UNIX-Type APIs (V5R2)

Generic Terminal APIs

Table of Contents

Generic Terminal APIs

The Generic Terminal APIs are:

- Qp0zControlTerminal() (Control a Generic Terminal) allows a program to control the terminal window to which it is connected.
- Qp0zEndTerminal() (End a Generic Terminal) ends the terminal session specified by handle.
- Qp0zGetTerminalPid() (Get Process ID for a Generic Terminal) returns the process ID of the interpreter process for the terminal specified by handle.
- Qp0zIsATerminal() (Determine Whether Descriptor Is Connected to a Generic Terminal) determines if the specified descriptor is connected to a terminal.
- Qp0zRunTerminal() (Run a Generic Terminal) runs the terminal specified by handle.
- Qp0zSetTerminalMode() (Set Modes for a Generic Terminal) allows a program to control the input mode and wrap mode of the terminal window to which it is connected.
- Qp0zStartTerminal() (Start a Generic Terminal) starts a new terminal.

Generic Terminal Concepts

The Generic Terminal provides an environment for running programs that use descriptors for reading input and writing output. Typically the programs are C, C++, or Java programs that read input from standard input, write regular output to standard output, and write error output to standard error.

A terminal is started, run, and ended from an interactive job. When a terminal is started by Qp0zStartTerminal(), an interpreter process is started in batch with descriptors 0, 1, and 2 connected to pipes in the interactive job. A user specified program runs in the interpreter process. After calling Qp0zRunTerminal(), an interactive user can send input to the program and see the output written by the program. The resources used by the terminal are cleaned up by calling Qp0zEndTerminal(). It closes the pipes and ends the interpreter process.

Terminal Window

After calling Qp0zRunTerminal(), the terminal window is displayed. The interactive user enters input that is sent to the interpreter process and sees output that comes from the interpreter process. The terminal window has these parts:

- A title line identifies the terminal window. The title is set in the Qp0z_Terminal_Attr_T parameter of Qp0zStartTerminal().
- An output area that contains an echo of the commands that were entered and any output from the interpreter process. When a program in the interpreter process writes to descriptors 1 or 2, the output is displayed in the output area.
- An input line for entering commands. The input is written to descriptor 0 in the interpreter process.
- A command key description. There are two lines of command key descriptions that are set in the Qp0z_Terminal_Attr_T parameter of Qp0zStartTerminal().

- A message line where messages to the user are displayed.

The terminal window supports these command keys:

Command Key	Description
F3 (Exit)	Returns to the caller of Qp0zRunTerminal() with a return value of 1 (or QP0Z_TERMINAL_F3).
F5 (Refresh)	Refreshes the output area.
F6 (Print)	Prints the output area to a QPRINT spool file.
F7 (Page up)	Page up output area. If a number is on the command line, the output area is rolled up by that number of lines.
F8 (Page down)	Page down output area. If a number is on the command line, the output area is rolled down by that number of lines.
F9 (Retrieve)	Retrieve a previous command. If the key is pressed multiple times, it retrieves previous commands from a buffer. For example, to retrieve the second to last command, press the key two times. A specific command can be selected by placing the cursor on that command and pressing the key. When the interactive job is running in a double-byte CCSID, this key is not available.
F11 (Toggle line wrap)	Toggles the line wrap/truncate mode in the output area. In line wrap mode, lines longer than the width of the terminal window are wrapped to the next line. In truncate mode, the portion of a line beyond the width of the terminal window is not shown.
F12 (Return)	Returns to the caller of Qp0zRunTerminal() with a return value of 0 (or QP0Z_TERMINAL_F12).
F13 (Clear)	Clears the output area.
F14 (Adjust command line length)	Adjust the command line length to four lines. If a number is on the command line, the command line length is adjusted to that number of lines.
F17 (Top)	Displays top of output area.
F18 (Bottom)	Displays bottom of output area.
F19 (Left)	Shifts the output area to the left. If a number is on the command line, the output area is shifted by that number of columns.
F20 (Right)	Shifts the output area to the right. If a number is on the command line, the output area is shifted by that number of columns.
F21 (CL command line)	Displays a command entry window where the user can enter CL commands.

Programs running in the interpreter process

The program can use descriptor 0 (or standard input) to read input, descriptor 1 (or standard output) to write regular output, and descriptor 2 (or standard error) to write error output. The program can use the following functions to work with the terminal to which it is connected.

- Use Qp0zIsATerminal() to see if a descriptor is connected to a terminal.
- Use Qp0zControlTerminal() to control the terminal window. For example, page up or page down in the terminal window.
- Use Qp0zSetTerminalMode() to set terminal modes. For example, switch to hidden input mode to read a password.>

The program also needs to decide how to handle the following signals:

- The terminal sends signal SIGINT when the interactive user enters SysReq 2 to interrupt the current request.
- The terminal sends signal SIGHUP when the terminal is ended.

Qp0zControlTerminal()--Control a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zControlTerminal( unsigned char action, int value );
```

Service Program Name: QP0ZTRMLC   Default Public Authority: *USE   Threadsafe: Yes

The **Qp0zControlTerminal()** function allows a program to control the terminal window to which it is connected. A program can perform the same actions on the terminal window as an interactive user of the terminal window. See Generic Terminal Concepts for details about using a terminal.

Qp0zControlTerminal() supports the following actions:

QP0Z_TERMINAL_BOTTOM (0xB6)

Display bottom of output area. The bottom of the output area is displayed.

QP0Z_TERMINAL_CLCMDLINE (0xB9)

Display CL command line. A pop-up window with a CL command line is displayed. The user can run a CL command without exiting the terminal window.

QP0Z_TERMINAL_CLEAR (0xB1)

Clear output area. The contents of the output area and the command retrieval buffer are cleared.

QP0Z_TERMINAL_EXIT (0x33)

Exit terminal window. The terminal window is ended and Qp0zRunTerminal() returns 1 (or QP0Z_TERMINAL_F3).

QP0Z_TERMINAL_LEFT (0xB7)

Shift output area left. The output area is shifted to the left by the number of columns specified by *value*. If *value* is zero, the output area is shifted left by the number columns currently in the output area.

QP0Z_TERMINAL_PAGEDOWN (0x38)

Page down output area. The output area is moved down by the number of rows specified by *value*. If *value* is zero, the output area is moved down by the number rows currently in the output area (one page).

QP0Z_TERMINAL_PAGEUP (0x37)

Page up output area. The output area is moved up by the number of rows specified by *value*. If *value* is zero, the output area is moved up by the number rows currently in the output area (one page).

QP0Z_TERMINAL_PRINT (0x36)

Print output area. The contents of the output area are printed to a QPRINT spool file.

QP0Z_TERMINAL_REFRESH (0x35)

Refresh output area. The contents of the output area are refreshed with any output that is available.

QP0Z_TERMINAL_RETRIEVE (0x39)

Retrieve previous command. The last command entered by the user is retrieved and displayed on the input line.

QP0Z_TERMINAL_RETURN (0x3C)

Return from terminal window. The terminal window is ended and Qp0zRunTerminal() returns 0 (or QP0Z_TERMINAL_F12).

QP0Z_TERMINAL_RIGHT (0xB8)

Shift output area right. The output area is shifted to the right by the number of columns specified by *value*. If *value* is zero, the output area is shifted right by the number columns currently in the output area.

QP0Z_TERMINAL_TOP (0xB5)

Display top of output area. The top of the output area is displayed.

Parameters

action

(Input)

Action to perform on the terminal window. The valid values are listed above.

value

(Input)

Value associated with *action*. For the QP0Z_TERMINAL_LEFT and QP0Z_TERMINAL_RIGHT actions, the value is the number of columns to shift or zero for the default number of columns. For the QP0Z_TERMINAL_PAGEDOWN and QP0Z_TERMINAL_PAGEUP actions, the value is the number of rows to page up or down or zero for the default number of rows. For all other actions, this parameter must be zero.

Authorities and Locks

None.

Return Value

0	**Qp0zControlTerminal()** was successful.
value	**Qp0zControlTerminal()** was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zControlTerminal()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[EBADF]

Descriptor not valid.

A file descriptor argument was out of range, referred to a file that was not open, or a read or write request was made to a file that is not open for that operation.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values or an operation was attempted on an object and the operation specified is not supported for that type of object.

Correct the argument in error and try your request again.

[EIO]

Input/output error.

A physical I/O error occurred.

See the previous message in the job log. Correct any errors indicated there and try your operation again.

[ENOTTY]

Inappropriate I/O control operation.

[EUNKNOWN]

Unknown system state.

The operation failed due to an unknown system state. See any messages in the job log and correct any errors that may be indicated and then retry the operation.

Usage Notes

1. Before calling **Qp0zControlTerminal()**, a program should check to see if descriptor 0 is connected to a terminal by calling Qp0zIsATerminal().

2. There is no way for the Generic Terminal to prevent multiple programs calling **Qp0zControlTerminal()** to control the terminal window. A program must provide appropriate synchronization between calls to **Qp0zControlTerminal()** to avoid confusing the user of the terminal.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal
- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal

API Introduced: V5R1

Qp0zEndTerminal()--End a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zEndTerminal( Qp0z_Terminal_T handle, ... );
```

Service Program Name: QP0ZTRML
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zEndTerminal()** function ends the terminal session specified by *handle*.

The terminal session is ended by:

1. Ending the terminal window.
2. Sending the SIGHUP signal to the process group of the interpreter process.
3. Closing the pipes connected to the interpreter process.

Qp0zEndTerminal() waits for the interpreter process to end before returning to the caller. The status information about how the interpreter process ended is returned in the optional second parameter.

Parameters

handle

(Input) Handle for terminal.

...

(Output) An optional pointer to an integer to store the status information about how the interpreter process ended. See the wait() API for information on interpreting the status information. The status information is only returned when the Return_Exit_Status field is set in the Qp0z_Terminal_Attr_T parameter when the terminal is started by Qp0zStartTerminal().

Authorities

None.

Return Value

0

Qp0zEndTerminal() was successful.

value

Qp0zEndTerminal() was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zEndTerminal()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

[EIO]

Input/output error.

A physical I/O error occurred.

A referenced object may be damaged.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Usage Notes

1. The default action for the SIGHUP signal is to end the request. The program running in the interpreter process can use a signal handler to catch the signal and perform any necessary cleanup. See Signals APIs for more information about signals.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal

- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal
- wait()--Wait for Child Process to End

Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zGetTerminalPid( Qp0z_Terminal_T handle,
                        pid_t *pid );
```

Service Program Name: QP0ZTRML
Default Public Authority: *USE
Threadsafe: No

The **Qp0zGetTerminalPid()** function returns the process ID of the interpreter process for the terminal specified by *handle*.

Parameters

handle

(Input) Handle for terminal.

pid

(Output) Pointer to area to store process ID of interpreter process.

Authorities

None.

Return Value

0

Qp0zGetTerminalPid() was successful.

value

Qp0zGetTerminalPid() was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zGetTerminalPid()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal
- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal
- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal

Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zIsATerminal( int descriptor );
```

Service Program Name: QP0ZTRMLC
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zIsATerminal()** function determines if the specified descriptor is connected to a terminal. See Generic Terminal Concepts for details about using a terminal.

Parameters

descriptor

(Input) The descriptor to check.

Authorities

None.

Return Value

0

The *descriptor* is **not** connected to a terminal.

1

The *descriptor* is connected to a terminal.

Error Conditions

None.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal

- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal
- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal

Qp0zRunTerminal()--Run a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zRunTerminal( Qp0z_Terminal_T handle );
```

Service Program Name: QP0ZTRML

Default Public Authority: *USE

Threadsafe: No

The **Qp0zRunTerminal()** function runs the terminal specified by *handle*. First, **Qp0zRunTerminal()** makes the terminal window the active window on the display. Then, **Qp0zRunTerminal()** waits for the user to enter input at the command line, press a command key, or for output to become available from the interpreter process. **Qp0zRunTerminal()** returns when either the user presses F3, the user presses F12, or the interpreter process ends.

When the user enters input at the terminal command line, **Qp0zRunTerminal()** writes the data to descriptor 0 in the interpreter process. The data is terminated with a new line (0x25) character.

When a program in the interpreter process writes to descriptor 1 or 2, **Qp0zRunTerminal()** displays the data in the output area of the terminal window.

When the user presses one of the following command keys, **Qp0zRunTerminal()** takes these actions:

F3 (Exit)

Returns to the caller with a return value of 1 (or QP0Z_TERMINAL_F3).

F5 (Refresh)

Refreshes the output area.

F6 (Print)

Prints the output area to a QPRINT spool file.

F7 (Page up)

Page up output area. If a number is on the command line, the output area is rolled up by that number of lines.

F8 (Page down)

Page down output area. If a number is on the command line, the output area is rolled down by that number of lines.

F9 (Retrieve)

Retrieve a previous command. If the key is pressed multiple times, it retrieves previous commands from a buffer. For example, to retrieve the second to last command, press the key two times. A specific command can be selected by placing the cursor on that command and pressing the key. When the interactive job is running in a double-byte CCSID, this key is not available.

F11 (Toggle line wrap)

Toggles the line wrap/truncate mode in the output area. In line wrap mode, lines longer than the width of the terminal window are wrapped to the next line. In truncate mode, the portion of a line beyond the width of the terminal window is not shown.

F12 (Return)

Returns to the caller with a return value of 0 (or QP0Z_TERMINAL_F12).

F13 (Clear)

Clears the output area.

F14 (Adjust command line length)

Adjust the command line length to four lines. If a number is on the command line, the command line length is adjusted to that number of lines.

F17 (Top)

Displays top of output area.

F18 (Bottom)

Displays bottom of output area.

F19 (Left)

Shifts the output area to the left. If a number is on the command line, the output area is shifted by that number of columns.

F20 (Right)

Shifts the output area to the right. If a number is on the command line, the output area is shifted by that number of columns.

F21 (CL command line)

Displays a command entry window where the user can enter CL commands.

When the user enters System Request 2, **Qp0zRunTerminal()** sends a SIGINT signal to the process group of the interpreter process.

Parameters

handle

(Input) Handle for terminal.

Authorities

None.

Return Value

0 (or QP0Z_TERMINAL_F12)

Qp0zRunTerminal() was successful and the user pressed F12 to return.

1 (or QP0Z_TERMINAL_F3)

Qp0zRunTerminal() was successful and the user pressed F3 to exit.

2 (or QP0Z_TERMINAL_ENDED)

Qp0zRunTerminal() was successful and the interpreter process ended.

value

Qp0zRunTerminal() was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zRunTerminal()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[EDESTROYED]

The mutex was destroyed.

A required object was destroyed.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

[EIO]

Input/output error.

A physical I/O error occurred.

A referenced object may be damaged.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Usage Notes

1. The default action for the SIGINT signal is to end the request. The program running in the interpreter process can use a signal handler to catch the signal and perform any necessary cleanup. See Signals APIs for more information about signals.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal

- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal
- Using the Generic Terminal APIs (see Examples)

Qp0zSetTerminalMode()--Set Modes for a Generic Terminal

Syntax

```
#include <qp0ztrml.h>
int Qp0zSetTerminalMode( unsigned char mode, unsigned char  type,
                         unsigned char *reserved );;
```

Service Program Name: QP0ZTRMLC

Default Public Authority: *USE

Threadsafe: Yes

The **Qp0zSetTerminalMode()** function allows a program to control the input mode and wrap mode of the terminal window to which it is connected. See Generic Terminal Concepts for details about using a terminal.

Qp0zSetTerminalMode() supports setting the following modes:

QP0Z_TERMINAL_INPUT_MODE (0x01)

Set the input mode for the terminal window. When *type* is QP0Z_TERMINAL_HIDDEN (0xBD), any input entered by the user is not visible on the terminal window and is not echoed to the output area. When *type* is QP0Z_TERMINAL_NORMAL (0xBE), any input entered by the user is visible on the terminal window and is echoed to the output area. When *type* is QP0Z_TERMINAL_PREVIOUS (0x49), the input mode is set to its previous value.

QP0Z_TERMINAL_WRAP_MODE (0x02)

Set the wrap mode for the terminal window. When *type* is QP0Z_TERMINAL_TRUNCATE (0x3E), for lines longer than the width of the terminal window, only the data that fits in the output area is displayed. When *type* is QP0Z_TERMINAL_WRAP (0x3D), for lines longer than the width of the terminal window, the data is wrapped to the next line in the output area. When *type* is QP0Z_TERMINAL_PREVIOUS (0x49), the wrap mode is set to its previous value.

Parameters

mode

(Input)

Mode to set for the terminal window. The valid values are QP0Z_TERMINAL_INPUT_MODE and QP0Z_TERMINAL_WRAP_MODE.

type

(Input)

Type associated with the mode. The valid values for QP0Z_TERMINAL_INPUT_MODE are QP0Z_TERMINAL_HIDDEN, QP0Z_TERMINAL_NORMAL, and QP0Z_TERMINAL_PREVIOUS. The valid values for QP0Z_TERMINAL_WRAP_MODE are

QP0Z_TERMINAL_TRUNCATE, QP0Z_TERMINAL_WRAP, and QP0Z_TERMINAL_PREVIOUS.

reserved

(Output)

Reserved parameter that must be set to NULL.

Authorities and Locks

None.

Return Value

0	**Qp0zSetTerminalMode()** was successful.
value	**Qp0zSetTerminalMode()** was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zSetTerminalMode()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[EBADF]

Descriptor not valid.

A file descriptor argument was out of range, referred to a file that was not open, or a read or write request was made to a file that is not open for that operation.

[EFAULT]

The address used for an argument was not correct.

In attempting to use an argument in a call, the system detected an address that was not valid.

Correct the argument in error.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values or an operation was attempted on an object and the operation specified is not supported for that type of object.

Correct the argument in error and try your request again.

[EIO]

Input/output error.

A physical I/O error occurred.

See the previous message in the job log. Correct any errors indicated there and try your operation again.

[ENOTTY]

Inappropriate I/O control operation.

[EUNKNOWN]

Unknown system state.

The operation failed due to an unknown system state. See any messages in the job log and correct any errors that may be indicated and then retry the operation.

Usage Notes

1. Before calling **Qp0zSetTerminalMode()**, a program should check to see if descriptor 0 is connected to a terminal by calling Qp0zIsATerminal().
2. There is no way for the Generic Terminal to prevent multiple programs calling **Qp0zSetTerminalMode()** to control the terminal. A program must provide appropriate synchronization between calls to **Qp0zSetTerminalMode()** to avoid confusing the user of the terminal.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal
- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal
- Qp0zStartTerminal()--Start a Generic Terminal

API Introduced: V5R1

Qp0zStartTerminal()--Start a Generic Terminal

Syntax

```
#include <qp0ztrml.h>

int Qp0zStartTerminal( Qp0z_Terminal_T *handle,
                       char *args[],
                       char *envs[],
                       Qp0z_Terminal_Attr_T attr);
```

Service Program Name: QP0ZTRML

Default Public Authority: *USE

Threadsafe: No

The **Qp0zStartTerminal()** function starts a new terminal by:

- starting a new interpreter process running the program specified in *args[0]*,
- creating pipes connected to descriptors 0, 1, and 2 in the interpreter process, and
- starting a terminal window.

The interpreter process is started with the environment variables specified in *envs*. Using *attr*, you can set attributes for the terminal, including the inheritance structure used by spawn() to start the interpreter process and the title line and command key descriptions in the terminal window. The program running in the interpreter process receives the arguments specified in *args*.

In the interpreter process, descriptors 0, 1, and 2 are connected to pipes in the process that started the terminal. When a command is entered in the terminal window, it is written to descriptor 0 in the interpreter process. When a program in the interpreter process writes to descriptors 1 or 2, the data is displayed in the terminal window.

After a new terminal is started, you must call Qp0zRunTerminal() to wait for the user to enter input at the command line, press a command key, or for output from the interpreter process to be displayed.

Parameters

***handle**

(Output) A pointer to the area to store the terminal handle. When successful, **Qp0zStartTerminal()** returns a handle to the started terminal.

***args**

(Input) A null-terminated array of pointers to the arguments passed to the interpreter program. The first element in the array is a pointer to the path name of the program to start in the interpreter process.

***envs**

(Input) A null-terminated array of pointers to the environment variables inherited by the interpreter process. If this parameter is NULL, the environment variables currently defined when

Qp0zStartTerminal() is called are inherited by the interpreter process.

attr

(Input) Attributes for the terminal session.

The members of the Qp0z_Terminal_Attr_T structure are as follows:

struct inherit Inherit

The inheritance structure used when calling spawn() to start the interpreter process. Using the inheritance structure you can control the attributes of the interpreter process.

int Buffer_Size

Size of buffer for reading data from interpreter process. If zero is specified, **Qp0zStartTerminal()** uses a default buffer size of 4096 bytes.

char DBCS_Capable

This field is no longer used.

char Return_Exit_Status

Return the exit status of the interpreter process from Qp0zEndTerminal(). You must specify an optional parameter when calling Qp0zEndTerminal() to receive the exit status.

char Send_End_Msg

Send message CPCA989 when the interpreter process ends during Qp0zRunTerminal(). The message is displayed on the message line of the terminal window to alert the user that the interpreter process has ended.

»*char Return_On_End*

Return immediately from Qp0zRunTerminal() when the interpreter process ends. By default, Qp0zRunTerminal() waits for the user to press either the F3 or F12 command key before returning when the interpreter process ends.«

*char *Title*

Pointer to null-terminated string with the title for the terminal window. If the string is too long to fit in the terminal window, it is truncated to the width of the window.

*char *Cmd_Key_Line1*

Pointer to null-terminated string with the first line of command key descriptions for the terminal window. If the string is too long to fit in the terminal window, it is truncated to the width of the window.

*char *Cmd_Key_Line2*

Pointer to null-terminated string with the second line of command key descriptions for the terminal window. If the string is too long to fit in the terminal window, it is truncated to the width of the window.

char reserved2[32]

Reserved field that must be set to zero.

Authorities

Figure 1-2. Authorization Required for Qp0zStartTerminal()

Object Referred to	Authority Required	errno

Each directory in the path name preceding the executable file that will run in the interpreter process	*X	EACCES
Executable file that will run in the interpreter process	*X	EACCES
If executable file that will run in the interpreter process is a shell script	*RX	EACCES

Return Value

0

Qp0zStartTerminal() was successful.

value

Qp0zStartTerminal() was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zStartTerminal()** is not successful, the return value usually indicates one of the following errors. Under some conditions, the return value could indicate an error other than those listed here.

[E2BIG]

Argument list too long.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

[EBUSY]

Resource busy.

An attempt was made to use a system resource that is not available at this time. A terminal session is already active in the job and another one cannot be started.

[ECONVERT]

Conversion error.

One or more characters could not be converted from the source CCSID to the target CCSID.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

[EIO]

Input/output error.

A physical I/O error occurred.

A referenced object may be damaged.

[ELOOP]

A loop exists in the symbolic links.

This error is issued if the number of symbolic links encountered is more than POSIX_SYMLOOP (defined in the limits.h header file). Symbolic links are encountered during resolution of the directory or path name.

[EMFILE]

Too many open files for this process.

An attempt was made to open more files than allowed by the value of OPEN_MAX. The value of OPEN_MAX can be retrieved using the sysconf() function.

The process has more than OPEN_MAX descriptors already open (see the **sysconf()** function).

[ENAMETOOLONG]

A path name is too long.

A path name is longer than PATH_MAX characters or some component of the name is longer than NAME_MAX characters while _POSIX_NO_TRUNC is in effect. For symbolic links, the length of the name string substituted for a symbolic link exceeds PATH_MAX. The PATH_MAX and NAME_MAX values can be determined using the **pathconf()** function.

[ENFILE]

Too many open files in the system.

A system limit has been reached for the number of files that are allowed to be concurrently open in the system.

The entire system has too many other file descriptors already open.

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

[ENOMEM]

Storage allocation request failed.

A function needed to allocate storage, but no storage is available.

There is not enough memory to perform the requested function.

[ENOTDIR]

Not a directory.

A component of the specified path name existed, but it was not a directory when a directory was expected.

Some component of the path name is not a directory, or is an empty string.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Usage Notes

1. Only one terminal at a time can be active in an interactive job. If a terminal is currently active, **Qp0zStartTerminal()** returns EBUSY.

2. If the interpreter program is a C or C++ program, it must be compiled for Integrated File System I/O by specifying the SYSIFCOPT(*IFSIO) parameter on the command used to create the program.

3. If the interpreter program is a C or C++ program, the environment variable QIBM_USE_DESCRIPTOR_STDIO=Y must be set in the interpreter process to enable the program to use descriptors 0, 1, and 2 for standard input, standard output, and standard error.

4. The interpreter program can always read and write directly to descriptors 0, 1, and 2 regardless of the language it is compiled with.

5. It is the responsibility of the interpreter program to end and cleanup any open resources when the descriptors are closed by the terminal, it receives the SIGHUP signal, or it receives the SIGINT signal.

Related Information

- The <**qp0ztrml.h**> file (see Header Files for UNIX-Type Functions)
- Qp0zControlTerminal()--Control a Generic Terminal
- Qp0zEndTerminal()--End a Generic Terminal
- Qp0zGetTerminalPid()--Get Process ID for a Generic Terminal
- Qp0zIsATerminal()--Determine Whether Descriptor Is Connected to a Generic Terminal
- Qp0zRunTerminal()--Run a Generic Terminal

- Qp0zSetTerminalMode()--Set Modes for a Generic Terminal
- spawn()--Spawn Process

Header Files for UNIX-Type Functions

Programs using the UNIX-type functions must include one or more header files that contain information needed by the functions, such as:

- Macro definitions
- Data type definitions
- Structure definitions
- Function prototypes

The header files are provided in the QSYSINC library, which is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files. For information on installing the QSYSINC library, see Data structures and the QSYSINC Library.

The table below shows the file and member name in the QSYSINC library for each header file used by the UNIX-type APIs in this publication.

Name of Header File	Name of File in QSYSINC	Name of Member
arpa/inet.h	ARPA	INET
arpa/nameser.h	ARPA	NAMESER
bse.h	H	BSE
bsedos.h	H	BSEDOS
bseerr.h	H	BSEERR
dirent.h	H	DIRENT
errno.h	H	ERRNO
fcntl.h	H	FCNTL
grp.h	H	GRP
»inttypes.h	H	INTTYPES«
limits.h	H	LIMITS
»mman.h	H	MMAN«
netdbh.h	H	NETDB
»netinet/icmp6.h	NETINET	ICMP6«
net/if.h	NET	IF
netinet/in.h	NETINET	IN
netinet/ip_icmp.h	NETINET	IP_ICMP
netinet/ip.h	NETINET	IP
»netinet/ip6.h	NETINET	IP6«
netinet/tcp.h	NETINET	TCP
netinet/udp.h	NETINET	UDP
netns/idp.h	NETNS	IDP
netns/ipx.h	NETNS	IPX
netns/ns.h	NETNS	NS
netns/sp.h	NETNS	SP
net/route.h	NET	ROUTE
nettel/tel.h	NETTEL	TEL

os2.h	H	OS2
os2def.h	H	OS2DEF
pwd.h	H	PWD
Qlg.h	H	QLG
qp0lflop.h	H	QP0LFLOP
»qp0ljrnl.h	H	QP0LJRNL«
»qp0lror.h	H	QP0LROR«
Qp0lstdi.h	H	QP0LSTDI
qp0wpid.h	H	QP0WPID
qp0zdipc.h	H	QP0ZDIPC
qp0zipc.h	H	QP0ZIPC
qp0zolip.h	H	QP0ZOLIP
qp0zolsm.h	H	QP0ZOLSM
qp0zripc.h	H	QP0ZRIPC
qp0ztrc.h	H	QP0ZTRC
qp0ztrml.h	H	QP0ZTRML
qp0z1170.h	H	QP0Z1170
»qsoasync.h	H	QSOASYNC«
qtnxaapi.h	H	QTNXAAPI
qtnxadtp.h	H	QTNXADTP
qtomeapi.h	H	QTOMEAPI
qtossapi.h	H	QTOSSAPI
resolv.h	H	RESOLVE
semaphore.h	H	SEMAPHORE
signal.h	H	SIGNAL
spawn.h	H	SPAWN
ssl.h	H	SSL
sys/errno.h	H	ERRNO
sys/ioctl.h	SYS	IOCTL
sys/ipc.h	SYS	IPC
sys/layout.h	H	LAYOUT
sys/limits.h	H	LIMITS
sys/msg.h	SYS	MSG
sys/param.h	SYS	PARAM
»sys/resource.h	SYS	RESOURCE«
sys/sem.h	SYS	SEM
sys/setjmp.h	SYS	SETJMP
sys/shm.h	SYS	SHM
sys/signal.h	SYS	SIGNAL
sys/socket.h	SYS	SOCKET
sys/stat.h	SYS	STAT
sys/statvfs.h	SYS	STATVFS

sys/time.h	SYS	TIME
sys/types.h	SYS	TYPES
sys/uio.h	SYS	UIO
sys/un.h	SYS	UN
sys/wait.h	SYS	WAIT
»ulimit.h	H	ULIMIT«
unistd.h	H	UNISTD
utime.h	H	UTIME

You can display a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to display the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(5)
  ```
- Using the Display Physical File Member command. For example, to display the **sys/stat.h** header file, enter the following command:

  ```
  DSPPFM FILE(QSYSINC/SYS) MBR(STAT)
  ```

You can print a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to print the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(6)
  ```
- Using the Copy File command. For example, to print the **sys/stat.h** header file, enter the following command:

  ```
  CPYF FROMFILE(QSYSINC/SYS) TOFILE(*PRINT) FROMMBR(STAT)
  ```

Symbolic links to these header files are also provided in directory /QIBM/include.

Top | UNIX-Type APIs | APIs by category

Errno Values for UNIX-Type Functions

Programs using the UNIX-type functions may receive error information as *errno* values. The possible values returned are listed here in ascending *errno* value sequence.

Name	**Value**	**Text**
EDOM	3001	A domain error occurred in a math function.
ERANGE	3002	A range error occurred.
ETRUNC	3003	Data was truncated on an input, output, or update operation.
ENOTOPEN	3004	File is not open.
ENOTREAD	3005	File is not opened for read operations.
EIO	3006	Input/output error.
ENODEV	3007	No such device.
ERECIO	3008	Cannot get single character for files opened for record I/O.
ENOTWRITE	3009	File is not opened for write operations.
ESTDIN	3010	The stdin stream cannot be opened.
ESTDOUT	3011	The stdout stream cannot be opened.
ESTDERR	3012	The stderr stream cannot be opened.
EBADSEEK	3013	The positioning parameter in fseek is not correct.
EBADNAME	3014	The object name specified is not correct.
EBADMODE	3015	The type variable specified on the open function is not correct.
EBADPOS	3017	The position specifier is not correct.
ENOPOS	3018	There is no record at the specified position.
ENUMMBRS	3019	Attempted to use ftell on multiple members.
ENUMRECS	3020	The current record position is too long for ftell.
EINVAL	3021	The value specified for the argument is not correct.
EBADFUNC	3022	Function parameter in the signal function is not set.
ENOENT	3025	No such path or directory.
ENOREC	3026	Record is not found.
EPERM	3027	The operation is not permitted.
EBADDATA	3028	Message data is not valid.
EBUSY	3029	Resource busy.
EBADOPT	3040	Option specified is not valid.
ENOTUPD	3041	File is not opened for update operations.
ENOTDLT	3042	File is not opened for delete operations.

EPAD	3043	The number of characters written is shorter than the expected record length.
EBADKEYLN	3044	A length that was not valid was specified for the key.
EPUTANDGET	3080	A read operation should not immediately follow a write operation.
EGETANDPUT	3081	A write operation should not immediately follow a read operation.
EIOERROR	3101	A nonrecoverable I/O error occurred.
EIORECERR	3102	A recoverable I/O error occurred.
EACCES	3401	Permission denied.
ENOTDIR	3403	Not a directory.
ENOSPC	3404	No space is available.
EXDEV	3405	Improper link.
EAGAIN	3406	Operation would have caused the process to be suspended.
EWOULDBLOCK	3406	Operation would have caused the process to be suspended.
EINTR	3407	Interrupted function call.
EFAULT	3408	The address used for an argument was not correct.
ETIME	3409	Operation timed out.
ENXIO	3415	No such device or address.
EAPAR	3418	Possible APAR condition or hardware failure.
ERECURSE	3419	Recursive attempt rejected.
EADDRINUSE	3420	Address already in use.
EADDRNOTAVAIL	3421	Address is not available.
EAFNOSUPPORT	3422	The type of socket is not supported in this protocol family.
EALREADY	3423	Operation is already in progress.
ECONNABORTED	3424	Connection ended abnormally.
ECONNREFUSED	3425	A remote host refused an attempted connect operation.
ECONNRESET	3426	A connection with a remote socket was reset by that socket.
EDESTADDRREQ	3427	Operation requires destination address.
EHOSTDOWN	3428	A remote host is not available.
EHOSTUNREACH	3429	A route to the remote host is not available.
EINPROGRESS	3430	Operation in progress.
EISCONN	3431	A connection has already been established.
EMSGSIZE	3432	Message size is out of range.
ENETDOWN	3433	The network currently is not available.
ENETRESET	3434	A socket is connected to a host that is no longer available.

ENETUNREACH	3435	Cannot reach the destination network.
ENOBUFS	3436	There is not enough buffer space for the requested operation.
ENOPROTOOPT	3437	The protocol does not support the specified option.
ENOTCONN	3438	Requested operation requires a connection.
ENOTSOCK	3439	The specified descriptor does not reference a socket.
ENOTSUP	3440	Operation is not supported.
EOPNOTSUPP	3440	Operation is not supported.
EPFNOSUPPORT	3441	The socket protocol family is not supported.
EPROTONOSUPPORT	3442	No protocol of the specified type and domain exists.
EPROTOTYPE	3443	The socket type or protocols are not compatible.
ERCVDERR	3444	An error indication was sent by the peer program.
ESHUTDOWN	3445	Cannot send data after a shutdown.
ESOCKTNOSUPPORT	3446	The specified socket type is not supported.
ETIMEDOUT	3447	A remote host did not respond within the timeout period.
EUNATCH	3448	The protocol required to support the specified address family is not available at this time.
EBADF	3450	Descriptor is not valid.
EMFILE	3452	Too many open files for this process.
ENFILE	3453	Too many open files in the system.
EPIPE	3455	Broken pipe.
ECANCEL	3456	Operation cancelled.
EEXIST	3457	File exists.
EDEADLK	3459	Resource deadlock avoided.
ENOMEM	3460	Storage allocation request failed.
EOWNERTERM	3462	The synchronization object no longer exists because the owner is no longer running.
EDESTROYED	3463	The synchronization object was destroyed, or the object no longer exists.
ETERM	3464	Operation was terminated.
ENOENT1	3465	No such file or directory.
ENOEQFLOG	3466	Object is already linked to a dead directory.
EEMPTYDIR	3467	Directory is empty.
EMLINK	3468	Maximum link count for a file was exceeded.

ESPIPE	3469	Seek request is not supported for object.
ENOSYS	3470	Function not implemented.
EISDIR	3471	Specified target is a directory.
EROFS	3472	Read-only file system.
EUNKNOWN	3474	Unknown system state.
EITERBAD	3475	Iterator is not valid.
EITERSTE	3476	Iterator is in wrong state for operation.
EHRICLSBAD	3477	HRI class is not valid.
EHRICLBAD	3478	HRI subclass is not valid.
EHRITYPBAD	3479	HRI type is not valid.
ENOTAPPL	3480	Data requested is not applicable.
EHRIREQTYP	3481	HRI request type is not valid.
EHRINAMEBAD	3482	HRI resource name is not valid.
EDAMAGE	3484	A damaged object was encountered.
ELOOP	3485	A loop exists in the symbolic links.
ENAMETOOLONG	3486	A path name is too long.
ENOLCK	3487	No locks are available.
ENOTEMPTY	3488	Directory is not empty.
ENOSYSRSC	3489	System resources are not available.
ECONVERT	3490	Conversion error.
E2BIG	3491	Argument list is too long.
EILSEQ	3492	Conversion stopped due to input character that does not belong to the input codeset.
ETYPE	3493	Object type mismatch.
EBADDIR	3494	Attempted to reference a directory that was not found or was destroyed.
EBADOBJ	3495	Attempted to reference an object that was not found, was destroyed, or was damaged.
EIDXINVAL	3496	Data space index used as a directory is not valid.
ESOFTDAMAGE	3497	Object has soft damage.
ENOTENROLL	3498	User is not enrolled in system distribution directory.
EOFFLINE	3499	Object is suspended.
EROOBJ	3500	Object is a read-only object.
EEAHDDSI	3501	Hard damage on extended attribute data space index.
EEASDDSI	3502	Soft damage on extended attribute data space index.
EEAHDDS	3503	Hard damage on extended attribute data space.
EEASDDS	3504	Soft damage on extended attribute data space.
EEADUPRC	3505	Duplicate extended attribute record.

ELOCKED	3506	Area being read from or written to is locked.
EFBIG	3507	Object too large.
EIDRM	3509	The semaphore, shared memory, or message queue identifier is removed from the system.
ENOMSG	3510	The queue does not contain a message of the desired type and (msgflg logically ANDed with IPC_NOWAIT).
EFILECVT	3511	File ID conversion of a directory failed.
EBADFID	3512	A file ID could not be assigned when linking an object to a directory.
ESTALE	3513	File handle was rejected by server.
ESRCH	3515	No such process.
ENOTSIGINIT	3516	Process is not enabled for signals.
ECHILD	3517	No child process.
EBADH	3520	Handle is not valid.
ETOOMANYREFS	3523	The operation would have exceeded the maximum number of references allowed for a descriptor.
ENOTSAFE	3524	Function is not allowed.
EOVERFLOW	3525	Object is too large to process.
EJRNDAMAGE	3526	Journal is damaged.
EJRNINACTIVE	3527	Journal is inactive.
EJRNRCVSPC	3528	Journal space or system storage error.
EJRNRMT	3529	Journal is remote.
ENEWJRNRCV	3530	New journal receiver is needed.
ENEWJRN	3531	New journal is needed.
EJOURNALED	3532	Object already journaled.
EJRNENTTOOLONG	3533	Entry is too large to send.
EDATALINK	3534	Object is a datalink object.
ENOTAVAIL	3535	IASP is not available.
ENOTTY	3536	I/O control operation is not appropriate.
EFBIG2	3540	Attempt to write or truncate file past its sort file size limit.
ETXTBSY	3543	Text file busy.
EASPGRPNOTSET	3544	ASP group not set for thread.
ERESTART	3545	A system call was interrupted and may be restarted.
