IBM Systems - iSeries

UNIX-Type -- Problem Determination APIs

Version 5 Release 4

IBM Systems - iSeries

UNIX-Type -- Problem Determination 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 29.

Sixth Edition (February 2006)

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Problem Determination APIs

The problem determination APIs are:

- "Qp0zDump()—Dump Formatted Storage Trace Data" on page 2 (Dump formatted storage trace data) dumps the user storage specified by area to the user trace.
- "Qp0zDumpStack()—Dump Formatted Stack Trace Data" on page 5 (Dump formatted stack trace data) dumps a formatted representation of the call stack of the calling thread to the user trace.
- "Qp0zDumpTargetStack()—Dump Formatted Stack Trace Data of the Target Thread" on page 9 (Dump formatted stack trace data of the target thread) dumps a formatted representation of the call stack of the target thread to the user trace.
- "Qp0zLprintf()—Print Formatted Job Log Data" on page 12 (Print formatted job log data) prints user data specified by format-string as an information message type to the job log.
- "Qp0zUprintf()—Print Formatted User Trace Data" on page 15 (Print formatted user trace data) prints user data specified by the format-string parameter to the user trace.

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" on page 17 for the file and member name of each header file.

The problem determination APIs are intended to be used as an aid in debugging exception or error conditions in application programs. These functions should not be used in performance critical code.

These functions can be used during application development, as well as after the application is made available, as debug mechanisms. For example, one of the following methods could be used:

- Use a compile option that activates the problem determination functions during application development. When the application is ready to be made available, recompile to deactivate the functions.
- Design a method to (quickly) check and see whether application problem determination is desired, as well as an external method to activate application problem determination. Then, use the problem determination functions in such a manner as to check (at run time) whether or not the functions should be called.

Some of the problem determination functions dump or print to the user trace. The user trace is a permanent user space object named *QP0Z<jobnumber>* in the QUSRSYS library. The user trace is created the first time any thread in a job writes trace output. The following CL commands can be used to manipulate the user trace properties and objects:

- Change User Trace (CHGUSRTRC) can be used to change the characteristics of the user trace.
- Dump User Trace (DMPUSRTRC) can be used to dump trace records to a file or to standard output.
- Delete User Trace (DLTUSRTRC) can be used to delete the user trace objects.

For those problem determination functions that use the user trace, the following should be considered:

- The functions require no authority to the user trace object. See CL commands CHGUSRTRC, DMPUSRTRC, and DLTUSRTRC for the authority required to administer, display, or modify tracing information using the CL commands.
- No locks are held on the user trace between calls to the tracing functions. The user trace can be deleted while in use. The next function that produces trace output will create the user trace again.
- If another job on the system has the same job number as an existing user trace, the existing trace data is cleared, and the trace data from the new job replaces it.

APIs

These are the APIs for this category.

Qp0zDump()—Dump Formatted Storage Trace Data

Syntax

```
#include <qp0ztrc.h>

void Qp0zDump(const char *label,
              void       *area,
              int        len);
```

Service Program Name: QP0ZCPA
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zDump()** function dumps the user storage specified by *area* to the user trace. The user-provided storage is formatted for viewing in hexadecimal representation for up to *len* number of bytes. The formatted storage is labeled with the text string specified by *label*.

If any input parameters are not valid, or an incorrect or error condition is detected, the **Qp0zDump()** function returns immediately and no error is indicated.

An application should not use the tracing function in performance critical code. These functions are intended for debugging exception or error conditions. The user trace is a permanent user space object named *QP0Z<jobnumber>* in the QUSRSYS library. The user trace is created the first time any thread in a job writes trace output. See the Change User Trace (CHGUSRTRC), Dump User Trace (DMPUSRTRC) and Delete User Trace (DLTUSRTRC) CL commands for information about manipulating the user trace properties and objects.

Parameters

label	(Input) A pointer to a string that is used to label the storage dump.
area	(Input) A pointer to storage area that is to be formatted and dumped to the user trace.
len	(Input) The number of bytes of storage to be formatted in the user trace.

Authorities

None.

Return Value

None.

Error Conditions

If **Qp0zDump()** is not successful, the function returns immediately and no error is indicated.

Usage Notes

1. No locks are held on the user trace between calls to the tracing functions. The user trace can be deleted while in use. The next function that produces trace output will create the user trace again.

2. If another job on the system has the same job number as an existing user trace, the existing trace data is cleared, and the trace data from the new job replaces it.
3. As the format of the user trace records can change, only the following CL commands can be used to manipulate the user trace properties and objects:

   - Change User Trace (CHGUSRTRC) can be used to change the characteristics of the user trace.
   - Dump User Trace (DMPUSRTRC) can be used to dump trace records to a file or to standard output.
   - Delete User Trace (DLTUSRTRC) can be used to delete the user trace objects.

Related Information

- "Qp0zDumpStack()—Dump Formatted Stack Trace Data" on page 5—Dump Formatted Stack Trace Data
- "Qp0zDumpTargetStack()—Dump Formatted Stack Trace Data of the Target Thread" on page 9—Dump Formatted Stack Trace Data of the Target Thread
- "Qp0zLprintf()—Print Formatted Job Log Data" on page 12—Print Formatted Job Log Data
- "Qp0zUprintf()—Print Formatted User Trace Data" on page 15—Print Formatted User Trace Data

Example

See Code disclaimer information for information pertaining to code examples.

The following example uses **Qp0zDump()** and **Qp0zUprintf()** functions to produce trace output.

```
#define _MULTI_THREADED
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <qp0ztrc.h>

#define  THREADDATAMAX    128

void *theThread(void *parm)
{
   char                    *myData = parm;

   printf("Entered the %s thread\n", myData);
   Qp0zUprintf("Tracing in the %s thread\n", myData);
   Qp0zDump("The Data", myData, THREADDATAMAX);
   free(myData);
   return NULL;
}


int main(int argc, char **argv)
{
   pthread_t             thread, thread2;
   int                   rc=0;
   char                 *threadData;


   printf("Enter Testcase - %s\n", argv[0]);
   Qp0zUprintf("Tracing Testcase Entry\n");

   printf("Create two threads\n");
   Qp0zUprintf("Tracing creation of two threads\n");

   threadData = (char *)malloc(THREADDATAMAX);
   memset(threadData, 'Z', THREADDATAMAX);
   sprintf(threadData, "50%% Cotton, 50%% Polyester");
   rc = pthread_create(&thread, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }

   threadData = (char *)malloc(THREADDATAMAX);
   memset(threadData, 'Q', THREADDATAMAX);
   sprintf(threadData, "Lacquered Camel Hair");
   rc = pthread_create(&thread2, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }

   printf("Wait for threads to complete\n");
   rc = pthread_join(thread, NULL);
   if (rc) { printf("Failed pthread_join() 1\n"); exit(EXIT_FAILURE); }

   rc = pthread_join(thread2, NULL);
   if (rc) { printf("Failed pthread_join() 2\n"); exit(EXIT_FAILURE); }

   printf("Testcase complete\n");
   Qp0zUprintf("Tracing completion of the testcase rc=%d\n", rc);
   return 0;
}
```

Trace Output:

This trace output was generated after the test case was run by using the CL command **DMPUSRTRC JOB(100464/USER/TPZDUMP0) OUTPUT(*STDOUT)**. The above example program ran as job 100464/USER/TPZDUMP0.

Note the following in the trace output:

1. Each trace record is indented by several spaces to aid in readability. Trace records from different threads have different indentation levels.
2. Each trace record is identified by the hexadecimal thread ID, a colon, and a timestamp. The timestamp can be used to aid in debugging of waiting or looping threads. For example, the third trace record shown below (the Tracing Testcase Entry trace point) was created by thread 0x13, and occurred 0.870960 seconds after the last full date and time label. This means that the trace record was created on 5 January 1998 at 14:08:28.870960. A full date and time label is placed between those trace points that occur during different whole seconds.

```
User Trace Dump for job 100464/USER/TPZDUMP0. Size: 300K, Wrapped 0
times. --- 01/05/1998 14:08:28 ---
    00000013:870960 Tracing Testcase Entry
    00000013:871720 Tracing creation of two threads
     00000014:879904 Tracing in the 50% Cotton, 50% Polyester thread
     00000014:880256 C66E80F4DF:001F60 L:0080 The Data
     00000014:880968  C66E80F4DF:001F60  F5F06C40 C396A3A3 96956B40 F5F06C40     *50% Cotton, 50% *
     00000014:881680  C66E80F4DF:001F70  D79693A8 85A2A385 9900E9E9 E9E9E9E9     *Polyester.ZZZZZZ*
     00000014:882392  C66E80F4DF:001F80  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
     00000014:883096  C66E80F4DF:001F90  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
     00000014:883808  C66E80F4DF:001FA0  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
     00000014:884512  C66E80F4DF:001FB0  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
     00000014:885224  C66E80F4DF:001FC0  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
      00000015:887872 Tracing in the Lacquered Camel Hair thread
      00000015:888216 C66E80F4DF:002000 L:0080 The Data
      00000015:888952  C66E80F4DF:002000  D3818398 A4859985 8440C381 94859340     *Lacquered Camel *
      00000015:889680  C66E80F4DF:002010  C8818999 00D8D8D8 D8D8D8D8 D8D8D8D8     *Hair.QQQQQQQQQQQ*
      00000015:890416  C66E80F4DF:002020  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
      00000015:891152  C66E80F4DF:002030  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
      00000015:891888  C66E80F4DF:002040  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
      00000015:892624  C66E80F4DF:002050  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
      00000015:893352  C66E80F4DF:002060  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
      00000015:894088  C66E80F4DF:002070  D8D8D8D8 D8D8D8D8 D8D8D8D8 D8D8D8D8     *QQQQQQQQQQQQQQQQ*
     00000014:896168  C66E80F4DF:001FD0  E9E9E9E9 E9E9E9E9 E9E9E9E9 E9E9E9E9     *ZZZZZZZZZZZZZZZZ*
    00000013:898832 Tracing completion of the testcase rc=0
Press ENTER to end terminal session.
```

API introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Qp0zDumpStack()—Dump Formatted Stack Trace Data

Syntax

```
#include <qp0ztrc.h>

void Qp0zDumpStack(const char *label);
```

Service Program Name: QP0ZCPA
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zDumpStack()** function dumps a formatted representation of the call stack of the calling thread to the user trace. The formatted call stack is labeled with the text string specified by *label*. The formatted call stack shows the library, program, module, and procedure information associated with each frame on the call stack.

The formatted dump of the current call stack shows the oldest entries first, followed by newer entries.

The following example is a call stack dump if the **Qp0zDumpStack()** function is used to dump the stack of the current thread. The label *Thread dumping my own stack* was inserted by the application program using the *label* parameter.

The thread start routine in this example is **threadfunc()** in program or service program ATEST5 that resides in library QP0WTEST. The **threadfunc()** function (at statement 2) has called the function **foo()**. The function **foo()** (at statement 1), in turn has called **bar()**. The function **bar()** (at statement 1), has dumped the current call stack due to some application-specific error condition.

```
 Thread dumping my own stack
  Library    / Program     Module      Stmt    Procedure
  QSYS       / QLESPI      QLECRTTH    7     : LE_Create_Thread2
  QSYS       / QP0WPTHR    QP0WPTHR    974   : pthread_create_part2
  QP0WTEST   / ATEST5      ATEST5      2     : threadfunc
  QP0WTEST   / ATEST5      ATEST5      1     : foo
  QP0WTEST   / ATEST5      ATEST5      1     : bar
  QSYS       / QP0ZCPA     QP0ZUDBG    5     : Qp0zDumpStack
  QSYS       / QP0ZSCPA    QP0ZSCPA    199   : Qp0zSUDumpStack
  QSYS       / QP0ZSCPA    QP0ZSCPA    210   : Qp0zSUDumpTargetStack
```

An application should not use the tracing function in performance critical code. These functions are intended for debugging exception or error conditions. The user trace is a permanent user space object named *QP0Z<jobnumber>* in the QUSRSYS library. The user trace is created the first time any thread in a job writes trace output. See the Change User Trace (CHGUSRTRC), Dump User Trace (DMPUSRTRC) and Delete User Trace (DLTUSRTRC) CL commands for information about manipulating the user trace properties and objects.

Parameters

label (Input) A pointer to a string that is used to label the stack dump.

Authorities

None.

Return Value

None.

Error Conditions

If **Qp0zDumpStack()** is not successful, the function returns immediately and no error is indicated.

Usage Notes

1. No locks are held on the user trace between calls to the tracing functions. The user trace can be deleted while in use. The next function that produces trace output will create the user trace again.
2. If another job on the system has the same job number as an existing user trace, the existing trace data is cleared, and the trace data from the new job replaces it.
3. If the calling thread has more than 128 call stack entries, **Qp0zDumpStack()** returns after dumping the first 128 entries of the call stack.
4. As the format of the user trace records can change, only the following CL commands can be used to manipulate the user trace properties and objects:

   - Change User Trace (CHGUSRTRC) can be used to change the characteristics of the user trace.
   - Dump User Trace (DMPUSRTRC) can be used to dump trace records to a file or to standard output.
   - Delete User Trace (DLTUSRTRC) can be used to delete the user trace objects.

Related Information

- "Qp0zDump()—Dump Formatted Storage Trace Data" on page 2—Dump Formatted Storage Trace Data
- "Qp0zDumpStack()—Dump Formatted Stack Trace Data" on page 5—Dump Formatted Stack Trace Data of the Target Thread
- "Qp0zLprintf()—Print Formatted Job Log Data" on page 12—Print Formatted Job Log Data
- "Qp0zUprintf()—Print Formatted User Trace Data" on page 15)—Print Formatted User Trace Data

Example

See Code disclaimer information for information pertaining to code examples.

The following example uses **Qp0zDumpStack()** and **Qp0zUprintf()** functions to produce trace output.

```
#define _MULTI_THREADED
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <qp0ztrc.h>

#define  THREADDATAMAX    128

void foo(char *string);
void bar(char *string);

void *theThread(void *parm)
{
   char                    *myData = parm;

   printf("Entered the %s thread\n", myData);
   foo(myData);
   free(myData);
   return NULL;
}

void foo(char *string)
{
  bar(string);
}

void bar(char *string)
{
  Qp0zUprintf("function bar(): Hit an error condition!\n");
  Qp0zDumpStack(string);
}

int main(int argc, char **argv)
{
   pthread_t             thread, thread2;
   int                   rc=0;
   char                 *threadData;


   printf("Enter Testcase - %s\n", argv[0]);
   Qp0zUprintf("Tracing Testcase Entry\n");

   printf("Create two threads\n");
   Qp0zUprintf("Tracing creation of two threads\n");

   threadData = (char *)malloc(THREADDATAMAX);
   sprintf(threadData, "50%% Cotton, 50%% Polyester");
   rc = pthread_create(&thread, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }

   threadData = (char *)malloc(THREADDATAMAX);
   sprintf(threadData, "Lacquered Camel Hair");
   rc = pthread_create(&thread2, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }
   printf("Wait for threads to complete\n");
   rc = pthread_join(thread, NULL);
   if (rc) { printf("Failed pthread_join() 1\n"); exit(EXIT_FAILURE); }

   rc = pthread_join(thread2, NULL);
   if (rc) { printf("Failed pthread_join() 2\n"); exit(EXIT_FAILURE); }

   printf("Testcase complete\n");
   Qp0zUprintf("Tracing completion of the testcase rc=%d\n", rc);
   return 0;
}
```

Trace Output:

This trace output was generated after the test case was run by using the CL command **DMPUSRTRC JOB(100465/USER/TPZSTK0) OUTPUT(*STDOUT)**. The above example program ran as job 100465/USER/TPZSTK0.

Note the following in the trace output:

1. Each trace record is indented by several spaces to aid in readability. Trace records from different threads have different indentation levels.
2. Each trace record is identified by the hexadecimal thread ID, a colon, and a timestamp. The timestamp can be used to aid in debugging of waiting or looping threads. For example, the third trace record shown below, (the Tracing Testcase Entry trace point) was created by thread 0x16, and occurred 0.841456 seconds after the last full date and time label. This means that the trace record was created on 5 January 1998 at 16:32:23.841456. A full date and time label is placed between those trace points that occur during different whole seconds.

```
User Trace Dump for job 100465/USER/TPZSTK0. Size: 300K, Wrapped 0 times.
--- 01/05/1998 16:32:23 ---
 00000016:841456 Tracing Testcase Entry
 00000016:842176 Tracing creation of two threads
 00000017:850328 function bar(): Hit an error condition!
 00000017:850552 Stack Dump For Current Thread
 00000017:850752 Stack:  50% Cotton, 50% Polyester
00000018:853288 function bar(): Hit an error condition!
00000018:853512 Stack Dump For Current Thread
00000018:853712 Stack:  Lacquered Camel Hair
00000018:888752 Stack:  Library    / Program     Module      Stmt    Procedure
 00000017:889400 Stack:  Library    / Program     Module      Stmt    Procedure
 00000017:904848 Stack:  QSYS       / QLESPI      QLECRTTH    774   : LE_Create_Thread2__FP12crtth_parm_t
 00000017:905088 Stack:  QSYS       / QP0WPTHR    QP0WPTHR    1004  : pthread_create_part2
 00000017:905312 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     2     : theThread
 00000017:905528 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     1     : foo
 00000017:905744 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     2     : bar
 00000017:905960 Stack:  QSYS       / QP0ZCPA     QP0ZUDBG    85    : Qp0zDumpStack
 00000017:906184 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    274   : Qp0zSUDumpStack
 00000017:906408 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    285   : Qp0zSUDumpTargetStack
 00000017:906536 Stack:  Completed
00000018:908504 Stack:  QSYS       / QLESPI      QLECRTTH    774   : LE_Create_Thread2__FP12crtth_parm_t
00000018:908744 Stack:  QSYS       / QP0WPTHR    QP0WPTHR    1004  : pthread_create_part2
00000018:908960 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     2     : theThread
00000018:909168 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     1     : foo
00000018:909384 Stack:  QP0WTEST   / TPZSTK0     TPZSTK0     2     : bar
00000018:909592 Stack:  QSYS       / QP0ZCPA     QP0ZUDBG    85    : Qp0zDumpStack
00000018:909816 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    274   : Qp0zSUDumpStack
00000018:910032 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    285   : Qp0zSUDumpTargetStack
00000018:910168 Stack:  Completed
 00000016:912792 Tracing completion of the testcase rc=0
Press ENTER to end terminal session.
```

API introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Qp0zDumpTargetStack()—Dump Formatted Stack Trace Data of the Target Thread

Syntax

```
#include <qp0ztrc.h>

int Qp0zDumpTargetStack(int handle,
                        const char *label);
```

Service Program Name: QP0ZCPA
Default Public Authority: *USE
Threadsafe: Conditional; see "Usage Notes" on page 10.

The **Qp0zDumpTargetStack()** function dumps a formatted representation of the call stack of the target thread to the user trace. The target thread is specified by *handle*, which can be filled in using the *pthread_t* structure. The formatted call stack is labeled with the text string specified by *label*. The formatted call stack shows the library, program, module, and procedure information associated with each frame on the call stack at the time the function was called.

The formatted dump of the target call stack shows the oldest entries first, followed by newer entries.

For consistent results, ensure that the target thread specified in the *handle* parameter is blocked or waiting for some resource and not actively running.

If a target thread that is actively running is specified, the stack trace information may be inconsistent.

An application should not use the tracing function in performance critical code. These functions are intended for debugging exception or error conditions. The user trace is a permanent user space object named *QP0Z<jobnumber>* in the QUSRSYS library. The user trace is created the first time any thread in a job writes trace output. See the Change User Trace (CHGUSRTRC), Dump User Trace (DMPUSRTRC) and Delete User Trace (DLTUSRTRC) CL commands for information about manipulating the user trace properties and objects.

Parameters

handle	(Input) A handle to the target thread.
label	(Input) A pointer to a string that is used to label the stack dump.

Authorities

None.

Return Value

0	**Qp0zDumpTargetStack()** was successful.
value	**Qp0zDumpTargetStack()** was not successful. The value returned is an errno indicating the failure.

Error Conditions

If **Qp0zDumpTargetStack()** 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.

[ESRCH]

No item could be found that matches the specified value.

Usage Notes

1. No locks are held on the user trace between calls to the tracing functions. The user trace can be deleted while in use. The next function that produces trace output will create the user trace again.
2. If another job on the system has the same job number as an existing user trace, the existing trace data is cleared, and the trace data from the new job replaces it.
3. The **Qp0zDumpTargetStack()** can only safely be used against a thread that is stopped or waiting for some activity to occur. If **Qp0zDumpTargetStack()** is used with a thread that is actively running, the output stack trace may show an inconsistent view of the call stack.
4. If the target thread has more than 128 call stack entries, **Qp0zDumpTargetStack()** returns after dumping the first 128 entries of the call stack.
5. As the format of the user trace records can change, only the following CL commands can be used to manipulate the user trace properties and objects:
   - Change User Trace (CHGUSRTRC) can be used to change the characteristics of the user trace.
   - Dump User Trace (DMPUSRTRC) can be used to dump trace records to a file or to standard output.
   - Delete User Trace (DLTUSRTRC) can be used to delete the user trace objects.

Related Information

- "Qp0zDump()—Dump Formatted Storage Trace Data" on page 2—Dump Formatted Storage Trace Data
- "Qp0zDumpStack()—Dump Formatted Stack Trace Data" on page 5—Dump Formatted Stack Trace Data
- "Qp0zLprintf()—Print Formatted Job Log Data" on page 12—Print Formatted Job Log Data
- "Qp0zUprintf()—Print Formatted User Trace Data" on page 15—Print Formatted User Trace Data

Example

See Code disclaimer information for information pertaining to code examples.

The following example uses **Qp0zDumpTargetStack()** and **Qp0zUprintf()** functions to produce trace output.

```
#define _MULTI_THREADED
#include <pthread.h>
#include <milib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <qp0ztrc.h>
```

```
void *threadfunc(void *);

int main(int argc, char **argv)
{
  int                   rc=0;
  pthread_t             thread;

  Qp0zUprintf("Entering Testcase\n");
  rc = pthread_create(&thread, NULL, threadfunc, NULL);
  sleep(2);  /* Let the thread block */

  /* If the other thread ends or is actively running (that is       */
  /* changing the call stack, you may get meaningless results in the*/
  /* stack dump for the target thread.)                             */
  Qp0zDumpTargetStack(thread.reservedHandle,
                      "Dumping target thread's stack\n");
  Qp0zUprintf("Exit with return code of 0\n");
  return 0;
}


void foo(void);
void bar(void);
void *threadfunc(void *parm)
{
  Qp0zUprintf("Inside secondary thread\n");
  foo();
  return NULL;
}

void foo(void)
{
  bar();
}

void bar(void)
{
  Qp0zDumpStack("Thread dumping my own stack\n");
  sleep(10); /* Ensure the thread is blocked */
}
```

Trace Output:

This trace output was generated after the test case was run by using the CL command **DMPUSRTRC JOB(107141/USER/TPZTSTK0) OUTPUT(*STDOUT)**. The above example program ran as job 107141/USER/TPZTSTK0.

Note the following in the trace output:

1. Each trace record is indented by several spaces to aid in readability. Trace records from different threads have different indentation levels.
2. Each trace record is identified by the hexadecimal thread ID, a colon, and a timestamp. The timestamp can be used to aid in debugging of waiting or looping threads. For example, the third trace record shown below, (the Entering Testcase trace point) was created by thread 0x36, and occurred 0.595584 seconds after the last full date and time label. This means that the trace record was created on 23 January 1998 at 12:38:10.595584. A full date and time label is placed between those trace points that occur during different whole seconds.

```
User Trace Dump for job 107141/USER/TPZTSTK0. Size: 300K, Wrapped 0 times.
--- 01/23/1998 12:38:10 ---
  00000036:595584 Entering Testcase
  00000037:598832 Inside secondary thread
  00000037:599024 Stack Dump For Current Thread
  00000037:599200 Stack:  Thread dumping my own stack
  00000037:695440 Stack:  Library    / Program     Module      Stmt    Procedure
  00000037:752984 Stack:  QSYS       / QLESPI      QLECRTTH    774   : LE_Create_Thread2__FP12crtth_parm_t
  00000037:805664 Stack:  QSYS       / QP0WPTHR    QP0WPTHR    1006  : pthread_create_part2
  00000037:805888 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    2     : threadfunc
  00000037:806088 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    1     : foo
  00000037:806288 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    1     : bar
  00000037:806496 Stack:  QSYS       / QP0ZCPA     QP0ZUDBG    85    : Qp0zDumpStack
  00000037:848280 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    274   : Qp0zSUDumpStack
  00000037:848504 Stack:  QSYS       / QP0ZSCPA    QP0ZSCPA    285   : Qp0zSUDumpTargetStack
  00000037:848616 Stack:  Completed
 --- 01/23/1998 12:38:12 ---
  00000036:628272 Stack Dump For Target Thread: Handle 7 (0x00000007)
  00000036:628464 Stack:  Dumping target thread's stack
  00000036:651608 Stack:  Library    / Program     Module      Stmt    Procedure
  00000036:651872 Stack:  QSYS       / QLESPI      QLECRTTH    774   : LE_Create_Thread2__FP12crtth_parm_t
  00000036:652088 Stack:  QSYS       / QP0WPTHR    QP0WPTHR    1006  : pthread_create_part2
  00000036:652304 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    2     : threadfunc
  00000036:652512 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    1     : foo
  00000036:652712 Stack:  QP0WTEST   / TPZTSTK0    TPZTSTK0    2     : bar
  00000036:677456 Stack:  QSYS       / QP0SSRV1    QP0SLIB     1061  : sleep
  00000036:700096 Stack:  QSYS       / QP0SSRV2    QP0SWAIT    248   : qp0swait__FP13qp0ssigwait_t
  00000036:700216 Stack:  Completed
  00000036:700408 Exit with return code of 0
Press ENTER to end terminal session.
```

API introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Qp0zLprintf()—Print Formatted Job Log Data

Syntax

```
#include <qp0ztrc.h>

int Qp0zLprintf(char *format-string, ...);
```

Service Program Name: QP0ZCPA
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zLprintf()** function prints user data specified by *format-string* as an information message type to the job log.

If a second parameter, *argument-list*, is provided, **Qp0zLprintf()** converts each entry in the *argument-list* and writes the entry to the job log according to the corresponding format specification in *format-string*. If there are more entries in *argument-list* than format specifications in *format-string*, the extra *argument-list* entries are evaluated and then ignored. If there are less entries in *argument-list* than format specifications in *format-string*, the job log output for those entries is undefined, and the **Qp0zLprintf()** function may return an error.

The data printed by **Qp0zLprintf()** is buffered one line at a time, and a new message in the job log is forced every 512 characters if a new line (\n) is not detected in the data before that time. The buffer used by **Qp0zLprintf()** is not physically written when the application ends. To ensure messages are written to the job log, always use a new line (\n) at the end of each *format-string*.

An application should not use the tracing function in performance critical code. These functions are intended for debugging exceptions or error conditions.

Parameters

format-string
: (Input) The format string representing the format of the data to be printed. See the **printf()** function in ILE C/C++ Run-Time Library Functions for a description of valid format strings.

... (argument-list)
: (Input) An optional list of arguments that contain entries to be formatted and printed to the job log.

Authorities

None.

Return Value

value	**Qp0zLprintf()** was successful. The value returned is the number of characters successfully printed.
-1	**Qp0zLprintf()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zLprintf()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than that listed here.

[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. An invalid *format-string* or *argument-list* was specified.

[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.

Usage Notes

None.

Related Information

- "Qp0zDump()—Dump Formatted Storage Trace Data" on page 2—Dump Formatted Storage Trace Data
- "Qp0zDumpStack()—Dump Formatted Stack Trace Data" on page 5—Dump Formatted Stack Trace Data
- "Qp0zDumpTargetStack()—Dump Formatted Stack Trace Data of the Target Thread" on page 9—Dump Formatted Stack Trace Data of the Target Thread
- "Qp0zUprintf()—Print Formatted User Trace Data" on page 15—Print Formatted User Trace Data

Example

See Code disclaimer information for information pertaining to code examples.

The following example uses **Qp0zLprintf()** to produce output in the job log.

```
#define _MULTI_THREADED
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <qp0ztrc.h>

#define  THREADDATAMAX    128

void *theThread(void *parm)
{
   char                    *myData = parm;

   Qp0zLprintf("%.8x %.8x: Entered the %s thread\n",
               pthread_getthreadid_np(), myData);
   free(myData);
   return NULL;
}

int main(int argc, char **argv)
{
   pthread_t             thread, thread2;
   int                   rc=0;
   char                 *threadData;


   printf("Create two threads\n");
   Qp0zUprintf("Tracing creation of two threads\n");

   threadData = (char *)malloc(THREADDATAMAX);
   sprintf(threadData, "50%% Cotton, 50%% Polyester");
   rc = pthread_create(&thread, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }

   threadData = (char *)malloc(THREADDATAMAX);
   sprintf(threadData, "Lacquered Camel Hair");
   rc = pthread_create(&thread2, NULL, theThread, threadData);
   if (rc) {
     printf("Failed to create a %s thread\n", threadData);
     exit(EXIT_FAILURE);
   }

   printf("Wait for threads to complete\n");
   rc = pthread_join(thread, NULL);
   if (rc) { printf("Failed pthread_join() 1\n"); exit(EXIT_FAILURE); }

   rc = pthread_join(thread2, NULL);
   if (rc) { printf("Failed pthread_join() 2\n"); exit(EXIT_FAILURE); }
   return 0;
}
```

Job Log Output:

The following two job log messages where generated by the example shown above. The output was retrieved from the spooled file created when the job ran to completion and when the job log was retained. The informational messages contain the contents of the **Qp0zLprintf()** function calls.

br>

```
*NONE    Information    01/05/98   16:55:05   QP0ZCPA      QSYS    *STMT   QP0ZCPA    QSYS     *STMT
                         From module . . . . . . . . :   QP0ZUDBG
                         From procedure  . . . . . . :   Qp0zVLprintf
                         Statement . . . . . . . . . :   296
                         To module . . . . . . . . . :   QP0ZUDBG
                         To procedure  . . . . . . . :   Qp0zVLprintf
                         Statement . . . . . . . . . :   296
                         Thread  . . . . : 0000001A
                         Message . . . . : 00000000 0000001a: Entered the 50% Cotton, 50% Polyester thread
*NONE   Information   01/05/98   16:55:05   QP0ZCPA     QSYS    *STMT   QP0ZCPA    QSYS     *STMT
                         From module . . . . . . . . :   QP0ZUDBG
                         From procedure  . . . . . . :   Qp0zVLprintf
                         Statement . . . . . . . . . :   296
                         To module . . . . . . . . . :   QP0ZUDBG
                         To procedure  . . . . . . . :   Qp0zVLprintf
                         Statement . . . . . . . . . :   296
                         Thread  . . . . : 0000001B
                         Message . . . . : 00000000 0000001b: Entered the Lacquered Camel Hair thread
```

API introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Qp0zUprintf()—Print Formatted User Trace Data

Syntax

```
#include <qp0ztrc.h>

int Qp0zUprintf(char *format-string, ...);
```

Service Program Name: QP0ZCPA
Default Public Authority: *USE
Threadsafe: Yes

The **Qp0zUprintf()** function prints user data specified by the *format-string* parameter to the user trace.

If a second parameter, *argument-list*, is provided, **Qp0zUprintf()** converts each entry in the *argument-list* and writes the entry to the user trace according to the corresponding format specification in *format-string*. If there are more entries in *argument-list* than format specifications in *format-string*, the extra *argument-list* entries are evaluated and then ignored. If there are less entries in *argument-list* than format specifications in *format-string*, the user trace output for those entries is undefined, and the **Qp0zUprintf()** function may return an error.

An application should not use the tracing function in performance critical code. These functions are intended for debugging exception or error conditions. The user trace is a permanent user space object named *QP0Z<jobnumber>* in the QUSRSYS library. The user trace is created the first time any thread in a job writes trace output. See the Change User Trace (CHGUSRTRC), Dump User Trace (DMPUSRTRC) and Delete User Trace (DLTUSRTRC) CL commands for information about manipulating the user trace properties and objects.

Parameters

format-string
(Input) The format string representing the format of the data to be printed. See the **printf()** function in the ILE C/C++ Programmer's Guide for a description of valid format strings.

... (argument-list)
(Input) An optional list of arguments that contain entries to be formatted and printed to the user trace.

Authorities

None.

Return Value

value	**Qp0zUprintf()** was successful. The value returned is the number of characters successfully printed.
-1	**Qp0zUprintf()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **Qp0zUprintf()** is not successful, *errno* indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[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. An invalid *format-string* or *argument-list* was specified.

[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.

Usage Notes

1. No locks are held on the user trace between calls to the tracing functions. The user trace can be deleted while in use. The next function that produces trace output will create the user trace again.
2. If another job on the system has the same job number as an existing user trace, the existing trace data is cleared, and the trace data from the new job replaces it.
3. As the format of the user trace records can change, only the following CL commands can be used to manipulate the user trace properties and objects:
   - Change User Trace (CHGUSRTRC) can be used to change the characteristics of the user trace.
   - Dump User Trace (DMPUSRTRC) can be used to dump trace records to a file or to standard output.
   - Delete User Trace (DLTUSRTRC) can be used to delete the user trace objects.

Related Information

- “Qp0zDump()—Dump Formatted Storage Trace Data” on page 2—Dump Formatted Storage Trace Data
- “Qp0zDumpStack()—Dump Formatted Stack Trace Data” on page 5—Dump Formatted Stack Trace Data
- “Qp0zDumpTargetStack()—Dump Formatted Stack Trace Data of the Target Thread” on page 9—Dump Formatted Stack Trace Data of the Target Thread
- “Qp0zLprintf()—Print Formatted Job Log Data” on page 12—Print Formatted Job Log Data

Example

See Code disclaimer information for information pertaining to code examples.

See "Qp0zDump()—Dump Formatted Storage Trace Data" on page 2—Dump Formatted Storage Trace Data.

API introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Concepts

These are the concepts for this category.

Header Files for UNIX-Type Functions

Programs using the UNIX[(R)]-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 Include files 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

Name of Header File	Name of File in QSYSINC	Name of Member
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
qp0lchsg.h	H	QP0LCHSG
qp0lflop.h	H	QP0LFLOP
qp0ljrnl.h	H	QP0LJRNL
qp0lror.h	H	QP0LROR
qp0lrro.h	H	QP0LRRO
qp0lrtsg.h	H	QP0LRTSG
qp0lscan.h	H	QP0LSCAN
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

Name of Header File	Name of File in QSYSINC	Name of Member
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(R)-type functions may receive error information as *errno* values. The possible values returned are listed here in ascending *errno* value sequence.

Name	Value	Text	Details
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.	You attempted to do an operation that required the file to be open.
ENOTREAD	3005	File is not opened for read operations.	You tried to read a file that is not open for read operations.
EIO	3006	Input/output error.	» A physical I/O error occurred or a referenced object was damaged. «
ENODEV	3007	No such device.	
ERECIO	3008	Cannot get single character for files opened for record I/O.	The file that was specified is open for record I/O and you attempted to read it as a stream file.
ENOTWRITE	3009	File is not opened for write operations.	You tried to update a file that has not been 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.	The mode that you attempted to open the file in is not correct.
EBADPOS	3017	The position specifier is not correct.	
ENOPOS	3018	There is no record at the specified position.	You attempted to position to a record that does not exist in the file.
ENUMMBRS	3019	Attempted to use ftell on multiple members.	Remove all but one member from the file.
ENUMRECS	3020	The current record position is too long for ftell.	
EINVAL	3021	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.
EBADFUNC	3022	Function parameter in the signal function is not set.	

Name	Value	Text	Details
ENOENT	3025	No such path or directory.	The directory or a component of the path name specified does not exist.
ENOREC	3026	Record is not found.	
EPERM	3027	The operation is not permitted.	You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.
EBADDATA	3028	Message data is not valid.	The message data that was specified for the error text is not correct.
EBUSY	3029	Resource busy.	An attempt was made to use a system resource that is not available at this time.
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.	The length of the record is longer than the buffer size that was specified. The data written was padded to the length of the record.
EBADKEYLN	3044	A length that was not valid was specified for the key.	You attempted a record I/O against a keyed file. The key length that was specified is not correct.
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.	An attempt was made to access an object in a way forbidden by its object access permissions.
ENOTDIR	3403	Not a directory.	A component of the specified path name existed, but it was not a directory when a directory was expected.
ENOSPC	3404	No space is 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.
EXDEV	3405	Improper link.	A link to a file on another file system was attempted.
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.	

Name	Value	Text	Details
EFAULT	3408	The address used for an argument was not correct.	In attempting to use an argument in a call, the system detected an address that is not valid.
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.	The operation, though supported in general, is not supported for the requested object or the requested arguments.

Name	Value	Text	Details
EOPNOTSUPP	3440	Operation is not supported.	The operation, though supported in general, is not supported for the requested object or the requested arguments.
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.	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.
EMFILE	3452	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.
ENFILE	3453	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.
EPIPE	3455	Broken pipe.	
ECANCEL	3456	Operation cancelled.	
EEXIST	3457	Object exists.	The object specified already exists and the specified operation requires that it not exist.
EDEADLK	3459	Resource deadlock avoided.	An attempt was made to lock a system resource that would have resulted in a deadlock situation. The lock was not obtained.
ENOMEM	3460	Storage allocation request failed.	A function needed to allocate storage, but no storage is available.
EOWNERTERM	3462	The synchronization object no longer exists because the owner is no longer running.	The process that had locked the mutex is no longer running, so the mutex was deleted.
EDESTROYED	3463	The synchronization object was destroyed, or the object no longer exists.	

Name	Value	Text	Details
ETERM	3464	Operation was terminated.	
ENOENT1	3465	No such file or directory.	A component of a specified path name did not exist, or the path name was an empty string.
ENOEQFLOG	3466	Object is already linked to a dead directory.	The link as a dead option was specified, but the object is already marked as dead. Only one dead link is allowed for an object.
EEMPTYDIR	3467	Directory is empty.	A directory with entries of only dot and dot-dot was supplied when a nonempty directory was expected.
EMLINK	3468	Maximum link count for a file was exceeded.	An attempt was made to have the link count of a single file exceed LINK_MAX. The value of LINK_MAX can be determined using the pathconf() or the fpathconf() function.
ESPIPE	3469	Seek request is not supported for object.	A seek request was specified for an object that does not support seeking.
ENOSYS	3470	Function not implemented.	An attempt was made to use a function that is not available in this implementation for any object or any arguments.
EISDIR	3471	Specified target is a directory.	The path specified named a directory where a file or object name was expected.
EROFS	3472	Read-only file system.	You have attempted an update operation in a file system that only supports read operations.
EUNKNOWN	3474	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.
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.	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.

Name	Value	Text	Details
ENAMETOOLONG	3486	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.
ENOLCK	3487	No locks are available.	A system-imposed limit on the number of simultaneous file and record locks was reached, and no more were available at that time.
ENOTEMPTY	3488	Directory is not empty.	You tried to remove a directory that is not empty. A directory cannot contain objects when it is being removed.
ENOSYSRSC	3489	System resources are not available.	
ECONVERT	3490	Conversion error.	One or more characters could not be converted from the source CCSID to the target CCSID.
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.	The type of the object referenced by a descriptor does not match the type specified on the interface.
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.	You attempted to use a function that requires you to be enrolled in the system distribution directory and you are not.
EOFFLINE	3499	Object is suspended.	You have attempted to use an object that has had its data saved and the storage associated with it freed. An attempt to retrieve the object's data failed. The object's data cannot be used until it is successfully restored. The object's data was saved and freed either by saving the object with the STG(*FREE) parameter, or by calling an API.
EROOBJ	3500	Object is read-only.	You have attempted to update an object that can be read only.
EEAHDDSI	3501	Hard damage on extended attribute data space index.	

Name	Value	Text	Details
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.	The read or write of an area conflicts with a lock held by another process.
EFBIG	3507	Object too large.	The size of the object would exceed the system allowed maximum size.
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.	» To recover from this error, run the Reclaim Storage (RCLSTG) command as soon as possible. «
EBADFID	3512	A file ID could not be assigned when linking an object to a directory.	The file ID table is missing or damaged. » To recover from this error, run the Reclaim Storage (RCLSTG) command as soon as possible. «
ESTALE	3513	File or object handle rejected by server.	
ESRCH	3515	No such process.	
ENOTSIGINIT	3516	Process is not enabled for signals.	An attempt was made to call a signal function under one of the following conditions: • The signal function is being called for a process that is not enabled for asynchronous signals. • The signal function is being called when the system signal controls have not been initialized.
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.	Function is not allowed in a job that is running with multiple threads.
EOVERFLOW	3525	Object is too large to process.	The object's data size exceeds the limit allowed by this function.

Name	Value	Text	Details
EJRNDAMAGE	3526	Journal is damaged.	A journal or all of the journal's attached journal receivers are damaged, or the journal sequence number has exceeded the maximum value allowed. This error occurs during operations that were attempting to send an entry to the journal.
EJRNINACTIVE	3527	Journal is inactive.	The journaling state for the journal is *INACTIVE. This error occurs during operations that were attempting to send an entry to the journal.
EJRNRCVSPC	3528	Journal space or system storage error.	The attached journal receiver does not have space for the entry because the storage limit has been exceeded for the system, the object, the user profile, or the group profile. This error occurs during operations that were attempting to send an entry to the journal.
EJRNRMT	3529	Journal is remote.	The journal is a remote journal. Journal entries cannot be sent to a remote journal. This error occurs during operations that were attempting to send an entry to the journal.
ENEWJRNRCV	3530	New journal receiver is needed.	A new journal receiver must be attached to the journal before entries can be journaled. This error occurs during operations that were attempting to send an entry to the journal.
ENEWJRN	3531	New journal is needed.	The journal was not completely created, or an attempt to delete it did not complete successfully. This error occurs during operations that were attempting to start or end journaling, or were attempting to send an entry to the journal.
EJOURNALED	3532	Object already journaled.	A start journaling operation was attempted on an object that is already being journaled.
EJRNENTTOOLONG	3533	Entry is too large to send.	The journal entry generated by this operation is too large to send to the journal.
EDATALINK	3534	Object is a datalink object.	
ENOTAVAIL	3535	Independent Auxiliary Storage Pool (ASP) is not available.	The independent ASP is in Vary Configuration (VRYCFG) or Reclaim Storage (RCLSTG) processing. To recover from this error, wait until processing has completed for the independent ASP.
ENOTTY	3536	I/O control operation is not appropriate.	
EFBIG2	3540	Attempt to write or truncate file past its sort file size limit.	

Name	Value	Text	Details
ETXTBSY	3543	Text file busy.	» An attempt was made to execute an i5/OS PASE program that is currently open for writing, or an attempt has been made to open for writing an i5/OS PASE program that is being executed. «
EASPGRPNOTSET	3544	ASP group not set for thread.	
ERESTART	3545	A system call was interrupted and may be restarted.	
ESCANFAILURE	3546	Object had scan failure.	An object has been marked as a scan failure due to processing by an exit program associated with the scan-related integrated file system exit points.

Top | UNIX-Type APIs | APIs by category

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:

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

Terms and Conditions