UNIX-Type APIs (V5R2)

Environment Variable APIs

Table of Contents

Environment Variable APIs

Environment variables are character strings of the form "name=value". There are two types of environment variables:

- Job-level environment variables. The job-level environment variables are stored in an environment space outside of the program associated with the job. They can be manipulated by using the **getenv()**, **putenv()**, **Qp0zDltEnv()**, **Qp0zGetEnv()**, **Qp0zInitEnv()**, and **Qp0zPutEnv()** APIs, as well as the CL commands ADDENVVAR, CHGENVVAR, RMVENVVAR, and WRKENVVAR. These variables exist for the duration of the job or until they are deleted. There is a limit of 4095 job-level environment variables.

- System-level environment variables. The system-level environment variables are stored in a global environment space that is persistent across IPLs and is not associated to a particular job. They can be manipulated by using the **Qp0zDltSysEnv()**, **Qp0zGetAllSysEnv()**, **Qp0zGetSysEnv()**, and **Qp0zPutSysEnv()** APIs, as well as the CL commands ADDENVVAR, CHGENVVAR, RMVENVVAR, and WRKENVVAR. These variables exist until they are deleted. There is a limit of 4095 system-level environment variables.

When a job calls one of the job-level environment variable APIs or CL commands for the first time, it inherits the system-level environment variables onto its job-level environment space. Any changes to job-level and system-level environment variables are then independent of one another.

The temporary space where the job-level environment variables are stored allows read and write access. Therefore, it is possible for the space to be corrupted. This could occur if a programmer accesses the space directly using the environ array rather than using the environment variable APIs. If the space is corrupted, subsequent calls using the APIs will have unpredictable results.

The environment variable APIs are:

- getenv() (Get value of environment variable) searches the job-level environment list for a string of the form name=value, where name is the environment variable and value is the value of the variable.
- putenv() (Change or add environment variable) sets the value of a job-level environment variable by changing an existing variable or creating a new one.
- Qp0zDltEnv() (Delete an environment variable) deletes a single job-level environment variable or deletes all environment variables from the current job.
- Qp0zDltSysEnv() (Delete a system-level environment variable) deletes a single system-level environment variable or deletes all system-level environment variables.
- Qp0zGetAllSysEnv() (Get all system-level environment variables) fills in the list_buf with a list of all the system-level environment variables.
- Qp0zGetEnv() (Get value of environment variable (extended)) is an OS/400 extension to the standard getenv() function.
- Qp0zGetSysEnv() (Get value of system-level environment variable) gets the value of a system-level environment variable name by searching the system-level environment variable list for a string of the form name=value.
- Qp0zInitEnv() (Initialize environment for variables) sets the external variable environ to a pointer to the current environment list.
- Qp0zPutEnv() (Change or add environment variable (extended)) is an OS/400 extension to the standard putenv() function.
- Qp0zPutSysEnv() (Change or add a system-level environment variable) sets the value of a

system-level environment variable by altering an existing variable or creating a new variable.

Note: These functions use header (include) files from the library QSYSINC, which is optionally installable. Make sure QSYSINC is installed on your system before using any of the functions. See Header Files for UNIX-Type Functions for the file and member name of each header file.

---

getenv()--Get Value of Environment Variable

Syntax

```
#include <stdlib.h>

char *getenv(const char *name);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Yes. See Usage Notes for more information.

The **getenv()** function searches the job-level environment list for a string of the form name=value, where name is the environment variable and value is the value of the variable.

The *name* parameter does not include the equal (=) symbol or the value of the environment variable name=value pair.

Parameters

name

(Input) The name of an environment variable.

Return Value

value	**getenv()** successfully found the environment string. The value returned is a pointer to the string containing the value for the specified name in the current environment.
NULL	**getenv()** could not find the environment string. The *errno* variable is set to indicate the error.

Error Conditions

If **getenv()** is not successful, *errno* indicates one of the following errors.

[EDAMAGE]	A damaged object was encountered. A referenced object is damaged. The object cannot be used.
[[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.
[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. No entry found for name specified.
[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. Although **getenv()** is threadsafe, if a thread calls an environment variable function while another thread is accessing an environment variable from the environ array the thread may see undefined results. The environ array can be accessed directly or by using a pointer returned from the **getenv()** or **Qp0zGetEnv()** functions. The environment contents are only protected during calls to the environment variable functions.

2. All environment variables are stored with an associated CCSID (coded character set identifier). Unless a different CCSID is specified, such as by using **Qp0zPutEnv()**, the default CCSID for the job is used as the CCSID associated with each environment variable string.

3. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

See the example of using **getenv()** in putenv()--Change or Add Environment Variable.

For other examples, see the following:

- Using Environment Variables
- Using the Spawn Process and Wait for Child Process APIs
- Using the Spawn Process (using NLS-enabled path name)

API Introduced: V3R6

---

putenv()--Change or Add Environment Variable

Syntax

```
#include <stdlib.h>

int putenv(const char *string);;
```

Threadsafe: Yes. See Usage Notes for more information.

The **putenv()** function sets the value of a job-level environment variable by changing an existing variable or creating a new one. The *string* parameter points to a string of the form name=value, where name is the environment variable and value is the new value for it.

The name cannot contain a blank. For example,

```
PATH NAME=/my_lib/joe_user
```

is not valid because of the blank between PATH and NAME. The name can contain an equal (=) symbol, but the system interprets all characters following the first equal symbol as being the value of the environment variable. For example,

```
PATH=NAME=/my_lib/joe_user
```

will result in a value of 'NAME=/my_lib/joe_user' for the variable PATH.

Parameters

string

(Input) A pointer to the name=value string.

Return Value

0

putenv() was successful.

-1

putenv() was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **putenv()** is not successful, *errno* indicates one of the following errors.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

[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.

For example, the string may not be in the correct format.

[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. (There is a limit of 4095 environment variables per job.)

[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. Although **putenv()** is threadsafe, if a thread calls an environment variable function while another thread is accessing an environment variable from the environ array the thread may see undefined results. The environ array can be accessed directly or by using a pointer returned from the **getenv()** or **Qp0zGetEnv()** functions. The environment contents are only protected during calls to the environment variable functions.
2. All environment variables are stored with an associated CCSID (coded character set identifier). Because **putenv()** does not specify a CCSID, the default CCSID for the job is used as the CCSID associated with strings that are stored using **putenv()**.
3. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- getenv()--Get Value of Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

The following example uses **putenv()** and **getenv()**.

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
 char     *var1  = "PATH=/:/home/userid";
 char     *name1 = "PATH";
 char     *val1  = NULL;
 int      rc;

 rc = putenv(var1);
 if (rc < 0) {
   printf("Error inserting <%s> in environ, errno = %d\n",
           var1, errno);
   return 1;
 }

 printf("<%s> inserted in environ\n", var1);
 val1 = getenv(name1);
 if (val1 == NULL) {
   printf("Error retrieving <%s> from environ, errno = %d\n",
           name1, errno);
   return 1;
 }

 printf("<%s> retrieved from environ, value is <%s>\n",
         name1, val1);
 return 0;
}
```

Output:

```
<PATH=/:/home/userid> inserted in environ
<PATH> retrieved from environ, value is </:/home/userid>
```

For other examples, see the following:

- Using Environment Variables.
- Using the Spawn Process and Wait for Child Process APIs.
- Using the Spawn Process (using NLS-enabled path name)

Top | Environment Variable APIs | APIs by category

Qp0zDltEnv()--Delete an Environment Variable

Syntax

```
#include <qp0z1170.h>

int Qp0zDltEnv(const char *name);
```

Threadsafe: Yes. See Usage Notes for more information.

The **Qp0zDltEnv()** function deletes a single job-level environment variable or deletes all environment variables from the current job. If the *name* parameter is NULL, all environment variables in the job are deleted.

The *name* parameter does not include the equal (=) symbol or the value of the environment variable name=value pair.

Parameters

name

(Input) A pointer to the name part of the environment variable name=value string.

Authorities

None.

Return Value

0

Qp0zDltEnv() was successful.

-1

Qp0zDltEnv() was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zDltEnv()** is not successful, *errno* indicates one of the following errors.

[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.

The parameter name is not NULL and does not point to an environment variable name that currently exists in the environment list.

Usage Notes

1. Although **Qp0zDltEnv()** is threadsafe, if a thread calls an environment variable function while another thread is accessing an environment variable from the environ array the thread may see undefined results. The environ array can be accessed directly or by using a pointer returned from the **getenv()** or **Qp0zGetEnv()** functions. The environment contents are only protected during calls to the environment variable functions.

Related Information

- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

The following example uses **Qp0zDltEnv()**, **putenv()** and the environ array.

```
#include <stdio.h>
#include <errno.h>
#include <qp0z1170.h>
#include <stdlib.h>

extern char **environ;

#define ASSERT(x, y)                          \
{ if (!(x)) {                                 \
    printf("Assertion Failed: " #x            \
           ", Description: " y                \
           ", errno=%d", errno);              \
    exit(EXIT_FAILURE);                       \
  }                                           \
}

int main(int argc, char **argv)
{
  int                   rc=0;
  int                   e=0;
  printf("Enter Testcase - %s\n", argv[0]);

  rc = putenv("PATH=/usr/bin:/home/me:%LIBL%");
  ASSERT((rc == 0), "putenv(PATH)");
  rc = putenv("TEST0=42");
  ASSERT((rc == 0), "putenv(TEST0)");
  rc = putenv("TEST1=42");
  ASSERT((rc == 0), "putenv(TEST1)");
  printf("Before delete, these environment variables are set: \n");

  while (environ[e] != NULL) {
    printf("  %s\n", environ[e]);
    ++e;
  }

  printf("Delete the environment variables\n");
  rc = Qp0zDltEnv("TEST0");
  ASSERT((rc==0), "Qp0zDltEnv(TEST0)");
  rc = Qp0zDltEnv("TEST1");
  ASSERT((rc==0), "Qp0zDltEnv(TEST1)");

  printf("After delete, these environment variables are set: \n");
  e=0;
  while (environ[e] != NULL) {
```

```
    printf("  %s\n", environ[e]);
    ++e;
  }
  printf("Main completed\n");
  return 0;
}
```

Output:

```
Enter Testcase - QP0WTEST/TPZDLTE0
Before delete, these environment variables are set:
  PATH=/usr/bin:/home/me:%LIBL%
  TEST0=42
  TEST1=42
Delete the environment variables
After delete, these environment variables are set:
  PATH=/usr/bin:/home/me:%LIBL%
Main completed
```

Qp0zDltSysEnv()--Delete a System-Level Environment Variable

Syntax

```
#include <qp0z1170.h>

int Qp0zDltSysEnv(const char *name, void  *reserved);
```

Threadsafe: Yes

The **Qp0zDltSysEnv()** function deletes a single system-level environment variable or deletes all system-level environment variables. If the *name* parameter is NULL, all system-level environment variables are deleted.

The *name* parameter does not include the equal (=) symbol or the value part of the environment variable name=value pair.

Parameters

name

(Input) The name of the environment variable to delete.

reserved

(Input) Reserved for future use. Currently, the only value allowed is NULL.

Authorities

*JOBCTL special authority is required to delete a system-level environment variable.

Return Value

0

Qp0zDltSysEnv() was successful.

errval

Qp0zDltSysEnv() was not successful. *errval* is set to indicate the error.

Error Conditions

If **Qp0zDltSysEnv()** is not successful, *errval* indicates one of the following errors.

[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.

The value for the *reserved* parameter was not NULL.

[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.

The parameter name is not NULL and does not point to an environment variable name that currently exists in the environment list.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

You must have *JOBCTL special authority to delete a system-level environment variable.

[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 <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

See the example of using **Qp0zDltSysEnv()** in Qp0zPutSysEnv()--Change or Add a System-Level Environment.

Qp0zGetAllSysEnv()--Get All System-Level Environment Variables

Syntax

```
#include <qp0z1170.h>

int Qp0zGetAllSysEnv(char *list_buf, int  *list_buf_size,
                 int *ccsid_buf, int  *ccsid_buf_size,
                 void *reserved);
```

Threadsafe: Yes

The **Qp0zGetAllSysEnv()** function fills in the list_buf with a list of all the system-level environment variables. The list consists of multiple null-terminated name=value strings followed by an ending null-terminator. The coded character set identifier (CCSID) associated with each name=value string is returned in the ccsid_buf buffer.

Authorities

None

Parameters

list_buf

(Input/Output) The address of the buffer to receive the null-terminated `name=value` list.

list_buf_size

(Input/Output) A pointer to an integer that contains the information about the size (in bytes) of the list_buf buffer. Before calling **Qp0zGetAllSysEnv()**, this parameter should be set to the size of list_buf. If the size of this parameter is large enough to receive the list, then this field will be set to the exact size of the list upon returning from **Qp0zGetAllSysEnv().** If the size of this parameter is not large enough to receive the list, then this field will contain the exact size required and ENOSPC will be the return value. In this case, the list_buf is not modified.

ccsid_buf

(Input/Output) The address of the buffer to receive the CCSIDs of the environment variables. The order of the CCSIDs returned corresponds to the order of the variables returned in the list_buf

ccsid_buf_size

(Input/Output) A pointer to an integer that contains the information about the size (in bytes) of the ccsid_buf buffer. Before calling **Qp0zGetAllSysEnv()**, this should be set to the size of ccsid_buf. If this size is enough to receive the CCSID list, then this field will contain the exact size of the CCSIDs received upon returning from **Qp0zGetAllSysEnv()**. If this size is not enough to receive the CCSID list, then this field will contain the exact size required and ENOSPC will be the return value. In this case, the *ccsid_buf* is not modified.

reserved

(Input) Reserved for future use. Currently, the only allowed value is NULL.

Return Value

0

Qp0zGetAllSysEnv() was successful. The `list_buf` contains the null-terminated system-level environment variable strings, and the ccsid_buf contains the CCSID of each variable in the same order. The list_buf_size contains the exact size of the environment variable list, and the ccsid_buf_size contains the exact size of the CCSID list.

errval

Qp0zGetAllSysEnv() was not succesful. *errval* indicates the error.

Error Conditions

If **Qp0zGetAllSysEnv()** is not successful, *errval* indicates one of the following errors.

[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.

The value for the *reserved* parameter was not NULL.

[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.

There were no system-level environment variables.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

The size of the buffers to receive the list and the CCSIDs was not enough. The *list_buf_size* and *ccsid_buf_size* parameters indicate the exact size needed for the *list_buf ccsid_buf* respectively.

[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. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- The <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

1. See the example in Qp0zPutSysEnv()--Change or Add a System-Level Environment.
2. See the two-part example in Appendix A for saving and restoring system-level environment variables.

Qp0zGetEnv()--Get Value of Environment Variable (Extended)

Syntax

```
#include <qp0z1170.h>

char *Qp0zGetEnv(const char *name, int *ccsid);
```

Threadsafe: Yes. See Usage Notes for more information.

The **Qp0zGetEnv()** function is an OS/400 extension to the standard **getenv()** function. **Qp0zGetEnv()** searches the job-level environment list for a string of the form name=value. The value and the CCSID (coded character set identifier) associated with the environment variable name are returned.

Parameters

name

(Input) The name of an environment variable.

ccsid

(Output) The CCSID for the named environment variable.

Return Value

value

Qp0zGetEnv() successfully found the environment string. The value returned is a pointer to the string containing the value for the specified name in the current environment.

NULL

Qp0zGetEnv() could not find the environment string. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zGetEnv()** is not successful, *errno* indicates one of the following errors.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

[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.

[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.

No entry found for name specified.

[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. Although **Qp0zGetEnv()** is threadsafe, if a thread calls an environment variable function while another thread is accessing an environment variable from the environ array the thread may see undefined results. The environ array can be accessed directly or by using a pointer returned from the **getenv()** or **Qp0zGetEnv()** functions. The environment contents are only protected during calls to the environment variable functions.
2. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- The <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

See the example of using **getenv()** in putenv()--Change or Add Environment Variable.

Qp0zGetSysEnv()--Get Value of System-Level Environment Variable

Syntax

```
#include <qp0z1170.h>

int Qp0zGetSysEnv(const char *name,
                  char *value, int *value_size,
                  int *ccsid, void *reserved);
```

Threadsafe: Yes

The **Qp0zGetSysEnv()** function gets the value of a system-level environment variable name by searching the system-level environment variable list for a string of the form name=value. The value and the coded character set identifier (CCSID) associated with the environment variable name are returned.

Authorities

None

Parameters

name

(Input) The name of an environment variable.

value

(Input/Output) The address of the buffer to receive the value.

value_size

(Input/Output) A pointer to an integer that contains the information about the size of the value buffer. Before calling **Qp0zGetSysEnv()**, this parameter should contain the size of the value buffer. If the size of this parameter is large enough to receive the value, then this field will contain the exact size of value upon returning from **Qp0zGetSysEnv()**. If the size of this parameter is not large enough to receive the value, then this field will contain the exact size required and ENOSPC will be the return value. In this case, the value buffer is not modified.

ccsid

(Input/Output) The address of the variable to receive the CCSID associated with this variable.

reserved

(Input) Reserved for future use. Currently, the only allowed value is NULL.

Return Value

0

Qp0zGetSysEnv() successfully found the environment string. value and ccsid contain the value and CCSID for the variable name in the system-level environment variable list.

errval

Qp0zGetEnv() was not successful. *errval* indicates the error.

Error Conditions

If **Qp0zGetSysEnv()** is not successful, *errval* indicates one of the following errors.

[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.

The value for the *reserved* parameter was not NULL.

[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.

No entry found for name specified.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

The size of the *value* buffer was not big enough to receive the value.

[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. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- The <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Example

See the example of using **Qp0zGetSysEnv()** in Qp0zPutSysEnv()--Change or Add a System-Level Environment.

Qp0zInitEnv()--Initialize Environment for Variables

Syntax

```
#include <qp0z1170.h>

int Qp0zInitEnv(void);;
```

Threadsafe: Yes

The **Qp0zInitEnv()** function sets the external variable environ to a pointer to the current environment list. (On the iSeries server, environ is initialized to NULL when an activation group is started.)

Note: Although it is possible for a user's program to directly read the environ array, use of the **getenv()** or **Qp0zGetEnv()** functions is recommended.

Parameters

None.

Return Value

0

Qp0zInitEnv() successfully initialized the environment.

-1

Qp0zInitEnv() was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zInitEnv()** is not successful, *errno* indicates the following error.

[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 <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)
- Qp0zPutSysEnv()--Change or Add a System-Level Environment

Qp0zPutEnv()--Change or Add Environment Variable (Extended)

Syntax

```
#include <qp0z1170.h>

int Qp0zPutEnv(const char *string, int ccsid);;
```

Threadsafe: Yes. See Usage Notes for more information.

The **Qp0zPutEnv()** function is an OS/400 extension to the standard **putenv()** function. **Qp0zPutEnv()** sets the value of an environment variable by altering an existing variable or creating a new variable. In addition, it specifies a CCSID (coded character set identifier) to be associated with the environment variable.

The *string* parameter points to a string of the form name=value, where name is the environment variable and value is the new value for it.

The name cannot contain a blank. For example,

```
PATH NAME=/my_lib/joe_user
```

is not valid because of the blank between PATH and NAME. The name can contain an equal (=) symbol, but the system interprets all characters following the first equal symbol as being the value of the environment variable. For example,

```
PATH=NAME=/my_lib/joe_user
```

will result in a value of 'NAME=/my_lib/joe_user' for the variable PATH.

Parameters

string

(Input) A pointer to the name=value string.

ccsid

(Input) A CCSID to be associated with this environment variable. If 0 is specified, the default CCSID for the job is used.

Return Value

0

Qp0zPutEnv() was successful.

-1

Qp0zPutEnv() was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zPutEnv()** is not successful, *errno* indicates one of the following errors.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

[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.

For example, the string may not be in the correct format.

[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. (There is a limit of 4095 environment variables per job.)

[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. Although **Qp0zPutEnv()** is threadsafe, if a thread calls an environment variable function while another thread is accessing an environment variable from the environ array the thread may see undefined results. The environ array can be accessed directly or by using a pointer returned from the **getenv()** or **Qp0zGetEnv()** functions. The environment contents are only protected during calls to the environment variable functions.
2. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- The <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutSysEnv()--Change or Add a System-Level Environment Variable

Example

See the example of using **putenv()** in putenv()--Change or Add Environment Variable.

Qp0zPutSysEnv()--Change or Add a System-Level Environment Variable

Syntax

```
#include <qp0z1170.h>

int Qp0zPutSysEnv(const char *string, int ccsid,
                  void *reserved);
```

Threadsafe: Yes

Qp0zPutSysEnv() function sets the value of a system-level environment variable by altering an existing variable or creating a new variable. In addition, it specifies a CCSID (coded character set identifier) to be associated with the environment variable.

The *string* parameter points to a string of the form name=value, where name is the environment variable and value is the new value for it.

The name cannot contain a blank. For example,

```
PATH NAME=/my_lib/joe_user
```

is not valid because of the blank between PATH and NAME. The name can contain an equal (=) symbol, but the system interprets all characters following the first equal symbol as being the value of the environment variable. For example,

```
PATH=NAME=/my_lib/joe_user
```

will result in a value of 'NAME=/my_lib/joe_user' for the variable PATH.

Parameters

string

(Input) A pointer to the name=value string.

ccsid

(Input) A CCSID to be associated with this environment variable. If 0 is specified, the default CCSID for the job is used.

reserved

(Input) Reserved for future use. Currently, the only allowed value is NULL.

Authorities

*JOBCTL special authority is required to add or change a system-level environment variable.

Return Value

0

Qp0zPutSysEnv() was successful.

errval

Qp0zPutSysEnv() was not successful. *errval* is set to indicate the error.

Error Conditions

If **Qp0zPutSysEnv()** is not successful, *errval* indicates one of the following errors.

[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.

For example, the *string* parameter was not in the correct format or the value for the *reserved* parameter was not NULL.

[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. (There is a limit of 4095 system-level environment variables.)

[EOPNOTSUPP]

Operation not supported.

The operation, though supported in general, is not supported for the requested object or the requested arguments.

This error is returned if the environment variable that is being added is QIBM_CHILD_JOB_SNDINQMSG. See **spawn()** in or **spawnp()** in for details on QIBM_CHILD_JOB_SNDINQMSG.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

You must have *JOBCTL special authority to add or change system-level environment variables.

[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. No translation is done based on the CCSID. The CCSID is just stored and retrieved as an integer value associated with each environment variable.

Related Information

- The <**qp0z1170.h**> file (see Header Files for UNIX-Type Functions)
- getenv()--Get Value of Environment Variable
- putenv()--Change or Add Environment Variable
- Qp0zDltEnv()--Delete an Environment Variable
- Qp0zDltSysEnv()--Delete a System-Level Environment Variable
- Qp0zGetAllSysEnv()--Get All System-Level Environment Variables
- Qp0zGetEnv()--Get Value of Environment Variable (Extended)
- Qp0zGetSysEnv()--Get Value of System-Level Environment Variable
- Qp0zInitEnv()--Initialize Environment for Variables
- Qp0zPutEnv()--Change or Add Environment Variable (Extended)

Example

The following example uses **Qp0zPutSysEnv()**, **Qp0zGetSysEnv()**, and **Qp0zDltSysEnv()**.

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <qp0z1170.h>

int main(int argc, char **argv)
{
 char     *var1  = "PATH=/:/home";
 char     *name1 = "PATH";
 char     *val1  = NULL;
 int      rc, ccsid, size;
```

```
 /* Add the system-level variable PATH */
 /* using default ccsid                */
 ccsid = 0;
 rc = Qp0zPutSysEnv(var1, ccsid, NULL);
 if(rc != 0)
 {
   printf("Error from Qp0zPutSysEnv while adding <%s>\n",var1);
   printf("errno = %d\n",rc);
   return rc;
 }

 printf("<%s> added to system-level env var list\n",var1);

 /* Get the value of the variable PATH */
 size = 100;
 val1 = (char *)malloc(size);

 rc = Qp0zGetSysEnv(name1, val1, &size, &ccsid, NULL);
 if(rc == ENOSPC)
 {
  /* The buffer size was not enough to get the value */
  /* Increase the buffer to size                     */
  val1 = (char *)realloc(val1, size);
  rc = Qp0zGetSysEnv(name1, val1, &size, &ccsid, NULL);
 }

 if(rc != 0)
 {
  printf("Error from Qp0zGetSysEnv while retrieving");
  printf("<%s>, errno = %d\n", name1, rc);
  return rc;
 }

 printf("<%s> retrieved, value is <%s>\n",name1,val1);

 /* Delete the PATH variable */
 rc = Qp0zDltSysEnv(name1, NULL);
 if(rc != 0)
 {
  printf("Error from Qp0zDltSysEnv while deleting");
  printf("<%s>, errno = %d\n", name1, rc);
  return rc;
 }

 printf("<%s> deleted from system-level env var list\n",name1);

 return 0;
}
```

Output:

```
 <PATH=/:/home> added to system-level variable list
 <PATH> retrieved, value is </:/home>
 <PATH> deleted from system-level variable list
```

For other examples, see the two-part example in API Examples for saving and restoring system-level environment variables.

Top | Environment Variable APIs | APIs by category

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.

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.
