IBM

IBM Systems - iSeries
Generic terminal APIs
Version 5 Release 4

IBM

IBM Systems - iSeries

Generic terminal APIs

Version 5 Release 4

> **Note**
>
> Before using this information and the product it supports, be sure to read the information in "Notices," on page 19.

Contents

Generic Terminal APIs

The Generic Terminal APIs are:

- "Qp0zControlTerminal()—Control a Generic Terminal" on page 3 (Control a Generic Terminal) allows a program to control the terminal window to which it is connected.
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5 (End a Generic Terminal) ends the terminal session specified by handle.
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7 (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" on page 8 (Determine Whether Descriptor Is Connected to a Generic Terminal) determines if the specified descriptor is connected to a terminal.
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9 (Run a Generic Terminal) runs the terminal specified by handle.
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12 (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" on page 14 (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$^{(R)}$ 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()—Start a Generic Terminal" on page 14, 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()—Run a Generic Terminal" on page 9, 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()—End a Generic Terminal" on page 5. It closes the pipes and ends the interpreter process.

Terminal Window

After calling "Qp0zRunTerminal()—Run a Generic Terminal" on page 9, 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()—Start a Generic Terminal" on page 14.
- 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()—Start a Generic Terminal" on page 14.
- 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()—Run a Generic Terminal" on page 9 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()—Run a Generic Terminal" on page 9 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()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8 to see if a descriptor is connected to a terminal.
- Use "Qp0zControlTerminal()—Control a Generic Terminal" on page 3 to control the terminal window. For example, page up or page down in the terminal window.
- Use "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12 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.

APIs

These are the APIs for this category.

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 APIs," on page 1 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()—Run a Generic Terminal" on page 9 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()—Run a Generic Terminal" on page 9 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()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8.

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"—End a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7—Get Process ID for a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9—Run a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12—Set Modes for a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal" on page 14—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()—Start a Generic Terminal" on page 14.

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]

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.

An argument value is not valid, out of range, or NULL.

[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" on page 3—Control a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal"—Get Process ID for a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9—Run a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12—Set Modes for a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal" on page 14—Start a Generic Terminal
- wait()—Wait for Child Process to End

API introduced: V4R2

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]

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.

An argument value is not valid, out of range, or NULL.

[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" on page 3—Control a Generic Terminal
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5—End a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal"—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9—Run a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12—Set Modes for a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal" on page 14—Start a Generic Terminal

API introduced: V4R2

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 APIs," on page 1 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" on page 3—Control a Generic Terminal
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5—End a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7—Get Process ID for a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal"—Run a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12—Set Modes for a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal" on page 14—Start a Generic Terminal

API introduced: V4R3

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]

> 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.

> An argument value is not valid, out of range, or NULL.

[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" on page 3—Control a Generic Terminal
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5—End a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7—Get Process ID for a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal"—Set Modes for a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal" on page 14—Start a Generic Terminal
- Using the Generic Terminal APIs (see Examples)

API introduced: V4R3

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 APIs," on page 1 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()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8.

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" on page 3—Control a Generic Terminal
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5—End a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7—Get Process ID for a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9—Run a Generic Terminal
- "Qp0zStartTerminal()—Start a Generic Terminal"—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, the title line and command key descriptions in the terminal window, and the i5/OS simple job name of the interpreter process. 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()—Run a Generic Terminal" on page 9 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()—End a Generic Terminal" on page 5. You must specify an optional parameter when calling "Qp0zEndTerminal()—End a Generic Terminal" on page 5 to receive the exit status.
char Send_End_Msg	Send message CPCA989 when the interpreter process ends during "Qp0zRunTerminal()—Run a Generic Terminal" on page 9. 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()—Run a Generic Terminal" on page 9 when the interpreter process ends. By default, "Qp0zRunTerminal()—Run a Generic Terminal" on page 9 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 Interpreter_Process_Name[10]	The 10 character i5/OS simple job name to use for the interpreter process. If specified, this field must be uppercase, and contain only those characters allowed for an i5/OS job name. » However, a period (.) is not considered a valid character. «
char reserved2[22]	Reserved field that must be set to zero.

Authorities

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]

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.

An argument value is not valid, out of range, or NULL.

[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" on page 3—Control a Generic Terminal
- "Qp0zEndTerminal()—End a Generic Terminal" on page 5—End a Generic Terminal
- "Qp0zGetTerminalPid()—Get Process ID for a Generic Terminal" on page 7—Get Process ID for a Generic Terminal
- "Qp0zIsATerminal()—Determine Whether Descriptor Is Connected to a Generic Terminal" on page 8—Determine Whether Descriptor Is Connected to a Generic Terminal
- "Qp0zRunTerminal()—Run a Generic Terminal" on page 9—Run a Generic Terminal
- "Qp0zSetTerminalMode()—Set Modes for a Generic Terminal" on page 12—Set Modes for a Generic Terminal
- spawn()—Spawn Process

API introduced: V5R1

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

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

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

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

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

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

```
IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.
```

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) IBM 2006. Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 1998, 2006. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This Application Programming Interfaces (API) publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
Advanced 36
Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI
DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
i5/OS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Terms and Conditions

Permissions for the use of these Publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these Publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these Publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE

IBM ®

Printed in USA