'NETDATA' without a number returns the number of files, the source address, the destination address, the time stamp and whether an acknowledgement is requested or not ('Ack' or 'NoAck'). Addresses are in the form 'nodeid(userid)', and unknown items are coded as '?'. (All information is taken from the INMR01 record of the NETDATA format.)
- 'NETDATA' with a number returns the type of the corresponding file ('File', 'Note' or 'Ack') and (for files) the dataset name in the form 'part1.part2...partn[(member)]' (with filemode as part1, filename as part2 and filetype as part3 for VM files) and the time stamp when the file was last changed.

Note: This function closes the virtual reader, reorders the reader files, and reads some of the reader file's data blocks. (If a function reads the reader file, it must not be in hold status.)

RESETVALUE('MAIL')

Result: rc

The function call disables mail events and returns no data.

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10	Problem with the virtual reader
11	HNDINT problem
12	Mail item not found
13	Mail item in hold status
14	Not in NETDATA format
15	Invalid number specified
16	Mail item is system held

Examples

```
SETVALUE('MAIL CLASS B A')      == '0 OFF'
WAIT('MAIL CLASS *')            == '0 MAIL 0014'  /* Spool file with two NETDATA files */
QUERYVALUE('MAIL 14')           == '0 1992/11/25 04:45:14 ZURLVM1(RFH) NOHOLD A 0 0 EXTINT52 MODULE'
SETVALUE('MAIL 14 NOHOLD')      == '0 NOHOLD'
QUERYVALUE('MAIL 14 TAG')       == '0 2 NetData'
QUERYVALUE('MAIL 14 TAG 1')     == '0 FILE (7359) ORIGIN ZURLVM1  RFH      11/25/92  4:45:15 PST'
QUERYVALUE('MAIL 14 TAG 2')     == '0 ALMVMA   HAUSER   00 W=HAUSER'
QUERYVALUE('MAIL 14 NETDATA')   == '0 2 ZURLVM1(RFH) ALMVMA(HAUSER) 19921125124515304351 NoAck'
QUERYVALUE('MAIL 14 NETDATA 1') == '0 Note'
QUERYVALUE('MAIL 14 NETDATA 2') == '0 File B1.EXTINT52.MODULE'
```

3.6 The Name FILE for Events from a Time File

The name 'FILE' is used for the basic event handler provided by REXX/WAIT for timer events specified in a time file. The lines in the time file specify the date and time when an event is supposed to happen.

The format of the event file is:

```
Columns   Field  Description                                Example
--------------------------------------------------------------------
 1 -  10  Date   This is the date specification of the event  1991/08/30
12 -  28  Time   This is the time specification of the event  10:41:03
30 -  39  Stamp  Date or time when last executed             1991/08/30
41 - 540  Data   Data line to be returned                    Some text
```

A sample event file is provided in the REXX/WAIT package as file RXWSAMPL TIMEFILE:

```
====/==/== ==:==:00                On exact minutes
====/==/== ==:==:30                On half minutes
EVERYDAY   +00:05:00               After five minute
MONDAY     08:00:00 17:00:00       It is Monday
TUESDAY    08:00:00 17:00:00       It is Tuesday
WEDNESDAY  08:00:00 17:00:00       It is Wednesday
THURSDAY   08:00:00 17:00:00       It is Thursday
FRIDAY     08:00:00 17:00:00       It is Friday
WEEKEND    08:00:00                It is a Weekend
HOLIDAY    10:00:00 14:00:00       It is a Holiday
```

The fields in the event file can be specified as follows:

- Ignored records (column 1):

  ```
  * This is a comment
  ```

  File records with '*' in the first column are ignored as comments. Also file records with '?' or '-' in the first column are ignored. The file event handler marks invalid records with a '?' in the first column, and records that will never be executed again with a '-'.

- Date (columns 1 - 10):

  – Explicit date yyyy/mm/dd:

    ```
    1991/08/01          ==> August 1st, 1991
    ====/08/01          ==> August 1st of every year
    ====/08/==          ==> Every day in August of every year
    2===/==/=2          ==> The 2nd, 12th, and 22nd each month starting in the year 2000
    ====/==/==          ==> Every day of every year
    ```

    Equal signs can appear in any place of the date.

  – Keywords:

    ```
    MONDAY              ==> Every Monday
    2TUESDAY            ==> 2nd Tuesday of every month
    4FRIDAY             ==> 4th Friday of every month
    5FRIDAY             ==> 5th Friday of every month (if existing)
    LSATURDAY           ==> Last Saturday of every month
    WEEKDAY             ==> Every weekday (Monday to Friday)
    WORKDAY             ==> Every working day (Monday to Friday, not a holiday)
    WEEKEND             ==> Every weekend day (Saturday and Sunday)
    HOLIDAY             ==> Holidays (defined in the holiday file)
    EVERYDAY            ==> Every day of every year
    LASTDAY             ==> Last day of every month
    MONTHLY             ==> Once every month
    YEARLY              ==> Once every year
    ```

    The names for a day of the week (`MONDAY,...,SUNDAY`) cannot be abbreviated and must appear in upper case letters. When preceded by a number between 1 and 5, they are restricted to the weekday with this number in that month. When preceded by the letter 'L', they are restricted to the last weekday in that month.

- Time (columns 12 - 28):

  – Explicit time hh:mm:ss [hh:mm:ss]:

```
08:00:00 08:59:59   ==> Once a day earliest at 8:00am, but before 9:00am
09:30:00            ==> Once a day earliest at 9:30am
==:00:00            ==> Every full hour
```

The second time allows to restrict a file event to a certain time interval. When equal signs appear in the time, no second time is allowed to restrict the event, and all equal signs must be on the left side of the first numeric digit. (Therefore, '==:50:==' is invalid.) The hours must be all or none equal signs. (Therefore, '=5:20:13' is invalid.)

– Relative time +hh:mm:ss:

```
+00:10:00           ==> Every 10 minutes
+00:00:19           ==> Every 19 seconds
```

When a file event with relative time can get executed at the same time as a file event with explicit time (with or without equal signs), the relative file event has lower priority. However, it may get executed afterwards, even if the timer expired at an earlier point in time.

• Stamp (columns 30 - 39):

```
1991/07/31          ==> Date of last execution
  17:22:03          ==> Time of last execution
```

File events which can get executed at most once a day (time of the form '17:00:00' [19:59:59]), are stamped with the date, while events which can get executed several times a day (time of the form '==:00:00' or '+00:10:00'), are stamped with the time of the last execution.

• Data (columns 41 - 540):

```
This is an arbitrary line
```

This is returned as the result of the WAIT function when the file event gets executed. Records may not be longer than 540 character leaving 500 characters for the application as arbitrary data.

WAIT(...,'FILE' [**filename** [**filetype** [**filemode**]]],**...)**
TEST(...,'FILE' [**filename** [**filetype** [**filemode**]]],**...)**

Result: rc 'FILE' number string

The function call waits for a file event and returns 'FILE', the record number and the data part of the record found in the file. The default filetype is 'TIMEFILE' and the default filemode is '*' if the filename is specified. If no parts of the name are specified, the default name set with the SETVALUE function will be used.

SETVALUE('FILE' [**filename** [**filetype** [**filemode**]]]**)**

Result: rc olddefaults

The function call sets the defaults for the event file and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is '' meaning no event file.

QUERYVALUE('FILE DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the event file.

RESETVALUE('FILE')

Result: rc

The function call resets the defaults for the event file and returns no data.

Note: Only one file event can be specified within one call of the WAIT function. (In other words, you cannot wait for events from two different files within one call of the WAIT function.) Further, the event file must be on a R/W disk. (Filemode 6, e.g. SOME TIMEFILE A6, is an optimal selection.)

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10	File not found
11	File not on R/W disk
12	File contains invalid records
13	File open error
14	File read error

Examples

```
SETVALUE('File RXWSAMPL')   == '0'                      /* Perhaps */
QUERYVALUE('File Defaults') == '0 RXWSAMPL TIMEFILE *'
WAIT('File')                == '0 FILE 7 It is Thursday'  /* Perhaps */
```

3.7 The Name TIME for Timer Events

The name 'TIME' is used for the basic event handler provided by REXX/WAIT for timer events. Time events can be specified as relative or as absolute time. They happen when the relative amount of time has passed or when the absolute time has been reached.

WAIT(...,'TIME' [**relative**|**absolute**],**...)**
TEST(...,'TIME' [**relative**|**absolute**],**...)**

Result: rc 'TIME' wakeup-date wakeup-time

The function call waits for a time event and returns 'TIME', the wakeup date and the wakeup time. The time event can be be specified in several forms:

- Relative time event:
  - Keyword 'FOREVER':

    ```
    WAIT('TIME FOREVER')        ==> wait for ever
    ```

  - [hHOURS][mMINUTES][s[SECONDS]]:

    ```
    WAIT('TIME 10SEC')          ==> wait for 10 seconds
    WAIT('TIME 5MIN 72SEC 5')   ==> wait for 6 minutes and 17 seconds
    WAIT('TIME 500MSEC')        ==> wait for half a second
    WAIT('TIME 0')              ==> terminates immediately
    ```

    The units can be specified as 'H', 'HRS' and 'HOURS' for hours, as 'M', 'MIN' and 'MINUTES' for minutes, as 'S', 'SEC' and 'SECONDS' for seconds, and as 'MS', 'MSEC', 'MSECONDS' and 'MILLISECONDS' for milliseconds. (The default unit is 'S'.)

  - + [h]h:mm[:ss][.u[u[u]]]:

```
WAIT('TIME +02:31')          ==> wait for 2 hours and 31 minutes
WAIT('TIME +2:31:30')        ==> wait for 2 hours, 31 minutes and 30 seconds
WAIT('TIME +0:00:00.1')      ==> wait for a tenth of a second
WAIT('TIME +0:00:00')        ==> terminates immediately
```

Relative wait time may not exceed 23 hours, 59 minutes and 59 seconds.

- Absolute time event:
  - [h]h:mm[:ss]:

    ```
    WAIT('TIME 16:35:40')        ==> wait until time exactly reached
    WAIT('TIME ==:00:00')        ==> wait until next full hour exactly reached
    WAIT('TIME ==:==:00')        ==> wait until next full minute exactly reached
    WAIT('TIME ==:==:=5')        ==> wait until the last digit of time is 5
    WAIT('TIME ==:==:==')        ==> terminates immediately
    ```

    When equal signs appear in the time, they must be on the left side of the first numeric digit. (Therefore, '==:50:==' is invalid.) The hours must be all or none equal signs. (Therefore, '=5:20:13' is invalid.)

  - > [h]h:mm[:ss]:

    ```
    WAIT('TIME >16:35:40')       ==> terminates immediately when time already passed
    WAIT('TIME >00:00:00')       ==> terminates immediately
    ```

  - < [h]h:mm[:ss]:

    ```
    WAIT('TIME <16:35:40')       ==> terminates immediately when time not yet passed
    WAIT('TIME <23:59:59')       ==> terminates immediately
    ```

SETVALUE('TIME' relative|absolute)

Result: rc olddefaults

The function call sets the defaults for the timer and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is 'FOREVER' meaning wait for ever.

Absolute and relative time can be specified as for the WAIT function. (The value specified gets converted into a canonical form by the SETVALUE function.)

QUERYVALUE('TIME DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the timer.

RESETVALUE('TIME')

Result: rc

The function call resets the defaults for the timer and returns no data.

Note: When the time 'FOREVER' is specified, the wait lasts for ever and no timer event will ever be returned. Therefore, never specify `WAIT('TIME')` without setting a finite default time when the wait is expected to terminate.

Examples

```
SETVALUE('Time 13:25:07') == '0 FOREVER'   /* Perhaps */
WAIT('Time')              == '0 TIME 1992/06/03 13:25:07'
WAIT('Time 5Sec')         == '0 TIME 1992/06/03 13:25:12'
```

3.8 The Name HOLIDAY for the Holiday Support

The name 'HOLIDAY' is used for the basic event handler provided by REXX/WAIT for its holiday support. The lines in the holiday file specify the date and name of the holidays.

The format of the holiday file is:

```
Columns   Field  Description                                  Example
---------------------------------------------------------------------
 1 -  10  Date   This is the date of the holiday              1993/12/25
12 -  50  Name   This is the name of the holiday              Christmas
```

A sample holiday file is provided in the REXX/WAIT package as file RXWSAMPL HOLIDAYS:

```
* Holidays independent of the year:
====/01/01 New Year's Day
====/05/01 Labor Day
====/12/25 Christmas
====/12/26 Boxing Day
* Other holidays for 1993:
1993/04/09 Good Friday
1993/04/11 Easter
1993/04/12 Easter Monday
1993/05/20 Ascension Day
1993/05/30 Whitsunday
1993/05/31 Whit Monday
```

The fields in the holiday file can be specified as follows:

- Ignored records (column 1):

  ```
  * This is a comment
  ```

  File records with '*' in the first column are ignored as comments. Also any other records which do not contain a valid date in the first ten columns are ignored.

- Date (columns 1 - 10):

  ```
  ====/12/25         ==> Christmas is every year December 25th
  ====-12-25         ==> Christmas is every year December 25th (ISO)
  1993/04/09         ==> Good Friday in 1993 was April 9th
  1993-04-09         ==> Good Friday in 1993 was April 9th (ISO)
  ```

  Equal signs can appear only in the year, and either all year digits must be numerical or equal signs.

- Name (columns 12 - 50):

  ```
  Christmas
  Good Friday
  ```

  The name of the holiday is returned by the QUERYVALUE function.

SETVALUE('HOLIDAY' [**filename** [**filetype** [**filemode**]]]**)**

Result: rc olddefaults

The function call sets the defaults for the holiday file and returns the default settings active before the new values are set. (In a REXX program, the result string without 'rc' can directly be used to reset the defaults before termination.) Initially, the default is '' meaning no holiday file.

Note: The holiday support is only available on VM/XA and VM/ESA releases. On older systems, the function reports a file I/O problem.

QUERYVALUE('HOLIDAY DEFAULTS')

Result: rc defaults

The function call returns the current default settings for the holiday file.

QUERYVALUE('HOLIDAY NAME' [**date**]**)**

Result: rc date [name]

The function call returns the date, and if the date corresponds to a holiday the name of the holiday. (If no name has been specified in the holiday file, the default name '?' will be used.)

RESETVALUE('HOLIDAY')

Result: rc

The function call resets the defaults for the holiday file and returns no data.

Additional Return Codes: In addition to the common values defined by REXX/WAIT, the functions return the following return codes:

10 File I/O problem

Examples

```
SETVALUE('HoliDay RXWSAMPL')         == '0'             /* Perhaps */
QUERYVALUE('HoliDay Defaults')       == '0 RXWSAMPL HOLIDAYS *'
QUERYVALUE('HoliDay Name')           == '0 1993/06/08'   /* Today */
QUERYVALUE('HoliDay Name 1993/06/03') == '0 1993/06/03'
QUERYVALUE('HoliDay Name 1993/04/11') == '0 1993/04/11 Easter'
```

4.0 REXX Sample Program

This chapter discusses the RXWSAMPL EXEC which is a REXX sample program showing the use of the functions provided by REXX/WAIT.

4.1 The REXX-EXEC RXWSAMPL

The RXWSAMPL program uses the RXWSAMPL TIMEFILE (by copying it to your 191 disk if available) and the default time '==:==:00', and it reports any events happening on the screen:

```
/*- RXWSAMPL -- Example demonstrating the usage of REXX/WAIT ----------*/
trace o

/* Set error code values                                              */
ecpref = 'RXW'
ecname = 'SMP'

parse arg argstring
argstring = strip(argstring)
if substr(argstring,1,1) = '?' then do
  say 'The  "RXWSAMPL" command  demonstrates the use of REXXWAIT.  It'
  say 'waits for interrupts, shows them on the screen, and terminates'
  say 'when "exit" is entered on the keyboard.                       '
  say 'Syntax:                                                       '
  say '    RXWSAMPL                                                  '
  exit 100
end

/* Load REXXWAIT and EXTINT52 if not yet loaded                       */
address command 'REXXWAIT LOAD'
if rc¬=0 then call error 'E', 200, 'Unable to load REXXWAIT MODULE'
say 'REXXWAIT Version:' QueryValue('WAIT VERSION')
address command 'EXTINT52 LOAD'

/* Initialize variables and event handlers and save old settings      */
savesettings. = ''
templist = QueryValue('ALL NAMES')
if wrc=0 then do
  do while templist¬=''
    parse var templist tempname templist
    select
      when tempname='WAIT' then do
        tr = trace()
        if tr='A' | tr='R' | tr='I' then debugopt = 'DEBUG'
        else debugopt = 'NODEBUG'
        savesettings.tempname = SetValue('WAIT' debugopt)
      end
      when tempname='CONS' then do
        savesettings.tempname = SetValue('CONS READ')
      end
      when tempname='WNG' then do
        savesettings.tempname = SetValue('WNG OFF')
      end
      when tempname='MSG' then do
        savesettings.tempname = SetValue('MSG OFF')
      end
      when tempname='SMSG' then do
        savesettings.tempname = SetValue('SMSG IUCV')
        if wrc¬=0 then savesettings.tempname = SetValue('SMSG VMCF')
        if wrc=0 then address command 'CP SMSG * This is a sample SMSG!'
        else call error 'W', 32, 'Cannot set SMSG to VMCF; rc='wrc
      end
      when tempname='OMSG' then do
        savesettings.tempname = SetValue('OMSG OFF')
      end
      when tempname='MAIL' then do
        savesettings.tempname = SetValue('MAIL OFF')
      end
      when tempname='TIME' then do
        savesettings.tempname = SetValue('TIME ==:==:00')
      end
      when tempname='FILE' then do
        timefile = 'RXWSAMPL CMSUT1 A6'
        address command 'ESTATEW' timefile
        if rc¬=0 then do
          address command 'COPYFILE RXWSAMPL TIMEFILE *' timefile
          if rc=0 then call error 'I', 0, 'Sample time file' timefile,
                                          'created; can be erased'
          else call error 'W', 28, 'Cannot create time file' timefile,
                                   'on a R/W disk'
        end
        address command 'ESTATEW' timefile
        if rc=0 then savesettings.tempname = SetValue('FILE' timefile)
      end
      when tempname='HOLIDAY' then do
        savesettings.tempname = SetValue('HOLIDAY RXWSAMPL HOLIDAYS *')
        parse value QueryValue('HOLIDAY NAME') with hdate hname
        if wrc=0 & hname¬='' then say 'Today is the holiday' hname
      end
      otherwise call ResetValue tempname
    end
  end
end
else call ResetValue 'ALL'

/* Show interrupts until "exit" is entered                              */
say 'To terminate the program, enter "exit".'
say 'Event handler names:' QueryValue('ALL NAMES')
waitrc = 0
do forever
  event = Wait()
  waitrc = wrc
  say time()':' event
  parse upper var event type keyword .
  if wrc¬=0 | (type='CONS' & keyword='EXIT') then leave
end

/* Terminate event handlers and exit                                    */
templist = QueryValue('ALL NAMES')
if wrc=0 then do
  do while templist¬=''
    parse var templist tempname templist
    call SetValue tempname savesettings.tempname
  end
end
else call ResetValue 'ALL'
exit waitrc

/* Calling the real WAIT function                                       */
Wait: procedure expose wrc
  a0 = arg(1)
  a1 = arg(2)
  a2 = arg(3)
  a3 = arg(4)
  a4 = arg(5)
  a5 = arg(6)
  a6 = arg(7)
  a7 = arg(8)
  a8 = arg(9)
  a9 = arg(10)
  parse value 'WAIT'(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9) with wrc res
return res

/* Calling the real SETVALUE function                                 */
SetValue: procedure expose wrc
  a0 = arg(1)
  parse value 'SETVALUE'(a0) with wrc res
return res

/* Calling the real QUERYVALUE function                               */
QueryValue: procedure expose wrc
  a0 = arg(1)
  parse value 'QUERYVALUE'(a0) with wrc res
return res

/* Calling the real RESETVALUE function                               */
ResetValue: procedure expose wrc
  a0 = arg(1)
  parse value 'RESETVALUE'(a0) with wrc res
return res

/* Error message and exit routine                                     */
error: procedure expose ecpref ecname
  type = arg(1)
  retc = arg(2)
  text = arg(3)
  ecretc = right(retc,3,'0')
  ectype = translate(type)
  ecfull = ecpref || ecname || ecretc || ectype
  address command 'EXECIO 1 EMSG (CASE M STRING' ecfull text
  if type¬='E' then return
exit retc
```

Note: The RXWSAMPL program sets only SMSG to IUCV (or to VMCF when setting to IUCV resulted in a non-zero return code) but neither WNG nor MSG to preserve your warnings and messages. However, if WNG and/or MSG are received via the IUCV path to the CP service *MSG, the OMSG event handler will report them. (This can happen when you issued `CP SET WNG IUCV` and/or similarly for MSG without an IUCV path to *MSG.) The program also sets MAIL events off to avoid busy loops because reader files are not processed.

The RXWSAMPL program tries to load the Assembler sample program EXTINT52 which is described in the next chapters. It reports external interrupts 0052 as shown in the following sample session.

4.2 Sample Session with RXWSAMPL

Executing RXWSAMPL should result in console output similar to:

```
rxwsampl
REXXWAIT Version: REXX/WAIT 2.09 18 June 1993
RXWSMP000I Sample time file RXWSAMPL CMSUT1 A6 created; can be erased
To terminate the program, enter "exit".
Event handler names: WAIT CONS WNG MSG SMSG OMSG MAIL FILE TIME HOLIDAY EXT0052
09:09:13: SMSG 1993/06/18 09:09:12 ZURLVM1(RFH): This is a sample SMSG!
09:09:13: FILE 9 It is Friday
09:09:30: FILE 3 On half minutes
```

```
09:10:00: FILE 2 On exact minutes
09:10:00: TIME 1993/06/18 09:10:00
I type a line on the keyboard
09:10:14: CONS I type a line on the keyboard
09:10:30: FILE 3 On half minutes
#CP SMSG * THIS IS ANOTHER SPECIAL MESSAGE
09:10:33: SMSG 1993/06/18 09:10:33 ZURLVM1(RFH): THIS IS ANOTHER SPECIAL MESSAGE
09:11:00: FILE 2 On exact minutes
09:11:00: TIME 1993/06/18 09:11:00
#CP EXT 52
09:11:04: EXT0052 External Interrupt Type 0052
#CP SMSG RSCS (ST.ABCDEF) CPQ USER RFH
09:11:28: SMSG 1993/06/18 09:11:28 ZURLVM1(RSCS): RSCS 0172 0001 ZURLVM1  ABCDEF M1L CPQ: RFH      -L04B9
09:11:28: SMSG 1993/06/18 09:11:28 ZURLVM1(RSCS): RSCS 0001 0002 ZURLVM1  ABCDEF M1L End of command response
09:11:30: FILE 3 On half minutes
exit
09:11:38: CONS exit
Ready;
```

The first SMSG was issued by the RXWSAMPL program and the second from the keyboard. The sample Assembler program EXTINT52 gets started, and it wakes up once because `#CP EXT 52` is entered from the keyboard.

5.0 Assembler Interfaces

This chapter describes the internal interfaces of REXX/WAIT as used by Assembler programmers writing additional event handlers. Name registration and exit routines are discussed.

5.1 Name Registration

The first step in a program which uses the Assembler interfaces of REXX/WAIT is registration of a name, and the last step is de-registration. In order to do that, REXXWAIT is called with a parameter `RXWCMD` 'SET' or 'CLR', respectively. As a third value recognized, `RXWCMD` can also be set to 'MOD'. This value is used to modify the six full-words from `RXWWTECB` to `RXWQRYV` or the flags `RXWFLAGS` later in time.

In order to register a name, a program calls SVC 202 directly:

```
         LA    R1,RXWPLIST             Get PLIST
         SVC   202                     Execute it
         DC    AL4(1)                  For errors fall through
```

or the CMSCALL macro:

```
         CMSCALL PLIST=RXWPLIST,CALLTYP=PROGRAM,...
```

with the parameter list:

```
         DS    0F                      Alignment
RXWPLIST DC    CL8'REXXWAIT'           PLIST for REXX/WAIT
RXWCMD   DC    CL4'SET'                Command
RXWFLAGS DC    XL1'00'                 Flags for REXX/WAIT
         DC    XL3'000000'             Special fence
RXWNAME  DC    CL8'ABCD'               Registered program name
RXWWTECB DC    F'-1'                   ECB address
RXWWAIT  DC    F'-1'                   BALR address for WAIT
RXWWAITE DC    F'-1'                   BALR address for WAITEND
RXWSETV  DC    F'-1'                   BALR address for SETVALUE
RXWQRYV  DC    F'-1'                   BALR address for QUERYVALUE
RXWRSTV  DC    F'-1'                   BALR address for RESETVALUE
         DC    F'-1'                   Not used
         DC    F'-1'                   Not used
```

This sample code registers the name 'ABCD'. The two values `RXWWTECB` and `RXWWAIT` should either both be set to F'-1' or contain an address. `RXWWTECB` is the address of the wait ECB (in OS format), and `RXWWAIT` the address of the exit routine for the WAIT function. The value `RXWWAITE` should also either be set to F'-1' or contain the address of an exit routine. A value other than F'-1' has only an effect if also the wait ECB and the exit routine for the WAIT function are not set to F'-1'. (Its purpose is described below.) Similarly, the values `RXWSETV`, `RXWQRYV` and `RXWRSTV` are either set to F'-1' or contain the address of the exit routine for the SETVALUE, QUERYVALUE or RESETVALUE function, respectively. The last two full-words in the parameter list (marked as `Not used`) are reserved for future use, and should be set to F'-1' to avoid possible problems in the future. In general, the value F'-1' means that there is no exit routine for this function.

The flags in `RXWFLAGS` have the following semantics:

```
KEEPBLNK EQU   X'04'                   Keep blanks in the arguments
KEEPCASE EQU   X'02'                   Keep the case of the arguments
MULTCALL EQU   X'01'                   Allow multiple calls in WAIT
```

When `KEEPBLNK` is set, REXX/WAIT does not automatically eliminate leading and trailing blanks in the argument string passed to the event handler program. When `KEEPCASE` is set, REXX/WAIT does not automatically translate the argument string passed to the event handler program to upper case characters. When `MULTCALL` is set, the event handler program can handle multiple calls in one call of the WAIT function.

Note: When REXX/WAIT is not yet loaded, a call of the RXWCMD 'SET' loads it in a first step. The module does not fit into the transient area, but it is linked relocatably and can therefore also be called when in CMS SUBSET.

A call of REXX/WAIT with 'SET', 'MOD', or 'CLR' terminates with one of the following return codes if the REXXWAIT MODULE exists:

0	Normal completion
4	Invalid parameters
8	Not enough free storage
16	Invalid parameters for subcommand SET, MOD and CLR
20	No more names allowed for SET

5.2 Exit Routines

To give an idea of exit routines in general, an example is presented first. It is called via the REXX function QUERYVALUE. Therefore, the address QRYVERS is copied into RXWQRYV field of the RXWPLIST above. It compares the arguments with the string 'VERSION' and returns a version description if the arguments are correct. Otherwise, it returns with rc=7 and a result length of zero bytes. (For simplicity, the program does not check the length of the arguments as provided in R0. Therefore, it would also return the version description for the arguments 'VERSIONSTRING' and so on.)

```
QRYVERS  EQU   *                      REXX/WAIT exit for QUERYVALUE
         USING QRYVERS,R12            Addressability
         LA    R15,7                  Return code = 7
         LA    R0,0                   Length of result string (error)
         CLC   0(7,R1),=C'VERSION'    Is it version ?
         BNER  R14                    Return to REXX/WAIT (error rc=7)
         LA    R0,L'VERSION           Length of result string
         LA    R1,VERSION             Address of result string
         LA    R15,0                  Return code = 0
         BR    R14                    Return to REXX/WAIT (rc=0)
         DROP  R12                    No longer needed
VERSION  DC    C'This is the version text to be returned'
```

REXX/WAIT eliminates the event handler name from the arguments as entered into the WAIT, SETVALUE and QUERYVALUE as well as leading and trailing blanks (if KEEPBLNK is not set), translates the rest to uppercase characters (if KEEPCASE is not set), and presents it to the exit routine. (The RESETVALUE function does not allow any arguments.) It passes the return string from the exit routines (together with the return code and possibly a blank) directly and unchanged to the REXX program except for the WAIT function where it also adds the name of the responding event handler.

5.3 Register Conventions

Passing control from REXX/WAIT to an exit routine and back to REXX/WAIT must follow the register conventions as described in the following:

- When control is passed to the exit routine:

R0	length of arguments
R1	pointer to arguments
R2	call sequence flag (in WAIT function only)
R12	address of exit routine (base register)
R13	save area for general purpose registers
R14	return address
R15	address of exit routine (same as R12)

- When the exit routine passes control back:

R0	length of result string

R1 pointer to result string (ignored when R0 = 0)
R15 return code

When the wait exit routine is called, register R2 is either 0 or 1 depending on the call sequence. The value 0 means that the exit routine is called the first time with this arguments, and the value 1 means that the exit routine has already been called with this arguments before. In other words, the flag passed in R2 can be used to determine whether the arguments have to be tested for correctness.

Other than 0 (and 1 in the wait exit routine), a program can pass back with register R15 return codes defined by REXX/WAIT as appropriate. For example, the return code 6 can be used when a request cannot be satisfied because of memory shortage. A return code 7 can be returned when the arguments are invalid. The return code 9 can be used for unspecified errors or in situations which should not occur. An event handler can also define its own return codes between 10 and 9999 according to its needs.

5.4 Call Sequence

Calls of the REXX functions SETVALUE, QUERYVALUE and RESETVALUE are related to one event handler program only. One call of a REXX function results in exactly one call of an exit routine when all parameters are correct. The only exception is `RESETVALUE('ALL')` which results in a call to all event handlers allowing calls of the RESETVALUE function.

However, one single call of the REXX function WAIT can result in multiple calls of one or multiple exit routines. For example, the call

```
WAIT('ABCD A','ABCD B')
```

calls the exit routine specified in `RXWWAIT` for the registered name 'ABCD' first with argument 'A'. (If the event handler 'ABCD' has not set the `MULTCALL` flag, this call would result in a return code 3, because there are two calls of the same event handler specified. Further, since the exit routine for 'ABCD' is called the first time in the first argument of WAIT, register R2 contains 0.) If the exit routine returns a return code 1 (indicating correct arguments but nothing pending), the exit routine is called again with argument 'B'. (Register R2 still contains 0 since the exit routine for 'ABCD' is called the first time as the second argument of WAIT.) If also this call ends with a return code 1, REXX/WAIT waits with the wait ECB provided by 'ABCD' in `RXWWTECB`. Assuming an event happens satisfying arguments 'B' but not 'A', the exit routine is called a second time with argument 'A' and again ends with a return code 1. (This time, register R2 contains a 1 to indicate that the exit routine for 'ABCD' has already been called as the first argument.) The exit routine is called a last time with argument 'B' and ends this time with a return code 0 because the event satisfying 'B' has happened. (Also in this call, register R2 contains the value 1.)

Therefore, for one call of the REXX function WAIT, an exit routine may get called several times. If an exit routine initializes when called the first time, it cannot clean up afterwards, because it may not get called again when another event happens. For this purpose, the wait-end exit routine specified in `RXWWAITE` has been introduced. When a wait exit routine gets called at least once during the call of the REXX function WAIT, the corresponding wait-end exit routine gets also called exactly once before the WAIT passes control back to REXX.

5.5 Notes for Programmers

1. Event handler names are character strings with up to eight characters from 'A...Z0...9-/'. Any other characters are not allowed. Especially, lowercase characters are not supported. (Names entered to the REXX functions are always translated to uppercase characters.)
2. The current version of REXX/WAIT allows a total of 50 registered names of which 20 can enable for the WAIT function.
3. Result strings returned by any of the exit routines must not be longer than 1000 bytes.
4. Exit routines should only produce return codes between 0 and 9999. Return codes below 10 should only be used according to their meaning. Return codes larger or equal to 10 can be defined independent

of the use of the same values by other event handlers. (The return code 1 in the WAIT function has the special meaning that no event is pending.)

5. All interrupts are disabled when control is passed to the exit routines. Do not enable any interrupts in the wait-end exit routine and enable interrupts in the wait exit routine with care to avoid unpredictable results in other event handler programs.
6. It is the responsibility of the wait exit routine to clear the ECB when appropriate. (Otherwise, the wait may result in a busy loop.) It is also the responsibility of the event handler program and its interrupt handler to post the ECB. REXX/WAIT does not modify any ECB not owned by itself.
7. All wait exit routines must handle an empty argument string because of calls with the special keyword 'ALL'. The routine may always set a return code of 1 in this case, but an empty argument string must be legal. (If a wait exit routine terminates with a return code other than 0 or 1 when no arguments are passed to it, the REXX function WAIT terminates whenever called with no arguments or the keyword 'ALL'.)
8. If a program allows the SETVALUE function, it may as well support the RESETVALUE function. Otherwise, the call `RESETVALUE('ALL')` would not allow resetting all default values with one REXX function call. (By the way, this function call is implicitly issued when REXX/WAIT is dropped from nucleus. This allows to reset all functions when NUCXDROP is called for the REXX/WAIT functions.)

6.0 Assembler Sample Program

This chapter discusses EXTINT52 ASSEMBLE which is an Assembler sample program showing the use of the Assembler interfaces provided by REXX/WAIT.

6.1 The EXTINT52 Program

The EXTINT52 program wakes up on external interrupts of type '0052' produced by the CP command 'EXTERNAL 52'. The source code (with line numbers for ease of reference) looks as follows:

```
 1 ***********************************************************************
 2 *  EXTINT52:                                                          *
 3 *                                                                     *
 4 *  EXTINT52 is a sample program showing the internal interfaces of    *
 5 *  REXXWAIT and how they are intended to be used.                     *
 6 *                                                                     *
 7 *  Parameters:                                                        *
 8 *     EXTINT52 ?         to tell about the program                    *
 9 *     EXTINT52           nucxloads EXTINT52 only                      *
10 *     EXTINT52 LOAD      nucxloads EXTINT52 and activates it          *
11 *     EXTINT52 RESET     deactivates EXTINT52                         *
12 *     EXTINT52 TEST      test whether nucleus loaded or not           *
13 *     NUCXDROP EXTINT52  deactivates EXTINT52 and unloads it          *
14 ***********************************************************************
15 *  Command sequence to generate a module:                             *
16 *     1. ASSEMBLE EXTINT52                                            *
17 *     2. LOAD EXTINT52 (ORIGIN TRANS                                  *
18 *     3. GENMOD (SYSTEM                                               *
19 ***********************************************************************
20 *  Return codes from EXTINT52:                                        *
21 *     0  - Execution OK                                               *
22 *     1  - Not nucleus loaded (parameter test)                        *
23 *     4  - Invalid parameters                                         *
24 *     8  - Not enough free storage and other initialization errors    *
25 ***********************************************************************
26 *  (C) COPYRIGHT IBM CORPORATION 1993                                 *
27 ***********************************************************************
28 *  Author: Rainer F. Hauser, Zurich, Switzerland (RFH at ZURLVM1)     *
29 *  Date: August, 1991                                                 *
30 ***********************************************************************
31 EXTINT52 START X'E000'                 It is for the transient area
32          USING *,R12                   Addressability
33          USING NUCON,0                 Low storage
34          LR    R10,R1                  Save R1
35          B     START                   Jump over copyright data
36          DC    C'EXTINT52 (C) COPYRIGHT IBM CORPORATION 1993'
37          DS    0H                      Alignment
38 START    LA    R1,0(R1)                To get rid of some flags
39          LA    R2,0                    Just a flag for USLOAD
40          CLI   8(R1),X'FF'             No args ?
41          BE    USLOAD                  Yes
42          LA    R2,1                    Just a flag for USLOAD
43          CLC   8(8,R1),=CL8'LOAD'      Is it a LOAD ?
44          BE    USLOAD                  Yes
45          CLC   8(8,R1),=CL8'RESET'     Is it a RESET ?
46          BE    USRESET                 Yes
47          CLC   8(8,R1),=CL8'TEST'      Is it a TEST ?
48          BE    USTEST                  Yes
49          LA    R2,TELL                 To be addressable in TELL
50          CLC   8(8,R1),=CL8'?'         Is it a Query ?
```

```
         BE    TELL                    Yes
         LA    R15,4                   Return code = 4
         BR    R14                     Return to caller
***********************************************************************
* User space LOAD loads EXTINT52 into system space                    *
***********************************************************************
USLOAD   LA    R15,0                   Return code = 0
         TM    E52FLAGS,LOADED         Program already loaded ?
         BNZR  R14                     Return to caller
         MVI   E52FLAGS,X'00'          Reset all flags
         NUCEXT QUERY,NAME=NLNAME,ERROR=* Tests for nucxloaded programs
         LTR   R15,R15                 Already nucxloaded ?
         BZR   R14                     Return to caller
         LA    R8,NUCXB                R8 -> nucxloaded code
         L     R9,=A(NUCXE-NUCXB)      R9 = length of nucxloaded code
         LA    R0,7(R9)                R0 = length of free ...
         SRL   R0,3                    ... storage code in dwords
         LR    R5,R0                   Save for later use
         ENABLE INTTYPE=NONE           Disable interrupts
         CMSSTOR OBTAIN,DWORDS=(0),SUBPOOL='NUCLEUS',MSG=NO,LOC=BELOW, +
               ERROR=USLOADE           Get storage in system space
         ENABLE INTTYPE=ALL            Enable interrupts
         LR    R6,R1                   R6 -> free storage
         LR    R7,R9                   R7 = length of free storage code
         LR    R3,R1                   Save address for later use
         LR    R4,R9                   Save length for later use
         MVCL  R6,R8                   Move the code to free storage
         NUCEXT SET,NAME=NLNAME,ENTRY=(3),AMODE=24,INTTYPE=ALL,        +
               ORIGIN=((3),(4)),KEY=NUCLEUS,SYSTEM=YES,SERVICE=YES,    +
               ERROR=USLOADEM          Nucxload EXTINT52
         LTR   R2,R2                   Do we leave here ?
         BZR   R14                     Yes
         LR    R1,R10                  Restore R1
         LR    R15,R3                  Simulate a BALR
         LR    R12,R3                  Install the base register
         BR    R3                      Jump into nucxloaded version
USLOADEM ENABLE INTTYPE=NONE           Disable interrupts
         CMSSTOR RELEASE,DWORDS=(5),ADDR=(3),SUBPOOL='NUCLEUS',MSG=YES,+
               ERROR=USLOADE           Return no longer used memory
         ENABLE INTTYPE=ALL            Enable interrupts
USLOADE  LA    R15,8                   Return code = 8
         BR    R14                     Return to caller
***********************************************************************
* User space RESET does nothing                                       *
***********************************************************************
USRESET  LA    R15,0                   Return code = 0
         BR    R14                     Return to caller
***********************************************************************
* User space TEST gives return code 1                                 *
***********************************************************************
USTEST   LA    R15,1                   Return code = 1
         BR    R14                     Return to caller
         DROP  R12                     No longer needed
***********************************************************************
* The following code resides in system storage, and is capable of     *
* replying to the same parameters as the user space version does.     *
***********************************************************************
         DS    0D                      Alignment of nucxloaded code
NUCXB    EQU   *                       The nucxloaded version
         USING *,R12                   Get addressability
         USING NUCON,0                 Nucleus
         LA    R15,0                   Return code = 0
         CLI   8(R1),X'FF'             No args ?
         BER   R14                     Yes
         CLC   8(8,R1),=CL8'LOAD'      Is it a LOAD ?
         BE    SYLOAD                  Yes
         CLC   8(8,R1),=CL8'RESET'     Is it a RESET ?
         BE    SYRESET                 Yes
         CLC   8(8,R1),=CL8'TEST'      Is it a TEST ?
         BE    SYTEST                  Yes
         LA    R2,TELL                 To be addressable in TELL
         CLC   8(8,R1),=CL8'?'         Is it a Query ?
         BE    TELL                    Yes
         LA    R15,4                   Return code = 4
         BR    R14                     Return to caller
***********************************************************************
* System space LOAD initializes EXTINT52                              *
***********************************************************************
SYLOAD   LA    R15,0                   Return code = 0
         TM    E52FLAGS,LOADED         Program already loaded ?
         BNZR  R14                     Yes
         LA    R2,HANDLER              R2 -> interrupt handler
         HNDEXT SET,(R2),CODE=0052,KEEP=YES,SYSTEM=YES,ERROR=SYLOADE
         OI    E52FLAGS,LOADED         Mark that program is loaded
         MVC   RXWCMD(4),=CL4'SET'     REXXWAIT command
         LA    R2,E52ECB               Address of ECB
         ST    R2,RXWWTECB             Save it into parameter list
         LA    R2,RXWAIT               Wait routine
         ST    R2,RXWWAIT              Save it into parameter list
         LA    R2,RXWAITE              Wait end routine
         ST    R2,RXWWAITE             Save it into parameter list
         LA    R2,RXQRYV               Queryvalue routine
         ST    R2,RXWQRYV              Save it into parameter list
         CMSCALL PLIST=RXWPLIST,CALLTYP=PROGRAM,COPY=NO,ERROR=*
         LA    R15,0                   Return code = 0
         BR    R14                     Return to caller
SYLOADE  LA    R15,8                   Return code = 8
         BR    R14                     Return to caller
***********************************************************************
* System space RESET terminates EXTINT52                              *
***********************************************************************
SYRESET  LA    R15,0                   Return code = 0
         TM    E52FLAGS,LOADED         Program loaded ?
         BZR   R14                     No
         HNDEXT CLR,CODE=0052,ERROR=*  Interrupts are no longer trapped
         NI    E52FLAGS,255-LOADED     Clear program loaded flag
         MVC   RXWCMD(4),=CL4'CLR'     REXXWAIT command
         CMSCALL PLIST=RXWPLIST,CALLTYP=PROGRAM,COPY=NO,ERROR=*
         BR    R14                     Return to caller
***********************************************************************
* System space TEST gives return code 0                               *
***********************************************************************
SYTEST   LA    R15,0                   Return code = 0
         BR    R14                     Return to caller
         DROP  R12                     No longer needed
***********************************************************************
* User space and system space TELL                                    *
***********************************************************************
TELL     EQU   *                       Tell the user about EXTINT52
         USING *,R12                   Get addressability
         LA    R12,0(R2)               Get addressability
         LA    R2,TELLMSG              R2 -> message
         LA    R3,L'TELLMSG            R3 = length
         LINEWRT DATA=((R2),(R3)),ERROR=* Write message on screen
         LA    R15,100                 Return code = 100
         BR    R14                     Return to caller
         DROP  R12                     No longer needed
***********************************************************************
* Interrupt handler for external interrupt 0052                       *
***********************************************************************
HANDLER  EQU   *                       External interrupt 0052 handler
         USING *,R12                   Addressability
         TM    E52FLAGS,WAITING        Is program waiting ?
         BZR   R14                     No (do not modify ECB)
         OI    E52ECB,X'40'            Post ECB by hand
         BR    R14                     Return to caller
         DROP  R12                     No longer needed
***********************************************************************
* REXXWAIT WAIT routine                                               *
***********************************************************************
RXWAIT   EQU   *                       REXXWAIT WAIT routine
         USING *,R12                   Addressability
         LTR   R0,R0                   Parameters ?
         BNZ   RXWAIT7                 Yes
         OI    E52FLAGS,WAITING        Mark that program is waiting
         TM    E52ECB,X'40'            Completed bit set ?
         BNZ   RXWAIT0                 Yes
         B     RXWAIT1                 Return with rc=1
RXWAIT0  LA    R15,0                   Return code = 0
         LA    R0,L'WAITMSG            Length of result
         LA    R1,WAITMSG              Address of result
         BR    R14                     Return to REXXWAIT
RXWAIT1  LA    R15,1                   Return code = 1
         LA    R0,0                    Length of result
         LA    R1,0                    Address of result
         BR    R14                     Return to REXXWAIT
RXWAIT7  LA    R0,0                    Length of result
         LA    R1,0                    Address of result
         LA    R15,7                   Return code = 7
         BR    R14                     Return to REXXWAIT
         DROP  R12                     No longer needed
***********************************************************************
* REXXWAIT WAIT end routine                                           *
***********************************************************************
RXWAITE  EQU   *                       REXXWAIT WAIT end routine
         USING *,R12                   Addressability
         NI    E52FLAGS,255-WAITING    Clear program waiting flag
         LA    R2,0                    R2 = 0
         ST    R2,E52ECB               Clear ECB
RXWAITE0 LA    R15,0                   Return code = 0
         LA    R0,0                    Length of result
         LA    R1,0                    Address of result
         BR    R14                     Return to REXXWAIT
         DROP  R12                     No longer needed
***********************************************************************
* REXXWAIT QUERYVALUE routine                                         *
***********************************************************************
RXQRYV   EQU   *                       REXXWAIT QUERYVALUE routine
         USING *,R12                   Addressability
         CLC   0(L'KWDVER,R1),KWDVER   Version keyword ?
         BNE   RXQRYV7                 No
         LA    R2,L'KWDVER             Length of version keyword
         CR    R0,R2                   Correct length ?
         BNE   RXQRYV7                 No
         LA    R15,0                   Return code = 0
         LA    R0,L'VERSION            Length of result string
         LA    R1,VERSION              Address of result string
         BR    R14                     Return to REXXWAIT
RXQRYV7  LA    R15,7                   Return code = 7
```

```
240          LA    R0,0                   Length of result
241          LA    R1,0                   Address of result
242          BR    R14                    Return to REXXWAIT
243          DROP  R12                    No longer needed
244 ***********************************************************************
245 * Storage used by EXTINT52                                            *
246 ***********************************************************************
247 LOADED   EQU   X'80'                  When program loaded
248 WAITING  EQU   X'40'                  When program is waiting
249 E52FLAGS DC    XL1'00'                Program status flags
250 E52ECB   DC    F'0'                   Corresponding ECB
251 NLNAME   DC    CL8'EXTINT52'          Function name
252 KWDVER   DC    C'VERSION'             Version keyword
253 VERSION  DC    C'EXTINT52 1.00 16 November 1992' Version string
254 WAITMSG  DC    C'External Interrupt Type 0052'
255 TELLMSG  DC    C'EXTINT52 is a sample program in the REXXWAIT package'
256          DS    0F                     Alignment
257 RXWPLIST DC    CL8'REXXWAIT'          PLIST for REXXWAIT
258 RXWCMD   DC    CL4' '                 Command (filled in later)
259 RXWFLAGS DC    XL1'00'                Flags for REXXWAIT
260          DC    XL3'000000'            Special fence
261 RXWNAME  DC    CL8'EXT0052'           Registered program name
262 RXWWTECB DC    F'-1'                  ECB address
263 RXWWAIT  DC    F'-1'                  BALR address for WAIT
264 RXWWAITE DC    F'-1'                  BALR address for WAITEND
265 RXWSETV  DC    F'-1'                  BALR address for SETVALUE
266 RXWQRYV  DC    F'-1'                  BALR address for QUERYVALUE
267 RXWRSTV  DC    F'-1'                  BALR address for RESETVALUE
268          DC    F'-1'                  Not used
269          DC    F'-1'                  Not used
270          LTORG                        Literal pool
271 NUCXE    EQU   *                      End of nucxloaded code
272          REGEQU
273          NUCON
274          END
```

Note: The EXTINT52 program does not run on VM/SP 5 (and lower) systems, but it should run on VM/SP 6, VM/XA SP and VM/ESA systems in 370, XA and XC mode, if available.

6.2 Discussion

To explain the purpose, the different sections of the EXTINT52 Assembler program are discussed:

- Lines 1 to 30 are comments explaining the use of the program and the command sequence to generate a module.
- Lines 31 to 53 are executed (in the transient area) when the program is called the first time. The parameters passed to it are tested. If correct, the corresponding action is performed, otherwise, the program terminates with a return code 4 (in line 52 and 53).
- Lines 57 to 92 react on no parameter or the parameter LOAD, obtain storage (line 70) in nucleus and copy the part of the program between the label NUCXB (line 109) and NUCXE (line 271) into this storage. The program part is also made known as a nucleus extension (line 78). Finally, the program jumps into the nucleus loaded version (line 83 to 86) when the parameter was LOAD.
- Lines 96 to 97 react on the parameter RESET when not yet loaded as a nucleus extension. The call immediately terminates with a zero return code.
- Lines 101 to 102 react on the parameter TEST and terminate with a return code 1 indicating that the program is not yet loaded as a nucleus extension.
- Lines 109 to 125 test similarly to lines 39 to 53 the parameters passed to the program and continues accordingly.

- Lines 129 to 148 react on the parameter LOAD. An interrupt handler for external interrupt 0052 gets installed (line 133), and the program calls REXX/WAIT to register the name 'EXT0052' (line 261).
- Lines 152 to 159 react on the parameter RESET and deactivate the interrupt handler and de-register from REXX/WAIT. (Note that the program is automatically called with the parameter RESET when the program gets dropped from nucleus.)
- Lines 163 to 164 react on the parameter TEST by terminating with a zero return code indicating that the program is already loaded.
- Lines 169 to 176 react on the parameter '?', both for the code running in the transient area and the code running as a nucleus extension.
- Lines 181 to 186 show the interrupt handler for external interrupts 0052. Note that the ECB is only posted when REXX/WAIT is really waiting.
- Lines 191 to 210 react on a call of the WAIT function. The WAITING flag is set for the interrupt handler. When the ECB is posted, it terminates with a return code 0 (and the result from line 254). Otherwise, it terminates with a return code 1. (Any parameters passed with the WAIT function result in a return code 7.)
- Lines 215 to 223 react on the termination of the WAIT function. The WAITING flag is cleared for the interrupt handler, and the ECB is cleared.
- Lines 228 to 242 react on a call of the QUERYVALUE function. Only the keyword VERSION is accepted and the version string (line 253) is returned.
- Lines 247 to 270 show the storage used by the program. The parameter list for REXX/WAIT starts at line 257 and ends at line 269.

7.0 Syntax of the Functions Related to Translation

The package also contains a second group of four REXX functions called AC2EC, EC2AC, CTYPE and CTABLE. They are related to ASCII/EBCDIC translation, and their syntax is summarized in Figure 3. The AC2EC and EC2AC functions allow translation of ASCII strings to EBCDIC and vice versa. The CTYPE function returns whether the system on which this REXX function is invoked encodes character strings in ASCII or EBCDIC. Finally, the CTABLE function can be used to manipulate the translation tables for the AC2EC and EC2AC functions.

AC2EC(string)
EC2AC(string)
CTYPE([type])
CTABLE(type,[option],[start],[end],[table])

Figure 3. Summary of translation related REXX functions in REXX/WAIT

All functions return a —possibly empty —result string depending on the function and arguments specified. The behavior of the individual functions is described in the next sections in more details.

7.1 The REXX Function AC2EC

AC2EC(asciistring)

Result: ebcdicstring

The function returns the ASCII string converted to EBCDIC encoding. The translation table for ASCII to EBCDIC translation can be changed with the CTABLE function.

Examples

```
AC2EC('414243'x) == 'ABC'              /* On EBCDIC systems */
```

7.2 The REXX Function EC2AC

EC2AC(ebcdicstring)

Result: asciistring

The function returns the EBCDIC string converted to ASCII encoding. The translation table for EBCDIC to ASCII translation can be changed with the CTABLE function.

Examples

```
EC2AC('ABC') == '414243'x              /* On EBCDIC systems */
```

7.3 The REXX Function CTYPE

CTYPE([**type**]**)**

Result: type or boolean value

The function returns a boolean value if the type (either 'ASCII' or 'EBCDIC') is specified or the type (always 'EBCDIC' on VM) otherwise.

Examples

```
CTYPE()         == 'EBCDIC'            /* On EBCDIC systems */
CTYPE('EBCDIC') == '1'
CTYPE('ASCII')  == '0'
```

7.4 The REXX Function CTABLE

CTABLE(type,[**option**]**,**[**start**]**,**[**end**]**,**[**table**]**)**

Result: table

The function returns the current translation table (in the range between start and end) for the type ('AC2EC' or 'EC2AC') before it is changed, and sets or resets the table in the range depending on the option ('GET', 'SET', 'RESET' or 'MAP') specified. Values for start and end are one character each, and the default values are '00'x and 'FF'x, respectively.

The table must not be specified except for the 'SET' option, where its size must exactly fit the range between the start and end parameter. The 'GET' option does not modify the table but only returns it. The 'SET' option sets the translation table as specified. The 'RESET' option resets the table to the initial default. The 'MAP' option sets the translation table by using the other translation table and computing the inverse translation. (To change the translation tables, one usually sets one table with the 'SET' option and synchronizes the other table with the 'MAP' option.)

Examples

```
CTABLE('EC2AC','SET',,,table) == ...    /* On EBCDIC systems */
CTABLE('AC2EC','MAP')         == ...
```

7.5 General Remarks

This section presents some general remarks on the use of the functions related to translation:

1. The initial translation tables (corresponding to the ISO Standard) are one-to-one mappings. In other words, the equation `AC2EC(EC2AC(string))=EC2AC(AC2EC(string))=string` is true for all character strings.
2. Using the CTABLE function, either the AC2EC or the EC2AC translation table can be set to any other mapping. The inverse mapping is determined if the CTABLE function is used with the 'MAP' option for the other table. If the mapping is not one-to-one, the values not available for the inverse mapping are set to '00'x, and the ambiguous values are set to the lowest value.
3. The start and end values for CTABLE are single characters. The CTABLE function gets or sets the translation table only in the range of these hexadecimal positions. The defaults are '00'x and 'ff'x for start and end position, respectively. It does not always make sense to set other values than the default, but all options support them consistently.

Index