

OS/400 PASE APIs (V5R2)

Table of Contents

- [OS/400 PASE APIs](#)
- APIs
 - [Callable Program APIs](#)
 - [Run an OS/400 PASE Shell Program](#) (QP2SHELL and QP2SHELL2)
 - [Run an OS/400 PASE Terminal Session](#) (QP2TERM)
 - [ILE Procedure APIs](#)
 - [Allocate OS/400 PASE Heap Memory](#) (Qp2malloc)◀◀
 - [Call an OS/400 PASE Procedure](#) (Qp2CallPase and Qp2CallPase2)
 - [Close a Dynamically Loaded OS/400 PASE Module](#) (Op2dlclose)◀◀
 - [Dynamically Load an OS/400 PASE Module](#) (Qp2dlopen)◀◀
 - [End an OS/400 PASE Program](#) (Qp2EndPase)◀◀
 - [Find an Exported OS/400 PASE Symbol](#) (Qp2dlsym)◀◀
 - [Free OS/400 PASE Heap Memory](#) (Qp2free)◀◀
 - [Post an OS/400 PASE Signal](#) (Qp2SignalPase)
 - [Retrieve Job CCSID for OS/400 PASE](#) (Qp2jobCCSID)◀◀
 - [Retrieve OS/400 PASE CCSID](#) (Qp2paseCCSID())◀◀
 - [Retrieve OS/400 PASE Dynamic Load Error Information](#) (Qp2dlerror)◀◀
 - [Retrieve OS/400 PASE errno Pointer](#) (Qp2errnop)◀◀
 - [Retrieve OS/400 PASE Pointer Size](#) (Qp2ptrsize)◀◀
 - [Run an OS/400 PASE Program](#) (Qp2RunPase)
 - [Runtime Functions for Use by OS/400 PASE Programs](#)
 - [Build an ILE Argument List for OS/400 PASE](#) (build_ILEarglist)
 - [Call an ILE Procedure for OS/400 PASE](#) (_ILECALL and _ILECALLX)
 - [Call an OS/400 Program for OS/400 PASE](#) (_PGMCALL)◀◀
 - [Compute ILE Argument List Size for OS/400 PASE](#) (size_ILEarglist)
 - [Convert ILE errno to OS/400 PASE errno](#) (_CVTERRNO)
 - [Convert Space Pointer for OS/400 PASE](#) (_CVTSPP)
 - [Copy Character String for OS/400 PASE](#) (_STRNCPY_SPP)
 - [Copy Memory With Tags for OS/400 PASE](#) (_MEMCPY_WT and _MEMCPY_WT2)
 - [Determine Character String Length for OS/400 PASE](#) (_STRLEN_SPP)

- [Find Exported ILE Symbol for OS/400 PASE](#) (`_ILESYM`)
- [Load an ILE Bound Program for OS/400 PASE](#) (`_ILELOAD`)
- [Override SQL CLI CCSID for OS/400 PASE](#) (`SQLOverrideCCSID400`)
- [Receive Nonprogram Message for OS/400 PASE](#) (`QMHRCVM` and `QMHRCVM1`)
- [Receive Program Message for OS/400 PASE](#) (`QMHRCVPM`, `QMHRCVPM1`, and `QMHRCVPM2`)
- [Resolve to an OS/400 Object for OS/400 PASE](#) (`_RSLOBJ`)
- [Retrieve Job CCSID for OS/400 PASE](#) (`Qp2jobCCSID`)
- [Retrieve OS/400 PASE CCSID](#) (`Qp2paseCCSID()`)
- [Return without Exiting OS/400 PASE](#) (`_RETURN`)
- [Run a CL Command for OS/400 PASE](#) (`systemCL`)
- [Send Nonprogram Message for OS/400 PASE](#) (`QMHSNDM` and `QMHSNDM1`)
- [Send Program Message for OS/400 PASE](#) (`QMHSNDPM`, `QMHSNDPM1`, and `QMHSNDPM2`)
- [Set OS/400 PASE CCSID](#) (`_SETCCSID`)
- [Set Space Pointer for OS/400 PASE](#) (`_SETSPP`)

- Related topics

- [OS/400 PASE Runtime Libraries](#)
- [OS/400 PASE Locales](#)
- [OS/400 PASE Environment Variables](#)
- [OS/400 PASE Signal Handling](#)

OS/400 PASE APIs

Portable Application Solutions Environment (OS/400 PASE) is an integrated runtime environment for AIX^(R) applications. OS/400 PASE supports the same binary executable format as AIX for PowerPC^(R) and a large subset of AIX runtime that allows many AIX applications to run with little or no change.

OS/400 PASE supports direct hardware execution of PowerPC instructions (not an emulator), while providing access to the same OS/400 support used by ILE applications for file systems, sockets, security, and many other system services.

An OS/400 PASE program can be stored in any bytestream file in the OS/400 Integrated File System because it is simply a binary file. OS/400 PASE programs can be created by any compiler and linker that produce executables compatible with AIX for PowerPC.

You must call a system API to run an OS/400 PASE program. The system provides both callable program APIs and ILE procedure APIs to run OS/400 PASE programs. The callable program APIs can be easier to use, but do not offer all the controls available with the ILE procedure APIs.

The functions available to you through OS/400 PASE are:

- [Callable Program APIs](#)
- [ILE Procedure APIs](#)
- [Runtime Functions for Use by OS/400 PASE Programs](#)

See also:

- [OS/400 PASE](#) for information about creating OS/400 PASE programs.
- [OS/400 PASE Runtime Libraries](#) for information about OS/400 PASE interfaces that are also supported on AIX.
- [OS/400 PASE Locales](#) for information about OS/400 PASE locales.
- [OS/400 PASE Environment Variables](#) for information about OS/400 PASE environment variables.
- [OS/400 PASE Signal Handling](#) for information about OS/400 PASE signals and how they relate to OS/400 exception messages.

OS/400 PASE Callable Program APIs

The callable program APIs run an OS/400 PASE program. They are:

- [Run an OS/400 PASE Shell Program](#) (QP2SHELL and QP2SHELL2) runs an OS/400 PASE program in the job that calls the API.
- [Run an OS/400 PASE Terminal Session](#) (QP2TERM) runs an interactive terminal session that communicates with an OS/400 PASE program (defaulting to the Korn shell) running in a batch job.

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

QP2SHELL() and QP2SHELL2()--Run an OS/400 PASE Shell Program

Syntax

```
#include <qp2shell.h>

void QP2SHELL(const char    *pathName,
              ...);

void QP2SHELL2(const char   *pathName,
               ...);
```

Default Public Authority: *USE

Threadsafe: No

Programs QP2SHELL and QP2SHELL2 run an OS/400 Portable Application Solutions Environment (OS/400 PASE) program in the job where the API is called. They load the OS/400 PASE program and any necessary shared libraries and then transfer control to the program. QP2SHELL runs in a new ILE activation group, while QP2SHELL2 runs in the caller's activation group. Control returns to the caller when the OS/400 PASE program either exits, terminates due to a signal, or returns without exiting.

Parameters

pathName

(Input) Pointer to a null-terminated character string that identifies the stream file in the Integrated File System that contains the OS/400 PASE program to run. The pathName string may include an absolute or relative path qualifier in addition to the stream file name. Relative path names are resolved using the current working directory.

If the base name part of the pathName value (excluding any prefix path qualifier) begins with a hyphen (-), QP2SHELL and QP2SHELL2 strip the hyphen when locating the bytestream file, but pass the full string (with the hyphen) to the OS/400 PASE program as the program name. Standard OS/400 PASE shell programs (including sh and ksh) run as login shells when called with a hyphen as the first character of the program name. Login shells look for a profile file and run it automatically when the shell starts.

argument strings

(Input) Optional pointers to null-terminated character strings that are passed to the OS/400 PASE program as arguments. The system copies argument strings into OS/400 PASE memory and converts them from the job default CCSID to the CCSID specified by ILE environment variable QIBM_PASE_CCSD.

Note: When calling QP2SHELL or QP2SHELL2 from CL, be sure to quote any argument string that could be interpreted as a numeric value. CL converts unquoted numeric arguments to decimal or floating-point format, which does not match the assumption made by these APIs and OS/400 PASE programs that all arguments are null-terminated character strings.

Authorities

Object Referred to	Authority Required
--------------------	--------------------

Each directory in the path to the OS/400 PASE program and shared libraries	*X
OS/400 PASE program (not a shell script) in a local file system	*X
OS/400 PASE program in a remote file system or shell script	*RX
OS/400 PASE shared library	*R

Return Value

QP2SHELL and QP2SHELL2 return no function result. Escape messages are sent to report errors.

Error Messages

Some of the more common error messages sent by QP2SHELL and QP2SHELL2 are:

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB9C0 E	Error loading program &1. See previous messages.
CPFB9C1 E	System support for OS/400 Portable Application Solutions Environment not available.
CPFB9C2 E	Hardware support for OS/400 Portable Application Solutions Environment not available.
CPFB9C3 E	OS/400 PASE CCSID and job default CCSID are not compatible.
CPFB9C5 E	OS/400 PASE program name required by QP2SHELL.
CPFB9C6 E	OS/400 PASE ended for signal &1, error code &2.
CPFB9C7 E	OS/400 PASE already running in this job.
CPFB9C8 E	File descriptors 0, 1, and 2 must be open to run the OS/400 PASE program.

Usage Notes

1. QP2SHELL and QP2SHELL2 provide callable program interfaces to ILE procedure Qp2RunPase. See [Qp2RunPase\(\)--Run an OS/400 PASE Program](#) for details about running an OS/400 PASE program.
2. QP2SHELL and QP2SHELL2 set the ILE pthread cancel state and cancel type to default values (PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED) before running the OS/400 PASE program. This is done to avoid unexpected behavior for the OS/400 PASE program if the job changed ILE pthread attributes before calling the API.
3. QP2SHELL and QP2SHELL2 set up handlers for all ILE signals (that call Qp2SignalPase to post an equivalent OS/400 PASE signal) while the OS/400 PASE program runs. QP2SHELL always restores original ILE signal handlers before returning to the caller. QP2SHELL2 restores original ILE signal handlers before returning if the OS/400 PASE program exits, but if the OS/400 PASE program returns without exiting, original ILE signal handlers are not restored until the system destroys the activation group that called QP2SHELL2.
4. To avoid unpredictable results, do not not change ILE environment variables QIBM_USE_DESCRIPTOR_STDIO or QIBM_PASE_DESCRIPTOR_STDIO in a job in which an OS/400 PASE program is running.

5. QP2SHELL and QP2SHELL2 initialize OS/400 PASE environment variables with a modified copy of the entire ILE environment. An OS/400 PASE environment variable is initialized for every ILE environment variable, but the initial value of any OS/400 PASE variable (except those whose name begins with "PASE_") can be overridden by the value of an ILE environment variable with a name that concatenates the prefix PASE_ with the original variable name. This processing avoids some interference between OS/400 PASE runtime and ILE runtime when they require different values for the same environment variable (for example, LANG).
6. For a login shell (only), QP2SHELL and QP2SHELL2 set ILE environment variable PASE_SHELL to the path name of the OS/400 PASE shell program.
7. QP2SHELL and QP2SHELL2 initialize any of the following ILE environment variables that are not already set, with default values as shown:

<i>HOME</i>	If HOME is not already set, QP2SHELL and QP2SHELL2 set it to the home directory path specified in the user profile identified by the LOGIN variable. If the job is not currently authorized to the LOGIN user profile, the HOME environment variable is set to a null string.
<i>LOGIN</i>	If LOGIN is not already set, QP2SHELL and QP2SHELL set it to the middle qualifier of the job name. For an interactive job, this is the name of the user who did a signon to start the job.
<i>PASE_PATH</i>	(Default: >>"/QOpenSys/usr/bin:/usr/ccs/bin:/QOpenSys/usr/bin/X11:/usr/sbin../usr/bin"<<< Initial value for the OS/400 PASE PATH environment variable.
<i>PASE_LANG and QIBM_PASE_CCSID</i>	Initial value for the OS/400 PASE LANG environment variable and what coded character set identifier (CCSID) the OS/400 PASE program will use. QP2SHELL and QP2SHELL2 set both these ILE environment variables if either or both is absent. The default values are function of the current LANGID and CNTRYID attributes of the job, but the system will use PASE_LANG=POSIX and QIBM_PASE_CCSID=819 if it does not recognize the LANGID and CNTRYID pair. The OS/400 PASE LANG environment variable controls the default locale for an OS/400 PASE program. See OS/400 PASE Locales to determine what locales are supported by OS/400 PASE.
<i>PASE_LOCPATH</i>	(Default: "/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat") Initial value for the OS/400 PASE LOCPATH environment variable.
<i>PASE_LC_FASTMSG</i>	(Default: "true") Initial value for the OS/400 PASE LC_FASTMSG environment variable.
<i>PASE_TZ</i>	(Default: "") Initial value for the OS/400 PASE TZ environment variable. If no timezone information is provided in environment variable TZ, the OS/400 PASE program sees UTC (Universal Standard Time) as local time. You may want to set ILE environment variable PASE_TZ at the system level to provide a default timezone other than UTC for OS/400 PASE programs. For example, this CL command sets the default timezone to US Central time: ADDENVVAR ENVVAR(PASE_TZ) VALUE('CST6CDT') LEVEL(*SYS)
<i>QIBM_IFS_OPEN_MAX</i>	(Default: "33000") Maximum number of Integrated File System open file descriptors desired in the job. QP2SHELL and QP2SHELL call the DosSetRelMaxFH API to set the maximum number of file descriptors to the value in this ILE environment variable, and updates the environment variable to reflect the actual limit (in case the requested limit is not currently allowed). Any change to the maximum number of file descriptors persists after the API returns.

OS/400 PASE programs assume the ability to open 32 767 files and the system requires an open file for each OS/400 PASE executable it loads, so the default of 33 000 files accomodates a maximally large OS/400 PASE program with a fairly large number of loaded executables.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)
- [Qp2SignalPase\(\)--Post an OS/400 PASE Signal](#)
- [QP2TERM\(\)--Run an OS/400 PASE Terminal Session](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

QP2TERM()--Run an OS/400 PASE Terminal Session

Syntax

```
#include <qp2term.h>
```

```
void QP2TERM(...);
```

Default Public Authority: *USE

Threadsafe: No

The QP2TERM() program runs an interactive terminal session that starts a batch job to run an OS/400 Portable Application Solutions Environment (OS/400 PASE) program. This program uses the workstation display in the interactive to present output and accept input for files stdin, stdout, and stderr in the batch job.

Parameters

argument strings

(Input) Optional pointers to null-terminated character strings that specify the path name of the OS/400 PASE program to run and any argument strings to pass to the program. If no parameters are specified, QP2TERM runs the default OS/400 PASE shell as an interactive login shell. The default OS/400 PASE shell is an implementation of the Korn shell, with path name /QOpenSys/usr/bin/sh.

Note: When calling QP2TERM from CL, be sure to quote any argument string that could be interpreted as a numeric value. CL converts unquoted numeric arguments to decimal or floating-point format, which does not match the assumption made by QP2TERM and OS/400 PASE programs that all arguments are null-terminated character strings.

Authorities

Object Referred to	Authority Required
Each directory in the path to the OS/400 PASE program and shared libraries	*X
OS/400 PASE program (not a shell script) in a local file system	*X
OS/400 PASE program in a remote file system or shell script	*RX
OS/400 PASE shared library	*R

Return Value

QP2TERM returns no function result. Escape messages are sent to report errors.

Error Messages

Message ID	Error Message Text
CPF9B9C4 E	Error running OS/400 PASE terminal session, reason code &1, errno &2.
CPF9B9C9 E	Terminal session already in use.
CPF9B9CA E	Batch job ended in error.

Usage Notes

1. QP2TERM uses the Qp0zStartTerminal API to manage the interactive display and start a batch job. The batch job copies most attributes of the interactive job and calls program QP2SHELL to run the OS/400 PASE program. See [QP2SHELL\(\)--Run an OS/400 PASE Shell Program](#) for details about running an OS/400 PASE shell program.
2. QP2TERM copies all ILE environment variables from the interactive job to the batch job before starting the batch job, except the following ILE environment variables, which are set or replaced in the batch job. These changes affect the batch job only. They do not modify the environment in the job that called QP2TERM.

<i>COLUMNS</i>	If COLUMNS is not already set, QP2TERM sets it to the number of columns available for program output on the interactive display.
<i>ROWS</i>	If ROWS is not already set, QP2TERM sets it to the number of rows available for program output on the interactive display.
<i>QIBM_USE_DESCRIPTOR_STDIO=I</i>	QP2TERM sets QIBM_USE_DESCRIPTOR_STDIO to ensure that files stdin, stdout, and stderr use Integrated File System descriptors 0, 1, and 2. The terminal session manager attaches open pipes to these file descriptors in the batch job.
<i>QIBM_PASE_DESCRIPTOR_STDIO=T</i>	QP2TERM sets QIBM_PASE_DESCRIPTOR_STDIO to ensure that OS/400 PASE runtime does ASCII/EBCDIC text conversion for data that the OS/400 PASE program reads or writes to files stdin, stdout, and stderr.

Related Information

- [Qp0zStartTerminal\(\)--Start a Terminal Session](#)
- [QP2SHELL\(\)--Run an OS/400 PASE Shell Program](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

OS/400 PASE ILE Procedure APIs

The ILE procedure APIs run an OS/400 PASE program and allow ILE programs to communicate with an OS/400 PASE program that is already running in the same job.

The OS/400 PASE ILE Procedure APIs are:

- [»Allocate OS/400 PASE Heap Memory](#) (Qp2malloc) allocates memory from the OS/400 PASE heap by calling the OS/400 PASE malloc() function.◀◀
- [Call an OS/400 PASE Procedure](#) (Qp2CallPase and Qp2CallPase2) calls a procedure in an OS/400 PASE program that is already running in the job that calls the API.
- [»Close a Dynamically Loaded OS/400 PASE Module](#) (Op2dlclose) closes and unloads an OS/400 PASE module previously opened by the Qp2dlopen API (or the OS/400 PASE dlopen function).◀◀
- [»Dynamically Load an OS/400 PASE Module](#) (Qp2dlopen) dynamically loads an OS/400 PASE module by calling the OS/400 PASE dlopen() function.◀◀
- [»End an OS/400 PASE Program](#) (Qp2EndPase) ends any OS/400 PASE program currently running in the job.◀◀
- [»Find an Exported OS/400 PASE Symbol](#) (Qp2dlsym) finds an exported OS/400 PASE symbol by calling the OS/400 PASE dlsym() function.◀◀
- [»Free OS/400 PASE Heap Memory](#) (Qp2free) frees an OS/400 PASE heap memory allocation by calling the OS/400 PASE free() function.◀◀
- [Post an OS/400 PASE Signal](#) (Qp2SignalPase) posts an OS/400 PASE signal to an OS/400 PASE program that is already running in the job that calls the API.
- [»Retrieve Job CCSID for OS/400 PASE](#) (Qp2jobCCSID) returns the job default CCSID (coded character set identifier) from the last time the OS/400 PASE CCSID was set.◀◀
- [»Retrieve OS/400 PASE CCSID](#) (Qp2paseCCSID) returns the OS/400 PASE CCSID from the last time the OS/400 PASE CCSID was set.◀◀
- [»Retrieve OS/400 PASE Dynamic Load Error Information](#) (Qp2dlerror) returns a pointer to a string that provides error information for the most recent dynamic load function (Qp2dlopen, Qp2dlsym, or Qp2dlclose API).◀◀
- [»Retrieve OS/400 PASE errno Pointer](#) (Qp2errnop) returns a pointer to the OS/400 PASE errno variable for the current thread.◀◀
- [»Retrieve OS/400 PASE Pointer Size](#) (Qp2ptrsize) returns the pointer size, in bytes, for the OS/400 Portable Application Solutions Environment (OS/400 PASE) program currently running in the job.◀◀
- [Run an OS/400 PASE Program](#) (Qp2RunPase) runs an OS/400 PASE program in the job that calls the API.

» Qp2malloc()--Allocate OS/400 PASE Heap Memory

Syntax

```
#include <qp2user.h>

void* Qp2malloc(QP2_dword_t size,
               QP2_ptr64_t *mem_pase);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2malloc() allocates memory from the OS/400 PASE heap by calling the OS/400 PASE malloc() function.

Parameters

size

(Input) The size, in bytes, of the desired memory allocation.

mem_pase

(Input) A pointer to a buffer, used to return the OS/400 PASE address of the allocated memory. The return value is always 64-bits, even for a 32-bit OS/400 PASE program. mem_pase can be null if the caller does not need the OS/400 PASE address of the memory allocation.

Authorities

None.

Return Value

The function result is a pointer to the OS/400 PASE heap memory allocation, or a null pointer if no memory was allocated. A buffer addressed by the mem_pase argument is unchanged if no memory was allocated.

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [OS/400 PASE malloc\(\)--See AIX documentation](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE errno Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

Qp2CallPase()--Call an OS/400 PASE Procedure

Syntax >>

```
#include <qp2user.h>
```

```
int Qp2CallPase(const void          *target,  
               const void          *arglist,  
               const QP2_arg_type_t *signature,  
               QP2_result_type_t   result_type,  
               void                 *buf);
```

```
int Qp2CallPase2(const void          *target,  
                 const void          *arglist,  
                 const QP2_arg_type_t *signature,  
                 QP2_result_type_t   result_type,  
                 void                 *buf,  
                 short                buflenIn);
```

◀ Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

The Qp2CallPase() and Qp2CallPase2 functions call a procedure in an OS/400 Portable Application Solutions Environment (OS/400 PASE) program in a job that is already running the OS/400 PASE program.

Parameters

target

(Input) Pointer to a function descriptor for the procedure (in the OS/400 PASE program) to call. The format and contents of a function descriptor are specified by the PowerPC Application Binary Interface (ABI) for AIX. A function descriptor contains three OS/400 PASE addresses (not MI pointers) that point to the executable instructions, table of contents (TOC), and environment for the target procedure.

arglist

(Input) Pointer to the argument list for the OS/400 PASE procedure. The format and contents of a PASE argument list generally are specified by the PowerPC ABI for AIX. The specific argument list structure for the OS/400 PASE procedure identified by the target parameter is determined by the list of argument data types specified by the signature parameter.

signature

(Input) Pointer to an array of values that specify the sequence and type of arguments passed to the

OS/400 PASE procedure. Each element in the array is either a special value defined in header file `qp2user.h` or a positive number that is the length in bytes of a structure or union argument passed by value. The last value in the array must be `QP2_ARG_END`. Header file `qp2user.h` defines the following constants for the data types supported as arguments for an OS/400 PASE procedure:

- QP2_ARG_END* (0) The end of the list of argument type values.
- QP2_ARG_WORD* (-1) A 4-byte signed or unsigned integer, or a structure or union no longer than four bytes. This value is allowed only when calling a procedure in a 32-bit OS/400 PASE program.
- QP2_ARG_DWORD* (-2) An 8-byte signed or unsigned integer, or a structure or union no longer than eight bytes. This value is allowed only when calling a procedure in a 64-bit OS/400 PASE program.
- QP2_ARG_FLOAT32* (-3) A 4-byte floating point number.
- QP2_ARG_FLOAT64* (-4) An 8-byte floating point number.
- »»*QP2_ARG_PTR32* (-5) A 4-byte pointer. The value in the arglist buffer is passed unchanged unless its high-order bits (excluding the lower 16 bits) match the corresponding part of constant `QP2_ARG_PTR_TOSTACK` (0x0fff0000). In that case, the arglist value is changed to the memory address used for a copy of the buf area plus an offset in the lower 16 bits of the arglist value, and the updated value is passed to the OS/400 PASE procedure. `QP2_ARG_PTR32` is allowed only when calling a procedure in a 32-bit OS/400 PASE program.◀◀
- »»*QP2_ARG_PTR64* (-6) An 8-byte pointer. The value in the arglist buffer is passed unchanged unless its high-order bits (excluding the lower 16 bits) match the corresponding part of constant `QP2_ARG_PTR_TOSTACK` (0x00000000fff0000). In that case, the arglist value is changed to the memory address used for a copy of the buf area plus an offset in the lower 16 bits of the arglist value, and the updated value is passed to the OS/400 PASE procedure. `QP2_ARG_PTR64` is allowed only when calling a procedure in a 64-bit OS/400 PASE program.◀◀

result_type

(Input) The data type of the function result returned by the OS/400 PASE procedure. »»`result_type` is either a special value defined in header file `qp2user.h` or a positive number that is the length in bytes of by-address result data copied from the OS/400 PASE stack to the buf area after the OS/400 PASE procedure returns.◀◀ Header file `qp2user.h` defines the following constants for function result data types:

- QP2_RESULT_VOID* (0) No function result returned.

QP2_RESULT_WORD (-1) A 4-byte signed or unsigned integer, or a structure or union no longer than four bytes. This value is allowed only when calling a procedure in a 32-bit OS/400 PASE program.

QP2_RESULT_DWORD (-2) An 8-byte signed or unsigned integer, or a structure or union no longer than eight bytes returned by a procedure in a 64-bit OS/400 PASE program.

QP2_RESULT_FLOAT64 (-4) An 8-byte floating point number.

»»*QP2_RESULT_PTR32* (-5) A 4-byte pointer. A pointer result from the OS/400 PASE procedure is returned unchanged. This value is allowed only when calling a procedure in a 32-bit OS/400 PASE program.◀◀

»»*QP2_RESULT_PTR64* (-6) An 8-byte pointer. A pointer result from the OS/400 PASE procedure is returned unchanged. This value is allowed only when calling a procedure in a 64-bit OS/400 PASE program.◀◀

buf

(Input/Output) Pointer to a buffer that contains by-address argument data and the function result. buf is ignored if result_type is QP2_RESULT_VOID and bufLenIn is either zero or omitted (for Qp2CallPase).

»»**bufLenIn**

(Input) Length of by-address argument input data. A positive number specifies the number of bytes copied from the buf area to the OS/400 PASE stack before the OS/400 PASE procedure is called.◀◀

Authorities

None.

Return Value

The function result is an integer that indicates whether the OS/400 PASE function was called successfully. Header file qp2user.h defines the following constants for the return code from Qp2CallPase and Qp2CallPase2:

- QP2CALLPASE_NORMAL* (0) The OS/400 PASE procedure ran to completion and its function result (if any) was stored in the location identified by the buf parameter.
- QP2CALLPASE_RESULT_ERROR* (1) The OS/400 PASE procedure ran to completion, but its function result could not be stored at the location identified by the buf parameter. buf may be a null pointer value, or the space addressed by buf may be damaged or destroyed.

<i>QP2CALLPASE_ENVIRON_ERROR</i> (2)	The operation is not allowed because no OS/400 PASE program is running in the job, or the thread that called Qp2CallPase or Qp2CallPase2 was neither the initial OS/400 PASE thread nor a thread created using OS/400 PASE pthread interfaces.
<i>QP2CALLPASE_ARG_ERROR</i> (4)	One or more values in the signature array are not valid.
<i>QP2CALLPASE_TERMINATING</i> (6)	The OS/400 PASE program is terminating. No function result was returned. The OS/400 PASE program may have run the exit function, or a signal might have caused the program to terminate.
» <i>QP2CALLPASE_RETURN_NOEXIT</i> (7)	The OS/400 PASE program returned without exiting by calling the OS/400 PASE _RETURN function. No function result was returned.⚡

Usage Notes

1. Qp2CallPase and Qp2CallPase2 are supported only when an OS/400 PASE program is currently running in the job. This means that Qp2RunPase must be running actively in the job, or the job must be a fork child process.
2. »You can run Qp2CallPase and Qp2CallPase2 only in the initial thread that started the OS/400 PASE program or in a thread created using OS/400 PASE pthread interfaces, unless OS/400 PASE environment variable PASE_THREAD_ATTACH was set to Y when a thread-enabled OS/400 PASE program was started.⚡
3. »Once an ILE thread has attached to OS/400 PASE (by calling an OS/400 PASE procedure), that thread is subject to asynchronous interruption for OS/400 PASE functions such as signal handling and thread cancellation. In particular, the thread will be canceled as part of ending the OS/400 PASE program (when **exit** runs or OS/400 PASE processing terminates for a signal).⚡
4. An OS/400 PASE procedure called by Qp2CallPase or Qp2CallPase2 must return to its caller. Unpredictable results occur if the OS/400 PASE procedure attempts to longjmp to an older call or if it performs an operation that terminates the thread or process (such as calling the exit function). If a signal handler is on the OS/400 PASE stack when Qp2CallPase or Qp2CallPase2 is called, the called OS/400 PASE procedure must also honor restrictions on runtime functions allowed in signal handlers (see AIX signal handling documentation for details).
5. A pointer to any function in an OS/400 PASE program is really a pointer to a function descriptor for the procedure. An OS/400 PASE program can easily provide a function descriptor to ILE code by passing an OS/400 PASE function pointer value converted to an ILE memory address. The conversion can be done using the _SETSPF function or the ARG_MEMPTR argument type on the _ILECALLX or _ILECALL function.
6. »Qp2CallPase and Qp2CallPase2 support arguments and results passed by-address through the use of

QP2_ARG_PTR32 or QP2_ARG_PTR64 values in the signature array and positive numbers for the result_type and/or bufLenIn arguments. <<

7. >> If the buf area is 16-byte aligned, any tagged ILE pointers are preserved in by-address (input) argument data copied from the buf area to OS/400 PASE memory, and in by-address result data copied from OS/400 PASE memory to the buf area. <<
8. >> A structure or union function result returned by-value that is short enough to fit into a register must be handled as QP2_RESULT_WORD for a 32-bit OS/400 PASE program or as QP2_RESULT_DWORD for a 64-bit OS/400 PASE program. Longer structure or union function results returned by-value are actually returned by-address through a buffer pointer passed as the first (hidden) argument to the OS/400 PASE procedure. <<
9. >> You may need to limit result_type and bufLenIn to avoid overrunning the end of the OS/400 PASE stack. Arguments and results that are too large for the stack can be passed by-address using argument pointers to OS/400 PASE heap storage. <<
10. The PowerPC ABI for AIX requires 4-byte alignment for each argument passed to a procedure in a 32-bit program, and 8-byte alignment for each argument passed to a procedure in a 64-bit program. Qp2CallPase and Qp2CallPase2 assume the caller provides an arglist data structure that provides this alignment, including any necessary pad bytes following a structure or union argument and following a QP2_ARG_FLOAT32 argument passed to a 64-bit OS/400 PASE program. The arglist structure also needs to store any 64-bit integer or floating point argument on a 4-byte boundary when the target procedure is in a 32-bit OS/400 PASE program (rather than the 8-byte boundary used as the default for these types in ILE C and C++ compilers).

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2dlclose()--Close a Dynamically Loaded OS/400 PASE Module

Syntax

```
#include <qp2user.h>

int Qp2dlclose(QP2_ptr64_t id);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2dlclose() closes and unloads an OS/400 PASE module previously opened by API Qp2dlopen (or the OS/400 PASE dlopen function).

Parameters

id

(Input) Specifies a value returned by API Qp2dlopen (or the OS/400 PASE dlopen function) that specifies what module is closed and unloaded.

Authorities

None.

Return Value

The function result is zero for normal completion, or -1 with an error indicated in ILE **errno** or OS/400 PASE **errno** (if ILE **errno** is zero). You can also call API Qp2dlerror for more information about any error.

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [OS/400 PASE dlclose\(\)--See AIX documentation](#)
- [Qp2dlerror\(\)--Retrieve OS/400 PASE Dynamic Load Error Information](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE errno Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2dlopen()--Dynamically Load an OS/400 PASE Module

Syntax

```
#include <qp2user.h>

QP2_ptr64_t Qp2dlopen(const char *path,
                    int flags,
                    int ccsid);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2dlopen() dynamically loads an OS/400 PASE module by calling the OS/400 PASE dlopen() function.

Parameters

path

(Input) A pointer to a null-terminated string that identifies the stream file in the Integrated File System that contains the OS/400 PASE module to load. This API copies the input path string and converts the copy from the CCSID specified by the **ccsid** argument to the current OS/400 PASE CCSID (required by the OS/400 PASE dlopen function).

If the input path pointer is null, the function result is a value for the main application that lets you find symbols in the OS/400 PASE process global name space, which includes all symbols exported by the OS/400 PASE program and shared executables except those loaded by OS/400 PASE dlopen using option RTLD_LOCAL.

flags

(Input) Flags passed to the OS/400 PASE dlopen function to control its behavior. These constants, declared in qp2user.h, match constants in AIX header dlfcn.h (without the leading prefix, QP2_) and can be **ORed** together for the flags argument:

QP2_RTLD_NOW (0x00000002)

Load all dependents of the module being loaded and resolve all symbols. Either QP2_RTLD_NOW or QP2_RTLD_LAZY must be specified.

QP2_RTLD_LAZY (0x00000004)

Allow the system to defer loading dependent modules. Either QP2_RTLD_NOW or QP2_RTLD_LAZY must be specified.

QP2_RTLD_GLOBAL (0x00010000)

Load the module into the global name space. Exported symbols in the module will be visible in the main application and will be used when resolving symbols used by other OS/400 PASE dlopen calls.

QP2_RTLD_LOCAL (0x00080000)

Load the module into a local name space. This option is the default when neither QP2_RTLD_GLOBAL nor QP2_RTLD_LOCAL is specified. It prevents symbols in the module being loaded from being used when resolving symbols used by other dlopen calls.

QP2_RTLD_MEMBER (0x00040000)

Specifies that the *path* argument string may contain the name of a member in an archive (shared library).

QP2_RTLD_NOAUTODEFER (0x00020000)

Prevent deferred imports in the module being loaded from being automatically resolved by subsequent loads.

ccsid

(Input) Specifies the CCSID for the input *path* argument string. Zero means the path is in the (EBCDIC) job default CCSID.

Authorities

Object Referred to	Authority Required
Each directory in the path to the OS/400 PASE module	*X
OS/400 PASE module	*R

Return Value

Successful completion returns a non-zero function result that can be used to call APIs Qp2dlsym and Qp2dlclose (and also OS/400 PASE functions dlsym and dlclose). Resources allocated for the function result are not freed until the OS/400 PASE program ends or the value is passed to API Qp2dlclose (or OS/400 PASE dlclose).

A zero function result indicates an error. The caller can check ILE **errno** or OS/400 PASE **errno** (if ILE **errno** is zero), or call the Qp2dlerror API for more information about the error.

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [OS/400 PASE dlopen\(\)--See AIX documentation](#)
- [Qp2dlerror\(\)--Retrieve OS/400 PASE Dynamic Load Error Information](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE errno Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2EndPase()--End an OS/400 PASE Program

Syntax

```
#include <qp2user.h>

int Qp2EndPase(void);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: No

The Qp2EndPase() function ends any OS/400 PASE program currently running in the job.

Parameters

None.

Authorities

None.

Return Value

The function result is nonzero if an error is detected attempting to end the OS/400 PASE program.

Usage Notes

1. Qp2EndPase is normally used to end an OS/400 PASE program that ran the **_RETURN** OS/400 PASE runtime function (to return without exiting). Such a program remains active (even if it exits or terminates due to an OS/400 PASE signal) until either Qp2EndPase is called or the ILE activation group that called the Qp2RunPase API exits. OS/400 PASE programs that do not use **_RETURN** are ended automatically before control returns from the Qp2RunPase API.
2. Qp2EndPase returns without error when no OS/400 PASE program is running in the job.
3. Undefined behavior results if Qp2EndPase is called while the Qp2RunPase API is running (in the same job), or if the activation group that ran the Qp2RunPase API attempts to use the OS/400 PASE program (without restarting it) after Qp2EndPase is called from a different activation group.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)
- [_RETURN\(\)--Return Without Exiting OS/400 PASE](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2dlsym()--Find an Exported OS/400 PASE Symbol

Syntax

```
#include <qp2user.h>

void* Qp2dlsym(QP2_ptr64_t id
               const char *name,
               int         ccsid,
               QP2_ptr64_t *sym_pase);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2dlsym() finds an exported OS/400 PASE symbol by calling the OS/400 PASE dlsym() function.

Parameters

id

(Input) Specifies a value returned by API Qp2dlopen (or the OS/400 PASE dlopen function) that controls what modules are searched for the exported symbol.

name

(Input) A pointer to a null-terminated string that contains the symbol name. This API copies the input name string and converts the copy from the CCSID specified by the **ccsid** argument to the current OS/400 PASE CCSID (required by the OS/400 PASE dlsym function).

ccsid

(Input) Specifies the CCSID for the input *name* argument string. Zero means the symbol name is in the (EBCDIC) job default CCSID.

sym_pase

(Input) A pointer to a buffer, used to return the OS/400 PASE address of the exported symbol. The return value is always 64-bits, even for a 32-bit OS/400 PASE program. *sym_pase* can be null if the caller does not need the OS/400 PASE address of the symbol.

Authorities

None.

Return Value

The function result is a pointer to the specified symbol, or a null pointer if the symbol could not be resolved. A buffer addressed by the `sym_pase` argument is unchanged if the symbol could not be resolved. The caller can check ILE **errno** or OS/400 PASE **errno** (if ILE **errno** is zero), or call the `Qp2dlerror` API for more information about any error.

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API `Qp2RunPase`, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [OS/400 PASE `dlsym\(\)`--See AIX documentation](#)
- [Qp2dlerror\(\)--Retrieve OS/400 PASE Dynamic Load Error Information](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE `errno` Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

» Qp2free()--Free OS/400 PASE Heap Memory

Syntax

```
#include <qp2user.h>

int Qp2free(void *mem);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2free() frees an OS/400 PASE heap memory allocation by calling the OS/400 PASE free() function.

Parameters

mem

(Input) A pointer to the start of the OS/400 PASE memory allocation to be freed.

Authorities

None.

Return Value

The function result is zero for normal completion, or -1 with an error indicated in ILE **errno** that is usually one of the following:

EPERM An error occurred attempting to call an OS/400 PASE function.

ETERM OS/400 PASE is terminating.

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [OS/400 PASE free\(\)--See AIX documentation](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE errno Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

Qp2SignalPase()--Post an OS/400 PASE Signal

Syntax

```
#include <qp2user.h>

int Qp2SignalPase(int signo);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

The Qp2SignalPase() function posts an OS/400 Portable Application Solutions Environment (OS/400 PASE) signal to an OS/400 PASE program that is already running in the job.

Parameters

signo

(Input) Signal number to post. A positive value is an ILE signal number, which causes the system to post a corresponding OS/400 PASE signal. ILE and OS/400 PASE signals correspond if they have the same name (for example, SIGTERM) in a system-provided header file. A negative value is the negation of an OS/400 PASE (and AIX) signal number.

Authorities

None.

Return Value

The function result is an integer that indicates whether the OS/400 PASE signal was posted successfully. Header file qp2user.h defines the following constants for the return code from Qp2SignalPase:

- | | |
|-------------------------------------|--|
| <i>QP2CALLPASE_NORMAL(0)</i> | An OS/400 PASE signal was posted successfully. |
| <i>QP2CALLPASE_ENVIRON_ERROR(2)</i> | The operation is not allowed because no OS/400 PASE program is running in the job, or the thread that called Qp2CallPase was neither the initial OS/400 PASE thread nor a thread created using OS/400 PASE pthread interfaces. |
| <i>QP2CALLPASE_ARG_ERROR(4)</i> | The signo parameter value is invalid. |

Qp2CALLPASE_TERMINATING(6)

The OS/400 PASE program is terminating. No function result was returned. The OS/400 PASE program may have run the exit function, or a signal might have caused the program to terminate.

Usage Notes

1. Qp2SignalPase is supported only when an OS/400 PASE program is currently running in the job. This means that Qp2RunPase must be actively called in the job, or the job must be a fork child process.
2. Not all ILE signals have an OS/400 PASE equivalent and Qp2SignalPase never converts ILE SIGCHLD to a corresponding PASE signal. This special handling for SIGCHLD avoids duplicate PASE signals for the termination of a single child process (because the system may send both ILE and OS/400 PASE signals to the parent of any fork child process that ends).
3. If there is only one OS/400 PASE thread running in the job, the signal remains pending until control is transferred to the OS/400 PASE program. If other OS/400 PASE threads are running at the time Qp2SignalPase is called, the system may chose one of the other threads to deliver the signal.

Related Information

- [Qp2RunPase\(\)--Call an OS/400 PASE Procedure](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2jobCCSID()--Retrieve Job CCSID for OS/400 PASE

Syntax

```
#include <qp2user.h>           /* for ILE programs */
#include <as400_protos.h>      /* for OS/400 PASE programs */

int Qp2jobCCSID(void);
```

Service Program Name: QP2USER (for ILE programs)

OS/400 PASE Library: libc.a (for OS/400 PASE programs)

Default Public Authority: *USE

Threadsafe: Yes

Note: This function can be used in either an ILE program or an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

Qp2jobCCSID() returns the job default CCSID (coded character set identifier) from the last time the OS/400 PASE CCSID was set. The OS/400 PASE CCSID is set when an OS/400 PASE program starts, and can be changed by the OS/400 PASE runtime function `_SETCCSID`.

Parameters

None.

Authorities

None.

Return Value

The function result is a coded character set identifier (CCSID), or 0 if OS/400 PASE CCSID information is not available (such as when no OS/400 PASE program is running in the job).

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)
- [Qp2paseCCSID\(\)--Retrieve OS/400 PASE CCSID](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2paseCCSID()--Retrieve OS/400 PASE CCSID

Syntax

```
#include <qp2user.h>           /* for ILE programs */
#include <as400_protos.h>      /* for OS/400 PASE programs */

int Qp2paseCCSID(void);
```

Service Program Name: QP2USER (for ILE programs)

OS/400 PASE Library: libc.a (for OS/400 PASE programs)

Default Public Authority: *USE

Threadsafe: Yes

Note: This function can be used in either an ILE program or an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

Qp2paseCCSID() returns the OS/400 PASE CCSID (coded character set identifier) from the last time the OS/400 PASE CCSID was set. The OS/400 PASE CCSID is set when an OS/400 PASE program starts, and can be changed by the OS/400 PASE runtime function `_SETCCSID`.

Parameters

None.

Authorities

None.

Return Value

The function result is a coded character set identifier (CCSID), or 0 if OS/400 PASE CCSID information is not available (such as when no OS/400 PASE program is running in the job).

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)
- [Qp2jobCCSID\(\)--Retrieve Job CCSID for OS/400 PASE](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2dlerror()--Retrieve OS/400 PASE Dynamic Load Error Information

Syntax

```
#include <qp2user.h>

char* Qp2dlerror(void);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: No

Qp2dlerror() returns a pointer to a string that provides error information for the most recent dynamic load function (API Qp2dlopen, Qp2dlsym, or Qp2dlclose).

Parameters

None.

Authorities

None.

Return Value

The function result is a pointer to a null-terminated character string (in the job default CCSID). A null pointer is returned if no error occurred during the most recent dynamic load operation. Once Qp2dlerror is called, subsequent calls without an intervening dynamic load error also return a null pointer.

The ILE **errno** is set and a null pointer is returned for any internal processing error (such as an error converting the string from the OS/400 PASE CCSID to the job default CCSID).

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.
2. Qp2dlerror is not threadsafe because it may call an OS/400 PASE function that is not threadsafe

(dlerror) and uses a buffer in static storage for the error string that is also updated by other dynamic load functions (APIs Qp2dlopen, Qp2dlsym, and Qp2dlclose). Applications may need to serialize use of dynamic load functions and copy the error information string to preserve its contents.

Related Information

- [OS/400 PASE dlerror\(\)--See AIX documentation](#)
- [Qp2errnop\(\)--Retrieve OS/400 PASE errno Pointer](#)
- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2errnop()--Retrieve OS/400 PASE errno Pointer

Syntax

```
#include <qp2user.h>

int* Qp2errnop(void);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2errnop() returns a pointer to the OS/400 PASE **errno** variable for the current thread.

Parameters

None.

Authorities

None.

Return Value

The function result is a pointer to the OS/400 PASE **errno** variable for the current thread, or a null pointer if **errno** location is not available (such as when no OS/400 PASE program is running in the job).

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» Qp2ptrsize()--Retrieve OS/400 PASE Pointer Size

Syntax

```
#include <qp2user.h>

size_t Qp2ptrsize(void);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: Yes

Qp2ptrsize() returns the pointer size, in bytes, for the OS/400 Portable Application Solutions Environment (OS/400 PASE) program currently running in the job.

Parameters

None.

Authorities

None.

Return Value

The function result is 4 for a 32-bit program, or 8 for a 64-bit program. The result is zero if OS/400 PASE pointer size is not available (such as when no OS/400 PASE program is running in the job).

Usage Notes

1. This API can only be used in the same activation group that started OS/400 PASE in the job. This is either the activation group that called API Qp2RunPase, or the default activation group in a job started by the OS/400 PASE runtime function **fork**.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

Qp2RunPase()--Run an OS/400 PASE Program

Syntax

```
#include <qp2user.h>
```

```
int Qp2RunPase(const char      *pathName,  
               const char      *symbolName,  
               const void      *symbolData,  
               unsigned int     symbolDataLen,  
               int              ccsid,  
               const char *const *argv,  
               const char *const *envp);
```

Service Program Name: QP2USER

Default Public Authority: *USE

Threadsafe: No

The Qp2RunPase() function runs an OS/400 Portable Application Solutions Environment (OS/400 PASE) program in the job where the API is called. It loads the OS/400 PASE program and any necessary shared libraries and then transfers control to the program. Control returns to the caller when the OS/400 PASE program exits, terminates due to a signal, or returns without exiting.

Parameters

pathName

(Input) Pointer to a null-terminated character string that identifies the stream file in the Integrated File System that contains the OS/400 PASE program to run. The pathName string may include an absolute or relative path qualifier in addition to the stream file name. Relative path names are resolved using the current working directory.

symbolName

(Input) Pointer to a null-terminated character string that names an external symbol in the OS/400 PASE program. The specified symbol is initialized with data addressed by the symbolData parameter. The OS/400 PASE program is run without initializing symbol data (and no error is reported) if either symbolName is a null pointer or the string does not match an external symbol name in the program.

The system copies the symbolName string internally and converts it from the job default CCSID to the CCSID specified by the ccsid parameter before searching for the (converted) symbol name in the OS/400 PASE program.

symbolData

(Input) Pointer to data used to initialize a symbol (identified by the symbolName parameter) in the OS/400 PASE program. The system copies the data (without modification) into memory that can be referenced by the OS/400 PASE program. Any MI pointers in the data are preserved in the copy. symbolData is ignored if it is a null pointer or if no external symbol name in the program matches the symbolName parameter.

symbolDataLen

(Input) The number of bytes to copy from the address specified by the symbolData parameter to OS/400 PASE memory. symbolDataLen is ignored if symbolData is a null pointer or if no external symbol name in the program matches the symbolName parameter.

ccsid

(Input) The coded character set identifier (CCSID) initially used by the OS/400 PASE program. ccsid must specify a single-byte encoding (normally an ASCII CCSID) that OS/400 can convert to and from the job default CCSID, or a value of 1208 to indicate that the OS/400 PASE program uses UTF-8 encoding.

The system uses ccsid to set the CCSID of any bytestream file created by the OS/400 PASE program, and also to control character encoding conversions done for OS/400 PASE runtime interfaces that use OS/400 services.

argv

(Input) Pointer to an array of pointers to null-terminated character strings that are passed as arguments to the OS/400 PASE program. The last element in the array must be a null pointer. An error is reported if the argv parameter pointer is null.

The system copies argument strings into OS/400 PASE memory and converts them from the job default CCSID to the CCSID specified by the ccsid parameter. By convention, the first argument string passed to an OS/400 PASE program should be the same as the pathName string.

envp

(Input) Pointer to an array of pointers to null-terminated character strings that are passed as environment variables to the OS/400 PASE program. The last element in the array must be a null pointer. envp can be a null pointer if no environment variables need to be initialized for the OS/400 PASE program.

The system copies environment variable strings into OS/400 PASE memory and converts them from the job default CCSID to the CCSID specified by the ccsid parameter. By convention, environment variable strings take the form "NAME=value".

Authorities

Object Referred to	Authority Required
Each directory in the path to the OS/400 PASE program and shared libraries	*X
OS/400 PASE program (not a shell script) in a local file system	*X
OS/400 PASE program in a remote file system or shell script	*RX
OS/400 PASE shared library	*R

Return Value

The function result may be one of these special values:

- QP2RUNPASE_ERROR (-1)* An internal error occurred during Qp2RunPase processing.
- » *QP2RUNPASE_RETURN_NOEXIT (-2)* The OS/400 PASE program returned without exiting (by calling the OS/400 PASE **_RETURN** function).«

If the result is not one of the special values above, it is a value that contains status information about how the OS/400 PASE program ended, in the same format as the *stat_val* parameter for the ILE waitpid function. You can use these macros in file <sys/wait.h> to interpret such a result:

- WIFEXITED(stat_val)* Evaluates to a nonzero value if OS/400 PASE program ended normally.
- WEXITSTATUS(stat_val)* If the value of the *WIFEXITED(stat_val)* is nonzero, evaluates to the low-order 8 bits of the value the OS/400 PASE program specified as the argument to exit or the function result returned by main.
- WIFSIGNALED(stat_val)* Evaluates to a nonzero value if OS/400 PASE program ended because of the receipt of a terminating signal that was not caught by the process.
- WTERMSIG(stat_val)* If the value of *WIFSIGNALED(stat_val)* is nonzero, evaluates to the number of the OS/400 PASE signal that caused the program to end. OS/400 PASE programs use the same signal numbers as AIX (which differ from ILE signal numbers).

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB9C0 E	Error loading program &1. See previous messages.
CPFB9C1 E	System support for OS/400 Portable Application Solutions Environment not available.
CPFB9C2 E	Hardware support for OS/400 Portable Application Solutions Environment not available.
CPFB9C3 E	OS/400 PASE CCSID and job default CCSID are incompatible.
CPFB9C7 E	OS/400 PASE already running in this job.
CPFB9C8 E	File descriptors 0, 1, and 2 must be open to run the OS/400 PASE program.

Usage Notes

1. Qp2RunPase works like the AIX `execve` function, including the ability to run shell scripts and the rules for resolving shared libraries (which may include using OS/400 PASE environment variable `LIBPATH`).
2. ➤ If an absolute path (starting with `"/`) is specified for the `pathName` string or in the first line of a shell script identified by `pathName` and that path cannot be opened or is not a regular bytestream file, the system generally searches the `/QOpenSys` file system for the file. See environment variable `PASE_EXEC_QOPENSYS` in [OS/400 PASE Environment Variables](#) for more information. ⏪
3. Qp2RunPase cannot run an OS/400 PASE program or shared library stored in a file system that is not threadsafe in a job that is multithread capable. Any job started by the OS/400 PASE fork function is multi-thread capable.
4. You can set these ILE environment variables before calling Qp2RunPase to control the OS/400 PASE operation:

QIBM_USE_DESCRIPTOR_STDIO When this ILE environment variable is set to Y or I, both OS/400 PASE runtime and ILE C runtime use Integrated File System file descriptors 0, 1, and 2 for `stdin`, `stdout`, and `stderr`. Otherwise, OS/400 PASE file descriptors 0, 1, and 2 are mapped to ILE C runtime files `stdin`, `stdout`, and `stderr` (which may not use any Integrated File System file descriptors).

OS/400 PASE and ILE generally use different descriptor numbers for the same open file, but when `QIBM_USE_DESCRIPTOR_STDIO` is set to Y or I, any operation against OS/400 PASE file descriptors 0, 1, or 2 is also done for the same Integrated File System file descriptor number so OS/400 PASE and ILE C use the same files for `stdin`, `stdout`, and `stderr`.

QIBM_PASE_DESCRIPTOR_STDIO This ILE environment variable controls ASCII/EBCDIC conversion for data read or written through OS/400 PASE files `stdin`, `stdout`, and `stderr` to Integrated File System file descriptors 0, 1, and 2. ASCII/EBCDIC conversion is always done (and this variable is ignored) unless `QIBM_USE_DESCRIPTOR_STDIO` is set to either Y or I. If `QIBM_PASE_DESCRIPTOR_STDIO` is set to B, the PASE program processes binary data (without ASCII/EBCDIC conversion). Otherwise, ASCII/EBCDIC conversion is done for any data read from or written to OS/400 PASE file descriptors 0, 1, or 2.

QIBM_PASE_USE_PRESTART_JOBS When this ILE environment variable is set to Y, OS/400 PASE runtime uses prestarted jobs for child processes created by `fork` and for any job started by the `systemCL` OS/400 PASE runtime function (to run a CL command). You should add prestarted job entries (`ADDPJE` command) for programs `QPOZSPWT` (used by `fork`) and `QPOZSPWP` (used by `systemCL`) to any subsystem description that will run jobs that use this support.

5. OS/400 PASE environment variables are independent of ILE environment variables. See [OS/400 PASE Environment Variables](#) for more information, including OS/400 PASE environment variables you can set to control runtime behaviors that differ from AIX.
6. A symbol name specified by the symbolName parameter must be defined (not just declared or referenced) in the OS/400 PASE program. The system relocates the symbol to memory that is dynamically allocated to ensure 16-byte alignment. Any initial value specified in the OS/400 PASE program for the relocated symbol is ignored.
7. The ccsid parameter provides the initial OS/400 PASE CCSID value, but the OS/400 PASE program can use the _SETCCSID function to change the OS/400 PASE CCSID or to rebind to a change in the job default CCSID. The OS/400 PASE CCSID should generally be the CCSID equivalent of the code set for the current locale. See [OS/400 PASE Locales](#) to determine what locales are supported by OS/400 PASE.
8. OS/400 PASE programs generally should use functions _ILELOAD and _ILESYM to acquire MI pointers to functions and data exported by an ILE program or service program rather than rely on pointers passed using the symbolName, symbolData, and symbolDataLen parameters on Qp2RunPase. This is because MI pointers in OS/400 PASE memory are destroyed by exec processing and pointers to ILE procedures are unusable in a child process created by fork (because they point to an activation group in the parent process).
9. You may want to increase the number of file descriptors in the job by calling DosSetRelMaxFH before you call Qp2RunPase. By default, OS/400 jobs support only 200 open file descriptors, while OS/400 PASE programs generally expect to be able to open 32 767 file descriptors, and the system requires file descriptors to open bytestream files that contain the OS/400 PASE program and any shared libraries it uses.
10. You may want to establish Qp2SignalPase as the handler for any ILE signal that needs to be visible to the OS/400 PASE program. For example, system support for Sockets (used by OS/400 PASE runtime) only sends SIGIO and SIGURG as ILE signals, so ILE signal handling must be set up before calling an OS/400 PASE program that relies on SIGIO or SIGURG as OS/400 PASE signals. OS/400 PASE runtime automatically establishes Qp2SignalPase as the handler for every ILE signal in a fork child process.
11. You may want to call ILE interfaces pthread_setcancelstate and pthread_setcanceltype to set pthread cancel state and cancel type before calling Qp2RunPase in a process that did prior pthread work. OS/400 PASE pthreads use ILE pthreads and Qp2RunPase assumes that ILE pthread cancel state and cancel type are set to defaults (PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED). The state of these attributes when a program ends is whatever value was last set by either ILE or OS/400 PASE code.
12. Time-of-day information in an OS/400 PASE program depends on the values for system value QUTCFFSET when the OS/400 PASE was started, and OS/400 PASE environment variable TZ when the program retrieves the time. For example, the correct settings for Central Standard Time in the USA are QUTCFFSET=-6 and TZ=CST6CDT.
13. Any credentials changes (user, group, or group list changes) made by an OS/400 PASE program are generally persistent in the job. The job (thread) credentials before and after a call to Qp2RunPase may not be the same if the OS/400 PASE program calls any of the setuid or setgid family of interfaces.

However, the system saves credentials before running any OS/400 PASE program with the S_ISUID or S_ISGID attribute, and automatically restores the saved credentials before returning to the caller of Qp2RunPase.

14. Character conversions controlled by the ccsid parameter only handle the single-byte component of an EBCDIC-mixed CCSID (for the job default CCSID). This restricts the OS/400 PASE program name specified by the pathName parameter, argument strings passed through the argv parameter, and environment variables passed through the envp parameter to single-byte characters. DBCS characters can be passed (unconverted) to an OS/400 PASE program using the symbolData parameter.
15. If an OS/400 PASE program needs to use DBCS characters for OS/400 PASE runtime functions such as file system interfaces, it must run with the OS/400 PASE CCSID (ccsid parameter) set to 1208 because OS/400 PASE runtime provides complete support for DBCS characters using UTF-8 encoding only.

Related Information

- [The <sys/wait.h> file \(see \[Header Files for UNIX-Type Functions\]\(#\)\)](#)
- [DosSetRelMaxFH\(\)--Change Maximum Number of File Descriptors](#)
- [pthread_setcancelstate\(\)--Set Cancel State](#)
- [pthread_setcanceltype\(\)--Set Cancel Type](#)
- [QP2SHELL\(\)--Run an OS/400 PASE Shell Program](#)
- [QP2TERM\(\)--Run an OS/400 PASE Terminal Session](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

Runtime Functions For Use by OS/400 PASE Programs

OS/400 PASE runtime includes interfaces supported on AIX and interfaces unique to OS/400 PASE. They are unique to OS/400 PASE

The runtime functions are:

- [Build an ILE Argument List for OS/400 PASE](#) (build_ILEarglist) builds an ILE argument list using argument values copied from an OS/400 PASE function with the same signature.
- [Call an ILE Procedure for OS/400 PASE](#) (_ILECALL and _ILECALLX) allows an OS/400 PASE program to call an ILE procedure.
- [Call an OS/400 Program for OS/400 PASE](#) (_PGMCALL) calls an OS/400 program (object type *PGM) from an OS/400 PASE program.⏪
- [Compute ILE Argument List Size for OS/400 PASE](#) (size_ILEarglist) computes the number of bytes of memory required to build an ILE argument list.
- [Convert ILE errno to OS/400 PASE errno](#) (_CVTERRNO) converts an ILE errno value to a corresponding OS/400 PASE errno value.
- [Convert Space Pointer for OS/400 PASE](#) (_CVTSPP) converts a tagged space pointer value to an equivalent OS/400 PASE memory address.
- [Copy Character String for OS/400 PASE](#) (_STRNCPY_SPP) copies a null-terminated character string.
- [Copy Memory With Tags for OS/400 PASE](#) (_MEMCPY_WT and _MEMCPY_WT2) allows an OS/400 PASE program to copy memory with tagged pointers.
- [Determine Character String Length for OS/400 PASE](#) (_STRLEN_SPP) determines the length of a null-terminated character string.
- [Find Exported ILE Symbol for OS/400 PASE](#) (_ILESYM) allows an OS/400 PASE program to get a tagged pointer to the data or procedure exported for a symbol exported by an ILE activation.
- [Load an ILE Bound Program for OS/400 PASE](#) (_ILELOAD) allows an OS/400 PASE program to load (activate) an ILE bound program.
- [Override SQL CLI CCSID for OS/400 PASE](#) (SQLOverrideCCSID400) allows an OS/400 PASE program to specify a CCSID for character arguments and results on SQL runtime functions.
- [Receive Nonprogram Message for OS/400 PASE](#) (QMHRMVM and QMHRMVM1) allows an OS/400 PASE program to receive a message from a nonprogram message queue.
- [Receive Program Message for OS/400 PASE](#) (QMHRMVM, QMHRMVM1, and QMHRMVM2) allows an OS/400 PASE program to receive a message from a program call message queue or from the job external message queue.
- [Resolve to an OS/400 Object for OS/400 PASE](#) (_RSLOBJ) resolves to an OS/400 object.⏪
- [Retrieve Job CCSID for OS/400 PASE](#) (Qp2jobCCSID) returns the job default CCSID (coded character set identifier) from the last time the OS/400 PASE CCSID was set.⏪
- [Retrieve OS/400 PASE CCSID](#) (Qp2paseCCSID) returns the OS/400 PASE CCSID from the last time the OS/400 PASE CCSID was set.⏪
- [Return without Exiting OS/400 PASE](#) (_RETURN) returns to the ILE called that called OS/400 PASE in this job, without exiting the OS/400 PASE program.⏪
- [Run a CL Command for OS/400 PASE](#) (systemCL) allows an OS/400 PASE program to run a CL

command.

- [Send Nonprogram Message for OS/400 PASE](#) (QMHSNDM and QMHSNDM1) allows an OS/400 PASE program to send a message to a nonprogram message queue so it can communicate with another job or user.
- [Send Program Message for OS/400 PASE](#) (QMHSNDPM, QMHSNDPM1, and QMHSNDPM2) allows an OS/400 PASE program to send a message to a program call message queue or to the job external message queue.
- [Set OS/400 PASE CCSID](#) (_SETCCSID) retrieves and sets the OS/400 PASE Coded Character Set Identifier (CCSID) value.
- [Set Space Pointer for OS/400 PASE](#) (_SETSPP) sets a tagged space pointer to the teraspace equivalent of an OS/400 PASE memory address.

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

build_ILEarglist()--Build an ILE Argument List for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int build_ILEarglist(ILEarglist_base *ILEarglist,
                    const void *PASEarglist,
                    const arg_type_t *signature);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **build_ILEarglist()** function builds an ILE argument list using argument values copied from an OS/400 PASE function with the same signature.

Parameters

ILEarglist

(Output) Pointer to a 16-byte aligned buffer allocated by the caller for the ILE argument list. ILEarglist must be long enough to contain all arguments specified in the signature list.

PASEarglist

(Input) Pointer to the first argument passed to an OS/400 PASE function that accepts arguments equivalent to those specified by the signature list.

signature

(Input) Pointer to a list of arg_type_t values that specify the sequence and type of arguments passed to the ILE procedure. ILE procedures can accept a maximum of 400 arguments. The actual number of arguments processed by the build_ILEarglist function is determined by the number of entries in the signature list, which is determined by the location of the first ARG_END value in the list. The following values are supported in the signature list:

<i>ARG_END(0)</i>	Specifies the end of the signature list.
<i>ARG_INT8 (-1)</i>	Signed 1-byte integer argument.
<i>ARG_UINT8 (-2)</i>	Unsigned 1-byte integer argument.
<i>ARG_INT16 (-3)</i>	Signed 2-byte integer argument.

<i>ARG_UINT16</i> (-4)	Unsigned 2-byte integer argument.
<i>ARG_INT32</i> (-5)	Signed 4-byte integer argument.
<i>ARG_UINT32</i> (-6)	Unsigned 4-byte integer argument.
<i>ARG_INT64</i> (-7)	Signed 8-byte integer argument.
<i>ARG_UINT64</i> (-8)	Unsigned 8-byte integer argument.
<i>ARG_FLOAT32</i> (-9)	4-byte floating-point argument.
<i>ARG_FLOAT64</i> (-10)	8-byte floating-point argument.
<i>ARG_MEMPTR</i> (-11)	The argument is a memory address. The OS/400 PASE procedure argument is an OS/400 PASE memory address that <code>build_ILEarglist</code> copies into the <code>ILEpointer</code> type value in the ILE argument list. See Call an ILE Procedure for OS/400 PASE (<code>_ILECALLX</code> or <code>_ILECALL</code>) for more information about how <code>ARG_MEMPTR</code> arguments are handled.
<i>Any positive number</i> (1-32767)	The argument is an aggregate (structure or union). The value in the signature list is the length, in bytes, of the aggregate.

Authorities

`build_ILEarglist` requires no authority.

Return Value

`build_ILEarglist` returns the number of bytes used to build the ILE argument list (including storage for the `ILEarglist_base` type), or zero if an error was detected in the input arguments.

Usage Notes

1. `build_ILEarglist` does no character encoding conversions, so the OS/400 PASE program may need to convert argument and result character strings between ASCII and EBCDIC. OS/400 PASE runtime function `iconv` can be used for character conversions.
2. `build_ILEarglist` does not support argument types `ARG_SPCPTR` or `ARG_OPENPTR` (which are supported by `_ILECALLX` and `_ILECALL`) because the AIX Application Binary Interface for PowerPC provides no way to ensure 16-byte alignment for arguments pushed onto the stack.
3. `build_ILEarglist` does not directly support aggregate function results. You need to set `result_r_aggregate.addr` in the `PASEarglist` structure to the address of a buffer where the ILE procedure will store the aggregate result.
4. Older versions of `build_ILEarglist` accepted additional arguments in an attempt to handle aggregate function results, but those arguments were removed because they cannot be supported reliably. If you need to compile source that passes the additional arguments, you must define macro `OLD_build_ILEarglist` and include `<as400_types.h>` to access the old support.

Related Information

- [_ILECALLX\(\)--Call an ILE Procedure for OS/400 PASE](#)
- [size_ILEarglist\(\)--Compute ILE Argument List Size for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_ILECALLX()--Call an ILE Procedure for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

» int _ILECALLX(const ILEpointer *target,
               ILEarglist_base *ILEarglist,
               const arg_type_t *signature,
               result_type_t result_type,
               int flags);
«

int _ILECALL(const ILEpointer *target,
             ILEarglist_base *ILEarglist,
             const arg_type_t *signature,
             result_type_t result_type);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information.

The **_ILECALLX()** and **_ILECALL()** functions call an ILE procedure from an OS/400 PASE program. They transfer control to an ILE procedure specified by a 16-byte tagged ILE procedure pointer, passing arguments and returning the function result.

Parameters

target

(Input) Pointer to a tagged procedure pointer that addresses the ILE procedure to call. target must be a 16-byte aligned OS/400 PASE memory address.

ILEarglist

(Input/Output) Pointer to a 16-byte aligned ILE argument list structure. ILEarglist is the address of the structure that contains any argument values to pass to the ILE procedure, as well as memory for a function result returned by the ILE procedure. ILEarglist must be long enough to contain all arguments required by the target ILE procedure to avoid unpredictable results.

The base structure of an ILE argument list (including a function result area) is specified by type ILEarglist_base. Any argument values for the ILE procedure are stored in memory immediately following the ILEarglist_base type. The specific format of the argument list is determined by the list of

arg_type_t values addressed by the signature argument. The alignment requirements for each argument value in the ILE argument list depends on its length:

Argument Length	Alignment
1 byte	any
2 bytes	2 bytes
3-4 bytes	4 bytes
5-8 bytes	8 bytes
9 or more bytes	16 bytes

signature

(Input) Pointer to a list of arg_type_t values that specify the sequence and type of arguments passed to the ILE procedure. ILE procedures can accept a maximum of 400 arguments. The actual number of arguments processed by the _ILECALLX or _ILECALL function is determined by the number of entries in the signature list, which is determined by the location of the first ARG_END value in the list. The following values are supported in the signature list:

<i>ARG_END</i> (0)	Specifies the end of the signature list.
<i>ARG_INT8</i> (-1)	Signed 1-byte integer argument.
<i>ARG_UINT8</i> (-2)	Unsigned 1-byte integer argument.
<i>ARG_INT16</i> (-3)	Signed 2-byte integer argument.
<i>ARG_UINT16</i> (-4)	Unsigned 2-byte integer argument.
<i>ARG_INT32</i> (-5)	Signed 4-byte integer argument.
<i>ARG_UINT32</i> (-6)	Unsigned 4-byte integer argument.
<i>ARG_INT64</i> (-7)	Signed 8-byte integer argument.
<i>ARG_UINT64</i> (-8)	Unsigned 8-byte integer argument.
<i>ARG_FLOAT32</i> (-9)	4-byte floating-point argument.
<i>ARG_FLOAT64</i> (-10)	8-byte floating-point argument.
<i>ARG_MEMPTR</i> (-11)	The argument is a field of type ILEpointer into which the caller has stored an OS/400 PASE memory address (in member address). _ILECALLX and _ILECALL convert the OS/400 PASE memory address to an equivalent teraspace address, except that address zero is converted to a special value for a null pointer. The converted result is passed as the argument value to the target ILE procedure. Both functions generally update the ILEpointer argument value in memory so it contains a tagged space pointer, but the memory may not be updated if the target ILE procedure uses ARGOPT linkage.
<i>ARG_SPCPTR</i> (-12)	The argument is a field of type ILEpointer where the OS/400 PASE program has stored a tagged space pointer (or an untagged or null pointer).

ARG_OPENPTR (-13) The argument is a field of type ILEpointer where the OS/400 PASE program has stored a 16-byte pointer of any type (including possibly an untagged or null pointer).

Any positive number (1-32767) The argument is an aggregate (structure or union). The value in the signature list is the length, in bytes, of the aggregate.

result_type

(Input) Specifies the type of function result returned by the ILE procedure.

The following values are supported:

RESULT_VOID(0) No function result.

RESULT_INT8 (-1) Signed 1-byte integer result, returned in field result.s_int8.r_int8 in the ILEarglist argument.

RESULT_UINT8 (-2) Unsigned 1-byte integer result, returned in field result.s_uint8.r_uint8 in the ILEarglist argument.

RESULT_INT16 (-3) Signed 2-byte integer result, returned in field result.s_int16.r_int16 in the ILEarglist argument.

RESULT_UINT16 (-4) Unsigned 2-byte integer result, returned in field result.s_uint16.r_uint16 in the ILEarglist argument.

RESULT_INT32 (-5) Signed 4-byte integer result, returned in field result.s_int32.r_int32 in the ILEarglist argument.

RESULT_UINT32 (-6) Unsigned 4-byte integer result, returned in field result.s_uint32.r_uint32 in the ILEarglist argument.

RESULT_INT64 (-7) Signed 8-byte integer result, returned in field result.r_int64 in the ILEarglist argument.

RESULT_UINT64 (-8) Unsigned 8-byte integer result, returned in field result.r_uint64 in the ILEarglist argument.

RESULT_FLOAT64 (-10) 8-byte floating-point result, returned in field result.r_float64 in the ILEarglist argument.

Any positive number (1-32767) The function result is an aggregate (structure or union). result_type is the length, in bytes, of the aggregate. An aggregate function result is returned in a buffer allocated by the caller and passed to the target ILE procedure using a special field in the argument list. The caller must provide a buffer large enough for the result returned by the target ILE procedure to avoid unpredictable results. An OS/400 PASE program must set field result.r_aggregate.addr in type ILEarglist_base to the OS/400 PASE memory address of the result buffer before calling an ILE procedure that returns an aggregate result. _ILECALLX and _ILECALL convert the OS/400 PASE memory address to a teraspace address the same way they convert ARG_MEMPTR arguments.

» flags

(Input) Specifies options to control how the ILE procedure program is called. The flags argument is a bitwise logical-or of one or more of the following values:

ILECALL_NOINTERRUPT (0x00000004) Specifies that OS/400 PASE signals will not interrupt the called ILE procedure. Some system functions (such as select) can be interrupted by signals. Normally either an ILE signal or an OS/400 PASE signal can interrupt such an operation, but *ILECALL_NOINTERRUPT* delays OS/400 PASE signal processing until control returns from the called ILE procedure. This option has no effect on ILE signal handling. «

Authorities

_ILECALL and *_ILECALLX* require no authority.

Return Value

Most errors from *_ILECALLX* and *_ILECALL* are reported with OS/400 exception messages that are converted to OS/400 PASE signals. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

If no OS/400 PASE signal is sent, one of these values is returned:

ILECALL_NOERROR(0) The target ILE procedure was called and returned normally.

ILECALL_INVALID_ARG (1) An invalid value was found in the signature list.

ILECALL_INVALID_RESULT (2) The result_type value is invalid.

» *ILECALL_INVALID_FLAGS* (3) The flags value is invalid. «

Usage Notes

1. » *_ILECALLX* and *_ILECALL* can only call ILE procedures in an OS/400 bound program. If your OS/400 PASE program needs to call an OS/400 program object (object type *PGM), you must use the *_PGM CALL* function or use the **systemCL** function to run the CL CALL command. «
2. Every module in a *PGM or *SRVPGM object containing a function directly called by PASE (using *_ILECALLX* or *_ILECALL*) must be Teraspace-safe. If any module in the program was created as TERASPACE(*NO), then OS/400 PASE will not be able to call any procedure in that program (even a procedure in a module created as TERASPACE(*YES)).

3. **_ILECALLX** and **_ILECALL** do no character encoding conversions, so the OS/400 PASE program may need to convert argument and result character strings between ASCII and EBCDIC. OS/400 PASE runtime function **iconv** can be used for character conversions.
4. An OS/400 PASE program can pass tagged space pointer arguments to an ILE procedure using either **ARG_SPCPTR** or **ARG_OPENPTR** unless the target ILE procedure uses **ARGOPT** linkage, in which case **ARG_SPCPTR** must be used. **ARG_MEMPTR** can be used for space pointer arguments regardless of what linkage is used by the target ILE procedure.
5. ILE procedure pointers address resources inside an ILE activation group. The machine prohibits use of activation group resources from a process other than the owner of the activation group. This means that the child process of a **fork** cannot use ILE procedure pointers inherited from the parent process. The child process can, however, use **_ILELOAD** to load the bound program (creating a new activation in the child process) and then use **_ILESYM** to obtain ILE procedure pointers into the new activation.
6. See [Set Space Pointer for OS/400 PASE](#) (**_SETSPP**) for more information about tagged space pointers and sharing tagged pointers between processes.
7. [»](#) **_ILECALL** is equivalent to **_ILECALLX** with the **ILECALL_NOINTERRUPT** flag. [«](#)

Related Information

- [» _PGMCALL\(\)--Call an OS/400 Program for OS/400 PASE](#) [«](#)
- [_ILESYM\(\)](#)--Find an Exported ILE Symbol for OS/400 PASE
- [_ILELOAD\(\)](#)--Load an ILE Bound Program for OS/400 PASE
- [size_ILEarglist\(\)](#)--Compute ILE Argument List Size for OS/400 PASE
- [build_ILEarglist\(\)](#)--Build an ILE Argument List for OS/400 PASE
- [_SETSPP\(\)](#)--Set Space Pointer for OS/400 PASE
- [» systemCL\(\)--Run a CL Command for OS/400 PASE](#) [«](#)

API Introduced: V4R5

» **_PGM_CALL()**--Call an OS/400 Program for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int _PGM_CALL(const ILEpointer *target,
              void **argv,
              unsigned flags);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_PGM_CALL()** function calls an OS/400 program (object type *PGM) from an OS/400 PASE program. It transfers control to the *PGM object specified by a 16-byte tagged system pointer (passing any necessary arguments) and resumes execution when control returns.

Parameters

target

(Input) Pointer to a tagged system pointer that addresses the OS/400 program (object type *PGM) to call. *target* must be a 16-byte aligned OS/400 PASE memory address.

argv

(Input/Output) Array of pointers to arguments. *argv* is the address of an array of pointers to argument variables that are passed by-address to the OS/400 program. *argv* can be zero (null) if there are no arguments to pass. The last element in the array must be a null pointer. A maximum of **PGM_CALL_MAXARGS** (255) arguments can be passed to an OS/400 program.

flags

(Input) Specifies options to control how the OS/400 program is called. The *flags* argument is a bitwise logical-or of one or more of the following values:

<i>PGMCALL_DIRECT_ARGS</i> (0x00000001)	Specifies that the addresses in the <i>argv</i> array are passed directly to the OS/400 program as formal arguments. If PGMCALL_DIRECT_ARGS is omitted, the system builds tagged space pointers to the argument memory locations identified in the <i>argv</i> array and passes the space pointers as formal arguments (standard OS/400 program linkage).
<i>PGMCALL_DROP_ADOPT</i> (0x00000002)	Specifies that authorizations adopted by OS/400 program invocations already in the stack are dropped so the called program only benefits from authorizations available to the user and group profiles for the thread.
<i>PGMCALL_NOINTERRUPT</i> (0x00000004)	Specifies that OS/400 PASE signals will not interrupt the called OS/400 program. Some system functions (such as select) can be interrupted by signals. Normally either an ILE signal or an OS/400 PASE signal can interrupt such an operation, but PGMCALL_NOINTERRUPT delays OS/400 PASE signal processing until control returns from the called OS/400 program. This option has no effect on ILE signal handling.

Authorities

Object Referred to	Authority Required
OS/400 program to call	*X

Return Value

Most errors from **_PGMCALL** are reported with OS/400 exception messages that are converted to OS/400 PASE signals. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

If no OS/400 PASE signal is sent, a function result of zero indicates the OS/400 program was called and returned normally. A function result of -1 indicates an error that is further qualified by an *errno* value.

Error Conditions

At least these *errno* values can be returned, with other values also possible (such as OS/400-unique ILE *errno* EAPAR):

[EINVAL] An invalid *flags* value was specified, or more than **PGMCALL_MAXARGS** (255) arguments were provided.

Usage Notes

1. `_PGMCALL` can only call OS/400 program objects (object type *PGM). If your OS/400 PASE program needs to call a particular ILE procedure inside a *PGM or *SRVPGM object, you must use the `_ILECALL` function.
2. You can use the `_RSLOBJ` or `_RSLOBJ2` function to obtain a system pointer to an OS/400 program (object type *PGM).
3. Any OS/400 program that accepts arguments must be Teraspace-safe (created using `TERASPACE(*YES)`) to be called using `_PGMCALL` because the arguments are always passed in Teraspace storage.
4. Arguments (addressed by pointers in the `argv` array) can be of any data type. Arguments are passed by-address, so the called OS/400 program can modify argument variables to return results to the OS/400 PASE program.
5. `_PGMCALL` does no character encoding conversions, so the OS/400 PASE program may need to convert argument and result character strings between ASCII and EBCDIC. OS/400 PASE runtime function `iconv` can be used for character conversions.

Related Information

- [_RSLOBJ\(\)--Resolve to an OS/400 Object for OS/400 PASE](#)
- [_ILECALL\(\)--Call an ILE Procedure for OS/400 PASE](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

size_ILEarglist()--Compute ILE Argument List Size for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

size_t size_ILEarglist(const arg_type_t *signature);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **size_ILEarglist()** function computes the number of bytes of memory required to build an ILE argument list for a specific function signature.

Parameters

signature

(Input) Pointer to a list of `arg_type_t` values that specify the sequence and type of arguments passed to the ILE procedure. ILE procedures can accept a maximum of 400 arguments. The actual number of arguments processed by the **size_ILEarglist** function is determined by the number of entries in the signature list, which is determined by the location of the first `ARG_END` value in the list. The following values are supported in the signature list:

<i>ARG_END</i> (0)	Specifies the end of the signature list.
<i>ARG_INT8</i> (-1)	Signed 1-byte integer argument.
<i>ARG_UINT8</i> (-2)	Unsigned 1-byte integer argument.
<i>ARG_INT16</i> (-3)	Signed 2-byte integer argument.
<i>ARG_UINT16</i> (-4)	Unsigned 2-byte integer argument.
<i>ARG_INT32</i> (-5)	Signed 4-byte integer argument.

<i>ARG_UINT32</i> (-6)	Unsigned 4-byte integer argument.
<i>ARG_INT64</i> (-7)	Signed 8-byte integer argument.
<i>ARG_UINT64</i> (-8)	Unsigned 8-byte integer argument.
<i>ARG_FLOAT32</i> (-9)	4-byte floating-point argument.
<i>ARG_FLOAT64</i> (-10)	8-byte floating-point argument.
<i>ARG_MEMPTR</i> (-11)	The argument is a field of type ILEpointer.
<i>ARG_SPCPTR</i> (-12)	The argument is a field of type ILEpointer.
<i>ARG_OPENPTR</i> (-13)	The argument is a field of type ILEpointer.
Any positive number (1-32767)	The argument is an aggregate (structure or union). The value in the signature list is the length, in bytes, of the aggregate.

Authorities

`size_ILEarglist` requires no authority.

Return Value

`size_ILEarglist` returns the number of bytes required to build the ILE argument list (including storage for the `ILEarglist_base` type and any necessary bytes skipped for alignment between arguments), or zero if an error was detected in the signature list.

Related Information

- [ILECALLX\(\)--Call an ILE Procedure for OS/400 PASE](#)
- [build_ILEarglist\(\)--Build an ILE Argument List for OS/400 PASE](#)

API Introduced: V4R5

_CVTERRNO()--Convert ILE errno to OS/400 PASE errno

Syntax

```
#include <as400_protos.h>

int _CVTERRNO(int  errno_ile);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information.

The `_CVTERRNO()` function converts an ILE errno value to a corresponding OS/400 PASE errno value.

Parameters

errno_ile

(Input) Specifies the ILE errno value to convert to a corresponding OS/400 PASE errno value. ILE and OS/400 PASE errno values correspond if they have the same name (for example, EFAULT) in a system-provided header file.

Authorities

`_CVTERRNO` requires no authority.

Return Value

`_CVTERRNO` returns the OS/400 PASE equivalent of the input ILE errno value. If the input has no OS/400 PASE errno equivalent (for example, EAPAR is an ILE errno value with no OS/400 PASE equivalent), the input is returned unchanged.

Usage Notes

1. The errno value set by an ILE runtime function must be determined by code running in the same thread and activation group that called the runtime function because ILE runtime sometimes maintains a separate errno variable for each activation group.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)

API Introduced: V5R1

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_CVTSPP()--Convert Space Pointer for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

void* _CVTSPP(const ILEpointer *source);
```

Default Public Authority: *USE

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information.

The **_CVTSPP()** function converts the teraspace address in a tagged space pointer to an equivalent OS/400 PASE memory address.

Parameters

source

(Input) Pointer to a tagged space pointer or 16-byte null pointer. The source address must be 16-byte aligned.

Authorities

_CVTSPP requires no authority.

Return Value

_CVTSPP returns the OS/400 PASE memory address equivalent of the input tagged space pointer. The result is zero (null) if the input is a 16-byte null pointer or a tagged space pointer that does not contain the teraspace address equivalent of some valid OS/400 PASE memory address.

Error Conditions

Any error is reported with an OS/400 exception message that the system converts to an OS/400 PASE signal. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

Usage Notes

1. `_CVTSPP` returns an OS/400 PASE memory address regardless of whether there is currently any memory at that address (as long as the input tagged pointer contains the teraspace address equivalent of a valid OS/400 PASE memory address).

Related Information

- [_SETSPP\(\)--Set Space Pointer for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_STRNCPY_SPP()--Copy Character String for OS/400 PASE

Syntax

```
#include <as400_protos.h>

void _STRNCPY_SPP(const ILEpointer *target,
                  const ILEpointer *source,
                  size_t length);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_STRNCPY_SPP()** function copies a null-terminated character string. It performs the same operation as the **strncpy** function, but uses 16-byte tagged space pointers to locate the source and target strings.

Parameters

target

(Output) Pointer to target buffer. Target is the 16-byte aligned address of a tagged space pointer to the target buffer.

source

(Input) Pointer to source string. source is the 16-byte aligned address of a tagged space pointer to the source character string.

length

(Input) Specifies the maximum number of bytes to copy between the source and target. If the source string is too long, then only the specified number of bytes are copied and the target string is not terminated with a null. If the source string is too short, the copy is padded with nulls to fill the target buffer.

Authorities

_STRNCPY_SPP requires no authority.

Error Conditions

Any error is reported with an OS/400 exception message that the system converts to an OS/400 PASE signal. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

Usage Notes

1. `_STRNCPY_SPP` can copy between any memory areas addressable through tagged space pointers, which need not be in the OS/400 PASE address space.
2. `_STRNCPY_SPP` is implemented with a kernel system call, so it generally runs slower than `strncpy`.

Related Information

- [_CVTSPP\(\)--Convert Space Pointer for OS/400 PASE](#)
- [_SETSPP\(\)--Set Space Pointer for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_MEMCPY_WT()--Copy Memory With Tags for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

void* _MEMCPY_WT(void      *target,
                 const void *source,
                 size_t     length);

void _MEMCPY_WT2(const ILEpointer *target,
                 const ILEpointer *source,
                 size_t           length);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The `_MEMCPY_WT()` and `_MEMCPY_WT2()` functions copy memory without destroying 16-byte tagged pointers.

Standard memory copy functions such as `memcpy` never produce a usable tagged pointer in the target memory. `_MEMCPY_WT` and `_MEMCPY_WT2` copy memory in a way that preserves the integrity of any complete (16-byte) tagged pointers copied, as long as the source and target have the same alignment with respect to a 16-byte boundary.

Parameters

target

(Output) Pointer to target memory. For `_MEMCPY_WT`, `target` is the OS/400 PASE address of the target memory. For `_MEMCPY_WT2`, `target` is the 16-byte aligned address of a tagged space pointer to the target memory.

source

(Input) Pointer to source memory. For `_MEMCPY_WT`, `source` is the OS/400 PASE address of the source memory. For `_MEMCPY_WT2`, `source` is the 16-byte aligned address of a tagged space pointer to the source memory.

length

(Input) Specifies the number of bytes to copy between the source and target.

Authorities

`_MEMCPY_WT` and `_MEMCPY_WT2` require no authority.

Return Value

`_MEMCPY_WT` returns the target memory address. `_MEMCPY_WT2` returns no function result.

Error Conditions

Any error is reported with an OS/400 exception message that the system converts to an OS/400 PASE signal. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

Usage Notes

1. `_MEMCPY_WT` only copies between memory areas in the OS/400 PASE address space. `_MEMCPY_WT2` can copy between any memory areas addressable through tagged space pointers, which need not be in the OS/400 PASE address space.
2. Memory is copied without error if the source and target do not have the same alignment with respect to a 16-byte boundary or if only part of a tagged pointer is copied, but the target will not contain a usable tagged pointer.
3. `_MEMCPY_WT` and `_MEMCPY_WT2` are implemented with kernel system calls, so they generally run slower than `memcpy`.

Related Information

- [_CVTSPP\(\)--Convert Space Pointer for OS/400 PASE](#)
- [_SETSPP\(\)--Set Space Pointer for OS/400 PASE](#)

API Introduced: V4R5

_STRLEN_SPP()--Determine Character String Length for OS/400 PASE

Syntax

```
#include <as400_protos.h>

size_t _STRLEN_SPP(const ILEpointer *string);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_STRLEN_SPP()** determines the length of a null-terminated character string. It performs the same operation as the **strlen** function, but uses a 16-byte tagged space pointer to locate the string.

Parameters

string

(Input) Pointer to character string. **string** is the 16-byte aligned address of a tagged space pointer to the character string.

Authorities

_STRLEN_SPP requires no authority.

Return Value

_STRLEN_SPP returns length of the character string.

Error Conditions

Any error is reported with an OS/400 exception message that the system converts to an OS/400 PASE signal. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

Usage Notes

1. `_STRLEN_SPP` can reference any memory addressable through a tagged space pointer, which need not be in the OS/400 PASE address space.
2. `_STRLEN_SPP` is implemented with a kernel system call, so it generally runs slower than `strlen`.

Related Information

- [_CVTSPP\(\)--Convert Space Pointer for OS/400 PASE](#)
- [_SETSPP\(\)--Set Space Pointer for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_ILESYM()--Find an Exported ILE Symbol for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int _ILESYM(ILEpointer *export,
            int actmark,
            const char *symbol);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_ILESYM()** function finds an exported symbol in the activation of an ILE bound program and returns a 16-byte tagged pointer to the data or procedure for the symbol.

Parameters

export

(Output) Pointer to a 16-byte aligned buffer for the tagged pointer return value. The export buffer is used to store a tagged pointer to the data or procedure for the exported symbol.

actmark

(Input) Specifies an activation mark that identifies the activation (in the current OS/400 job) to search for the symbol. A value of zero causes the system to search all activations in the activation group that started OS/400 PASE (either the activation group that called the **Qp2RunPase** API, or the default activation group for a job running program **QP2FORK**). The **_ILELOAD** function returns an activation mark when it loads a bound program.

symbol

(Input) Pointer to the symbol name to find. **symbol** is the address of a null-terminated character string in the OS/400 PASE CCSID that specifies the name of a symbol exported by the actmark activation.

Authorities

_ILESYM calls the ILE **QleGetExp** API to find the exported symbol. See [QleGetExp\(\)--Get Export](#) for information about authorities required to use **_ILESYM**.

Return Value

A function result of -1 indicates an error that is further qualified by an errno value. If the symbol was successfully found, the export pointer is set to the address of the function or data for the symbol, and the function result is set to one of these values:

- ILESYM_PROCEDURE (1)* The export return value is a tagged pointer to an ILE procedure. An ILE procedure pointer can be used with the `_ILECALLX` or `_ILECALL` function to call the ILE procedure.
- ILESYM_DATA(2)* The export return value is a tagged space pointer to a data item in the ILE activation.

Error Conditions

Memory errors and errors during ILE symbol resolution processing may be reported with an OS/400 exception message that the system converts to an OS/400 PASE signal (not return code and errno values). See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

At least these errno values can be returned, with other values also possible (such as OS/400-unique ILE errno EAPAR):

- [EACCES]* Not authorized to the actmark activation.
- [ENOENT]* The symbol was not found in the actmark activation.

Related Information

- [_ILELOAD\(\)--Load an ILE Bound Program for OS/400 PASE](#)
- [_ILECALLX\(\)--Call an ILE Procedure for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

_ILELOAD()--Load an ILE Bound Program for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int _ILELOAD(⟨⟩const void⟨⟩ *id,
             unsigned int);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_ILELOAD()** function loads a bound program into the ILE activation group associated with the procedure that started OS/400 PASE (either the activation group that called the Qp2RunPase API, or the default activation group for a job running program QP2FORK).

Parameters

⟨⟩id

(Input) Pointer to the identification of the bound program. *id* is either the address of a null-terminated character string in the OS/400 PASE CCSID that names the program, or the address of a system pointer to the program, depending on the value of the flags argument. ⟨⟩

flags



(Input) Specifies options to control how the bound program is found and activated. The flags argument is a bitwise logical-or of one or more of the following values:

ILELOAD_PATH
(0x00000000)

Specifies that the *id* argument is the address of a string that contains an absolute or relative path in the Integrated File System to a program or service program object. Alphabetic case is either ignored or honored depending on the attributes of the File System that contains the path. *ILELOAD_PATH*, *ILELOAD_LIBOBJ*, and *ILELOAD_PGMPTR* are mutually exclusive.

ILELOAD_LIBOBJ
(0x00000001)

Specifies that the *id* argument is the address of a string that contains a qualified library/object name of a service program (where omitting the library name implies resolving to the object through the job library list). Alphabetic case is honored when searching for a library/object name (so the string should be all uppercase). *ILELOAD_PATH*, *ILELOAD_LIBOBJ*, and *ILELOAD_PGMPTR* are mutually exclusive.

ILELOAD_PGMPTR (0x00000002)  Specifies that the id argument is the address of a system pointer to the bound program (object type *SRVPGM or *PGM) to load. ILELOAD_PATH, ILELOAD_LIBOBJ, and ILELOAD_PGMPTR are mutually exclusive. 

Authorities

_ILELOAD calls the ILE **QleActBndPgm** API to activate the bound program. See [QleActBndPgm\(\)--Activate Bound Program](#) for information about authorities required to use **_ILELOAD**.

Return Value

A function result of -1 indicates an error that is further qualified by an errno value. If the bound program was successfully activated (including the case where it was already activated before **_ILELOAD** ran), the function result is an activation mark that uniquely identifies the activation within the process.

Error Conditions

Memory errors and errors while activating the bound program may be reported with an OS/400 exception message that the system converts to an OS/400 PASE signal (not return code and errno values). See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

At least these errno values can be returned, with other values also possible (such as OS/400-unique ILE errno EAPAR):

<i>[EACCES]</i>	Not authorized to a library or directory needed to resolve the id.
<i>[EBUSY]</i>	A library or directory needed to resolve the specified id is currently in use (locked).
<i>[EFAULT]</i>	A memory fault occurred attempting to reference the id.
<i>[EINVAL]</i>	An invalid argument value was specified.
<i>[EINTER]</i>	An signal interrupted the operation.
<i>[ENAMETOOLONG]</i>	Some component of the specified id is too long, or the entire id exceeds the system limit.
<i>[ENOENT]</i>	No file/object was found for the specified id.
<i>[ENOTDIR]</i>	A qualifier part of the id is not a directory.

[ELOOP]

Too many levels of symbolic links.

Related Information

- [_ILECALLX\(\)--Call an ILE Procedure for OS/400 PASE](#)
- [_ILESYM\(\)--Find an Exported ILE Symbol for OS/400 PASE](#)
- [» RSLOBJ\(\)--Resolve to an OS/400 Object for OS/400 PASE«](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

SQLOverrideCCSID400()--Override SQL CLI CCSID for OS/400 PASE

Syntax

```
#include <as400_protos.h>

int SQLOverrideCCSID400(int newCCSID);
```

Library: OS/400 PASE SQL CLI Library (libdb400.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **SQLOverrideCCSID400()** function allows an OS/400 PASE program to specify a Coded Character Set Identifier (CCSID) used to convert character data arguments and results on OS/400 PASE SQL Call Level Interface (CLI) functions.

Parameters

newCCSID

(Input) Specifies the CCSID used for OS/400 PASE SQL CLI functions.

Authorities

No special authorities required.

Return Value

The function result is zero for success, or -1 for an error that is further qualified by an errno value.

Error Conditions

At least these errno values can be returned:

[EINVAL] The conversion between newCCSID and the OS/400 job default CCSID is not supported.

[ENFILE] A converter could not be opened because the maximum number of files in the system are already opened.

[EMFILE] A converter could not be opened because the maximum number of files are already opened.

Usage Notes

1. The system automatically converts character arguments and results between the CCSID of the job or database field and a CCSID used for OS/400 PASE SQL CLI functions that defaults to the OS/400 PASE CCSID value in effect when the first OS/400 PASE SQL CLI function is called. You must call `SQLOverrideCCSID400` before any other OS/400 PASE SQL CLI function, or it will have no effect on CCSID conversions.

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

QMHRVCVM()--Receive Nonprogram Message for OS/400 PASE

Syntax

```
#include <os400msg.h>

int QMHRVCVM(void          *msginfo,
              int          msginfoLen,
              const char   *format,
              const void   *msgq,
              const char   *msgtype,
              int          *msgkey,
              int          wait,
              const char   *action,
              void         *errcode);

int QMHRVCVM1(void        *msginfo,
               int        msginfoLen,
               const char  *format,
               const void  *msgq,
               const char  *msgtype,
               int         *msgkey,
               int         wait,
               const char  *action,
               void        *errcode,
               int         ccsid);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The Receive Nonprogram Message (QMHRVCVM and QMHRVCVM1) OS/400 PASE runtime functions allow an OS/400 PASE program to receive a message from a nonprogram message queue.

Parameters

These OS/400 PASE runtime functions accept the same arguments as the Receive Nonprogram Message (QMHRVCVM) OS/400 API, except that the OS/400 PASE functions use character string inputs that are null-terminated strings in the OS/400 PASE CCSID. OS/400 PASE runtime automatically converts input character strings to the job default CCSID and pads with blanks (as necessary) to match the fixed-length inputs required by the system API.

No conversions are done by OS/400 PASE runtime for the msginfo and errcode (input/output) arguments because they can contain a mixture of character and binary data. The ccsid argument specifies the CCSID for character data returned by the system API in the msginfo argument, and users can request CCSID information

for the errcode argument by using ERRRC0200 format. The QMHRCVM OS/400 PASE runtime function uses a default for the ccsid value passed to the system API that does not do any CCSID conversion for character data in the received message.

See [QMHRCVM\(\)--Receive Nonprogram Message](#) for further description of the arguments for the QMHRCVM and QMHRCVM1 OS/400 PASE runtime functions.

Authorities

See [QMHRCVM\(\)--Receive Nonprogram Message](#) for information about authorities required for the QMHRCVM and QMHRCVM1 OS/400 PASE runtime functions.

Return Value

The function result is zero for normal completion. The result is nonzero if any input character string could not be converted to the job default CCSID or was too long for the QMHRCVM API, or if the QMHRCVM API returned error information in the errcode argument.

Related Information

- [QMHRCVM\(\)--Receive Nonprogram Message](#) (system API)
- [QMHRCVM\(\)--Receive Nonprogram Message for OS/400 PASE](#)
- [QMHSNDPM\(\)--Send Program Message for OS/400 PASE](#)
- [QMHRCVPM\(\)--Receive Program Message for OS/400 PASE](#)

API Introduced: V5R1

QMHRCVPM()--Receive Program Message for OS/400 PASE

Syntax

```
#include <os400msg.h>

int QMHRCVPM(void          *msginfo,
              int          msginfoLen,
              const char   *format,
              const char   *pgmq,
              int          pgmqDelta,
              const char   *msgtype,
              int          *msgkey,
              int          wait,
              const char   *action,
              void         *errcode);

int QMHRCVPM1(void        *msginfo,
               int        msginfoLen,
               const char  *format,
               const char  *pgmq,
               int        pgmqDelta,
               const char  *msgtype,
               int        *msgkey,
               int        wait,
               const char  *action,
               void        *errcode,
               int        pgmqLen,
               const char  *pgmqQual);

int QMHRCVPM2(void        *msginfo,
               int        msginfoLen,
               const char  *format,
               const void  *pgmq,
               int        pgmqDelta,
               const char  *msgtype,
               int        *msgkey,
               int        wait,
               const char  *action,
               void        *errcode,
               int        pgmqLen,
               const char  *pgmqQual,
               const char  *pgmqType,
               int        ccsid);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The Receive Program Message (QMHRVPM, QMHRVPM1, and QMHRVPM2) OS/400 PASE runtime functions allow an OS/400 PASE program to receive a message from a program call message queue or from the job external message queue.

Parameters

These OS/400 PASE runtime functions accept the same arguments as the Receive Program Message (QMHRVPM) OS/400 API, except that the OS/400 PASE functions use character string inputs that are null-terminated strings in the OS/400 PASE CCSID. OS/400 PASE runtime automatically converts input character strings to the job default CCSID and pads with blanks (as necessary) to match the fixed-length inputs required by the system API.

No conversions are done by OS/400 PASE runtime for the msginfo and errcode (input/output) arguments because they can contain a mixture of character and binary data. The ccsid argument specifies the CCSID for character data returned by the system API in the msginfo argument, and users can request CCSID information for the errcode argument by using ERRC0200 format. The QMHRVPM and QMHRVPM1 OS/400 PASE runtime functions use a default for the ccsid value passed to the system API that does not do any CCSID conversion for character data in the received message.

See [QMHRVPM\(\)--Receive Program Message](#) for further description of the arguments for the QMHRVPM, QMHRVPM1, and QMHRVPM2 OS/400 PASE runtime functions.

Authorities

See [QMHRVPM\(\)--Receive Program Message](#) for information about authorities required for the QMHRVPM, QMHRVPM1, and QMHRVPM2 OS/400 PASE runtime functions.

Return Value

The function result is zero for normal completion. The result is nonzero if any input character string could not be converted to the job default CCSID or was too long for the QMHRVPM API, or if the QMHRVPM API returned error information in the errcode argument.

Usage Notes

1. The system only creates program call message queues ILE procedures and OMI programs, so you cannot send to or receive from a program message queue for a specific function in an OS/400 PASE program.
2. When "*" is specified for the pgmq argument, the system locates the program call message queue for an (internal) ILE procedure in service program QP2USER that is the apparent caller of any ILE procedure called by the OS/400 PASE program using OS/400 PASE runtime function _ILECALLX or _ILECALL. This queue is the target for messages a called ILE procedure sends to its caller, and is also used for machine exceptions caused by operations inside the OS/400 PASE program (such as message MCH0601 for storage reference error).

Related Information

- [QMHRVPM\(\)--Receive Program Message](#) (system API)
- [QMHRVPM\(\)--Receive Program Message for OS/400 PASE](#)
- [QMHSNDM\(\)--Send Nonprogram Message for OS/400 PASE](#)
- [QMHRVPM\(\)--Receive Nonprogram Message for OS/400 PASE](#)

API Introduced: V5R1

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

» **_RSLOBJ()--Resolve to an OS/400 Object for OS/400 PASE**

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int _RSLOBJ(ILEpointer      *sysptr,
            const char      *path,
            char             *objtype);

int _RSLOBJ2(ILEpointer      *sysptr,
             unsigned short  type_subtype,
             const char      *objname,
             const char      *libname);
```

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **_RSLOBJ()** and **_RSLOBJ2()** functions resolve to an OS/400 object. They accept symbolic information that identifies the object and return a 16-byte tagged system pointer to the specified object.

Parameters

sysptr

(Output) Pointer to the OS/400 object. *sysptr* is the address of a 16-byte aligned buffer allocated by the caller and used to return a system pointer to the OS/400 object.

path

(Input) Pointer to an Integrated File System path name that locates the OS/400 object. *path* is the address of a null-terminated string in the OS/400 PASE CCSID that contains a path name for the OS/400 object.

objtype

(Output) Pointer to the returned OS/400 object type. *objtype* is the address of a buffer allocated by the caller and used to return a null-terminated string in the OS/400 PASE CCSID that identifies the OS/400 object type. If *objtype* is a null pointer, no OS/400 object type is returned. When *objtype* is not null, the caller must provide a buffer of length **RSLOBJ_OBJTYPE_MAXLEN** (11) to avoid errors.

type_subtype

(Input) Object type and subtype. *type_subtype* specifies the MI object type and MI object subtype of the OS/400 object. Header file <as400_types.h> declares these constants for type and subtype values:

RSLOBJ_TS_PGM (0x0201) Specifies the MI type and subtype for an OS/400 program (object type *PGM).

RSLOBJ_TS_SRVPGM (0x0203) Specifies the MI type and subtype for an OS/400 service program (object type *SRVPGM).

objname

(Input) Pointer to the name of the OS/400 object. *objname* is the address of a null-terminated string in the OS/400 PASE CCSID that contains the name of the OS/400 object.

libname

(Input) Pointer to the name of the OS/400 library that contains the object. *libname* is the address of a null-terminated string in the OS/400 PASE CCSID that contains the name of an OS/400 library. Specifying a null pointer or a pointer to a null string is the same as specifying "*LIBL", which searches the thread library list.

Authorities

Object Referred to	Authority Required
Every directory in the Integrated File System path to the OS/400 object	*X
OS/400 library that contains the object	*X

Return Value

The function result is zero if the OS/400 object was found and a system pointer was returned in the *sysptr* argument. A function result of -1 indicates an error that is further qualified by an *errno* value.

Error Conditions

Memory errors may be reported with an OS/400 exception message that the system converts to an OS/400 PASE signal (not return code and *errno* values). See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

At least these *errno* values can be returned, with other values also possible (such as OS/400-unique ILE *errno* EAPAR):

<i>[EACCES]</i>	Not authorized to a library or directory needed to resolve to the OS/400 object.
<i>[EBUSY]</i>	A library or directory needed to resolve to the OS/400 object is currently in use (locked).
<i>[EFAULT]</i>	A memory fault occurred attempting to reference an argument.
<i>[EINVAL]</i>	An invalid argument value was specified.
<i>[EINTER]</i>	An signal interrupted the operation.
<i>[ENAMETOOLONG]</i>	Some component of the specified <i>path</i> is too long, or the entire <i>path</i> exceeds the system limit, or the <i>objname</i> or <i>libname</i> string is longer than 30 characters.
<i>[ENOENT]</i>	The specified OS/400 object was not found.
<i>[ENOTDIR]</i>	A qualifier part of the <i>path</i> is not a directory.
<i>[ELOOP]</i>	Too many levels of symbolic links.

Usage Notes

1. For **_RSLOBJ**, alphabetic case is either ignored or honored depending on the attributes of the file system that contains the path. Alphabetic case is always honored by **_RSLOBJ2**, so the *objname* and *libname* strings must be uppercase.

Related Information

- [_ILELOAD\(\)--Load an ILE Bound Program for OS/400 PASE](#)
- [_PGMCALL\(\)--Call an OS/400 Program for OS/400 PASE](#)



API Introduced: V5R2

» **_RETURN()--Return Without Exiting OS/400 PASE**

Syntax

```
#include <as400_protos.h>

int _RETURN(void);
```

Default Public Authority: *USE

Library: Standard C Library (libc.a)

Threadsafe: No

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The `_RETURN()` function returns to the ILE caller that called OS/400 PASE in this job, without exiting the OS/400 PASE program. OS/400 PASE remains active in the job, so APIs `Qp2CallPase` and `Qp2CallPase2` can be used to call procedures in the OS/400 PASE program.

Parameters

None.

Authorities

None.

Return Value

`_RETURN` does not return to the OS/400 PASE program if it successfully returns to the ILE caller. A function result of -1 with an `errno` is returned for any error.

Error Conditions

EPERM is set if `_RETURN` is used in a fork child process or in an OS/400 PASE program that is currently running multiple threads.

Usage Notes

1. The system provides two OS/400 PASE programs, /usr/lib/start32 (for 32-bits) and /usr/lib/start64 (for 64-bits), that return without exiting immediately after initializing the standard C library (libc.a) and pthreads library (libpthreads.a).
2. The system ends any OS/400 PASE program when it destroys the activation group that called API Qp2RunPase. API program QP2SHELL always calls Qp2RunPase in a *NEW activation group that is destroyed before return, so QP2SHELL is not useful for running an OS/400 PASE program that returns without exiting.
3. You may need to call ILE API Qp2EndPase to end an OS/400 PASE program that uses **_RETURN**. See [Qp2EndPase\(\)--End an OS/400 PASE Program](#) for more information.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)
- [Qp2EndPase\(\)--End an OS/400 PASE Program](#)



API Introduced: V5R2

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

systemCL()--Run a CL Command for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

int systemCL(const char *command,
             int flags);
```

Library: Standard C Library (libc.a)

Threadsafe: Conditional

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The **systemCL()** function runs a CL command.

Parameters

command

(Input) Pointer to a null-terminated string in the OS/400 PASE CCSID that specifies the CL command with any parameters.

flags

(Input) Specifies option flags that control how the CL command runs. *flags* is a bit-wise OR of any of the following values:

SYSTEMCL_MSG_STDOUT (0x00000001)

Directs the system to receive OS/400 messages after normal command completion, convert the text of each message from the job default CCSID to the OS/400 PASE CCSID, and write converted text lines to Integrated File System descriptor 1 (stdout).

SYSTEMCL_MSG_STDERR (0x00000002)

Directs the system to receive OS/400 messages after error command completion, convert the text of each message from the job default CCSID to the OS/400 PASE CCSID, and write converted text lines to Integrated File System descriptor 2 (stderr).

<i>SYSTEMCL_MSG_NOMSGID</i> (0x00000004)	Suppresses message identifiers in text lines written to stdout or stderr for messages processed on behalf of <i>SYSTEMCL_MSG_STDOUT</i> and <i>SYSTEMCL_MSG_STDERR</i> . When this option is omitted, message text lines have the form "XXX1234: message text", where "XXX1234" is the OS/400 message identifier.
<i>SYSTEMCL_SPOOL_STDOUT</i> (0x00000008)	Directs the system to process any spooled output files created by the CL command by reading each file, converting file data from the job default CCSID to the OS/400 PASE CCSID, and writing converted text lines to Integrated File System descriptor 1 (stdout).
<i>SYSTEMCL_SPOOL_KEEP</i> (0x00000010)	Directs the system to keep any spooled output files after they are processed for option <i>SYSTEMCL_SPOOL_STDOUT</i> , instead of deleting the files after their contents is written to stdout.
<i>SYSTEMCL_FILTER_STDIN</i> (0x00000020)	Directs the system to setup a filter thread that converts from the OS/400 PASE CCSID to the job default CCSID for any data the CL command reads from Integrated File System descriptor 0 (stdin).
<i>SYSTEMCL_FILTER_STDOUT</i> (0x00000040)	Directs the system to setup a filter thread that converts any data the CL command writes to Integrated File System descriptor 1 (stdout) from the job default CCSID to the OS/400 PASE CCSID.
<i>SYSTEMCL_FILTER_STDERR</i> (0x00000080)	Directs the system to setup a filter thread that converts any data the CL command writes to Integrated File System descriptor 2 (stderr) from the job default CCSID to the OS/400 PASE CCSID.
<i>SYSTEMCL_SPAWN</i> (0x00000100)	Directs the system to run the CL command in a separate process. If this option is omitted, the CL command runs in the process that calls the <i>systemCL</i> function.
<i>SYSTEMCL_SPAWN_JOBLOG</i> (0x00000200)	Forces the system to generate an OS/400 job log for the job submitted using option <i>SYSTEMCL_SPAWN</i> .

SYSTEMCL_ENVIRON (0x00000400)

Directs the system to copy OS/400 PASE environment variables to ILE environment variables before running the CL command. This option sets ILE environment variables in the process that calls the `systemCL` function, regardless of whether the command runs in this process or a child process (created for option `SYSTEMCL_SPAWN`).

Authorities

No authority is needed to run the `systemCL` function, but the caller must be authorized to run the specified CL command.

Return Value

If the command argument is a null pointer, the function result is zero if system support to call the OS/400 Command Analyzer is available, or a nonzero value otherwise.

If option `SYSTEMCL_SPAWN` is specified, the function result is the exit code from the spawned job (returned by the ILE `waitpid` function), which is non-zero if any error occurred.

Otherwise, the function result is zero for normal command completion, or -1 if an error occurred. No `errno` value is set for CL command errors.

Usage Notes

1. `systemCL` is only threadsafe in these two cases:
 - You use option `SYSTEMCL_SPAWN` and do not use `SYSTEMCL_ENVIRON`.
 - You only run threadsafe CL commands and do not use `SYSTEMCL_SPAWN`, `SYSTEMCL_FILTER_STDIN`, `SYSTEMCL_FILTER_STDOUT`, `SYSTEMCL_FILTER_STDERR`, or `SYSTEMCL_ENVIRON`.
2. You must set ILE environment variable `QIBM_USE_DESCRIPTOR_STDIO` to Y or I before the CL command does any file I/O to `stdin`, `stdout`, or `stderr` if you need CCSID conversion controlled by options `SYSTEMCL_FILTER_STDIN`, `SYSTEMCL_FILTER_STDOUT`, and `SYSTEMCL_FILTER_STDERR`.
3. Processing for options `SYSTEMCL_FILTER_STDIN`, `SYSTEMCL_FILTER_STDOUT`, and `SYSTEMCL_FILTER_STDERR` creates ILE pthreads (not OS/400 PASE threads) for CCSID conversion in the process that calls the `systemCL` function. Integrated File System descriptors 0, 1, and 2 are replaced in whatever job runs the CL command with pipes handled by the filter threads. The original file descriptors are restored and the filter threads are ended before the `systemCL` function returns.

4. Many CL commands are not supported in a job with multiple threads. Processing for **SYSTEMCL_SPAWN** runs the CL command in a job that is not multithread-capable, so it can run commands that do not work in a job that is multithread-capable.
5. Processing for option **SYSTEMCL_SPAWN** uses the ILE **spawn** API to run a batch job that inherits ILE attributes such as Integrated File System descriptors and job CCSID, but the batch job does not inherit any OS/400 PASE program (unlike a job created by the OS/400 PASE **fork** function).
6. Processing for **SYSTEMCL_ENVIRON** uses the same name for the ILE copy and the OS/400 PASE environment variable for most variables, but the system adds a prefix "PASE_" to the name of the ILE copy of some environment variables. You can control what variables names add the prefix by storing a colon-delimited list of variable names in OS/400 PASE environment variable **PASE_ENVIRON_CONFLICT**. If **PASE_ENVIRON_CONFLICT** is not defined, the system defaults to adding the prefix when copying OS/400 PASE environment variables **SHELL**, **PATH**, **NLSPATH**, and **LANG**.
7. Processing for **SYSTEMCL_ENVIRON** sets two ILE environment variables for each OS/400 PASE environment variable with a name prefix of "ILE_". The OS/400 PASE environment variable value is used to set both a variable with the same name and a variable with the name minus the prefix "ILE_" in the ILE environment. For example, if the OS/400 PASE environment contains a variable named **ILE_PATH**, the value of this variable is used to set both **ILE_PATH** and **PATH** in the ILE environment.

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

QMHSNDM()--Send Nonprogram Message for OS/400 PASE

Syntax

```
#include <os400msg.h>

int QMHSNDM(const char *msgid,
            const char *msgf,
            const void *msgdata,
            int msgdataLen,
            const char *msgtype,
            const char *msgqList,
            int msgqCount,
            const char *rpyq,
            int *msgkey,
            void *errcode);

int QMHSNDM1(const char *msgid,
             const char *msgf,
             const void *msgdata,
             int msgdataLen,
             const char *msgtype,
             const char *msgqList,
             int msgqCount,
             const char *rpyq,
             int *msgkey,
             void *errcode,
             int ccsid);
```

Public Default Authority: *USE

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The Send Nonprogram Message (QMHSNDM and QMHSNDM1) OS/400 PASE runtime functions allow an OS/400 PASE program to send a message to a nonprogram message queue so it can communicate with another job or user.

Parameters

These OS/400 PASE runtime functions accept the same arguments as the Send Nonprogram Message (QMHSNDM) OS/400 API, except that the OS/400 PASE functions use character string inputs that are null-terminated strings in the OS/400 PASE CCSID. OS/400 PASE runtime automatically converts input character strings to the job default CCSID and pads with blanks (as necessary) to match the fixed-length inputs required by the system API.

No conversions are done for the *msgdata* (input) argument and the *errcode* (input/output) argument because they can contain a mixture of character and binary data. The *ccsid* argument specifies the CCSID for character data in the *msgdata* argument, and users can request CCSID information for the *errcode* argument by using ERRC0200 format. The QMHSNDM OS/400 PASE runtime function uses the current OS/400 PASE CCSID as a default for the *ccsid* value passed to the system API.

See [QMHSNDM\(\)--Send Nonprogram Message](#) for further description of the arguments for the QMHSNDM and QMHSNDM1 OS/400 PASE runtime functions.

Authorities

See [QMHSNDM\(\)--Send Nonprogram Message](#) for information about authorities required for the QMHSNDM and QMHSNDM1 OS/400 PASE runtime functions.

Return Value

The function result is zero for normal completion. The result is nonzero if any input character string could not be converted to the job default CCSID or was too long for the QMHSNDM API, or if the QMHSNDM API returned error information in the *errcode* argument.

Related Information

- [QMHSNDM\(\)--Send Nonprogram Message](#) (system API)
- [QMHRVCVM\(\)--Receive Nonprogram Message for OS/400 PASE](#)
- [QMHSNDPM\(\)--Send Program Message for OS/400 PASE](#)
- [QMHRVCVPM\(\)--Receive Program Message for OS/400 PASE](#)

API Introduced: V5R1

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

QMHSNDPM()--Send Program Message for OS/400 PASE

Syntax

```
#include <os400msg.h>
```

```
int QMHSNDPM(const char    *msgid,  
             const char    *msgf,  
             const void    *msgdata,  
             int           msgdataLen,  
             const char    *msgtype,  
             const char    *pgmq,  
             int           pgmqDelta,  
             int           *msgkey,  
             void          *errcode);
```

```
int QMHSNDPM1(const char    *msgid,  
              const char    *msgf,  
              const void    *msgdata,  
              int           msgdataLen,  
              const char    *msgtype,  
              const char    *pgmq,  
              int           pgmqDelta,  
              int           *msgkey,  
              void          *errcode,  
              int           pgmqLen,  
              const char    *pgmqQual,  
              int           extWait);
```

```
int QMHSNDPM2(const char    *msgid,  
              const char    *msgf,  
              const void    *msgdata,  
              int           msgdataLen,  
              const char    *msgtype,  
              const void    *pgmq,  
              int           pgmqDelta,  
              int           *msgkey,  
              void          *errcode,  
              int           pgmqLen,  
              const char    *pgmqQual,  
              int           extWait,  
              const char    *pgmqType,  
              int           ccsid);
```

Library: Standard C Library (libc.a)

Default Public Authority: *USE

Threadsafe: Yes

Note: These functions can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The Send Program Message (QMHSNDPM, QMHSNDPM1, and QMHSNDPM2) OS/400 PASE runtime functions allow an OS/400 PASE program to send a message to a program call message queue or to the job external message queue.

Parameters

These OS/400 PASE runtime functions accept the same arguments as the Send Program Message (QMHSNDPM) OS/400 API, except that the OS/400 PASE functions use character string inputs that are null-terminated strings in the OS/400 PASE CCSID. OS/400 PASE runtime automatically converts input character strings to the job default CCSID and pads with blanks (as necessary) to match the fixed-length inputs required by the system API.

No conversions are done for the *msgdata* (input) argument and the *errcode* (input/output) argument because they can contain a mixture of character and binary data. The *ccsid* argument specifies the CCSID for character data in the *msgdata* argument, and users can request CCSID information for the *errcode* argument by using ERRC0200 format. The QMHSNDPM and QMHSNDPM1 OS/400 PASE runtime functions use the current OS/400 PASE CCSID as a default for the *ccsid* value passed to the system API.

See [QMHSNDPM\(\)--Send Program Message](#) for further description of the arguments for the QMHSNDPM, QMHSNDPM1, and QMHSNDPM2 OS/400 PASE runtime functions.

Authorities

See [QMHSNDPM\(\)--Send Program Message](#) for information about authorities required for the QMHSNDPM, QMHSNDPM1, and QMHSNDPM2 OS/400 PASE runtime functions.

Return Value

The function result is zero for normal completion. The result is nonzero if any input character string could not be converted to the job default CCSID or was too long for the QMHSNDPM API, or if the QMHSNDPM API returned error information in the *errcode* argument.

Usage Notes

1. The system only creates program call message queues ILE procedures and OMI programs, so you cannot send to or receive from a program message queue for a specific function in an OS/400 PASE program.
2. When "*" is specified for the *pgmq* argument, the system locates the program call message queue for an (internal) ILE procedure in service program QP2USER that is the apparent caller of any ILE procedure called by the OS/400 PASE program using OS/400 PASE runtime function _ILECALLX or _ILECALL. OS/400 PASE programs should generally use "*PGMBDY" or "*CTLBDY" instead of "*" to send messages to their caller because a variable number of program call message queues can exist between the queue identified by *pgmq* "*" and the queue for the ILE API that called the OS/400

PASE program.

Related Information

- [QMHSNDPM\(\)--Send Program Message](#) (system API)
- [QMHRVPM\(\)--Receive Program Message for OS/400 PASE](#)
- [QMHSNDM\(\)--Send Nonprogram Message for OS/400 PASE](#)
- [QMHRVVM\(\)--Receive Nonprogram Message for OS/400 PASE](#)

API Introduced: V5R1

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

`_SETCCSID()`--Set OS/400 PASE CCSID

Syntax

```
#include <as400_protos.h>

int _SETCCSID(int  ccsid);
```

Default Public Authority: *USE

Library: Standard C Library (libc.a)

Threadsafe: No

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The `_SETCCSID()` function returns the previous value of the OS/400 PASE Coded Character Set Identifier (CCSID) and optionally sets a new OS/400 PASE CCSID.

Parameters

ccsid

(Input) Specifies the new OS/400 PASE CCSID value, or -1 to retrieve the current OS/400 PASE CCSID without changing it. An OS/400 PASE CCSID must be either a single-byte ASCII encoding that the ILE version of **iconv** can convert to and from the job default CCSID, or 1208 for UTF-8 encoding.

Authorities

`_SETCCSID` requires no authority.

Return Value

`_SETCCSID` returns either the original OS/400 PASE CCSID (before it was changed), or -1 if an error occurred and the OS/400 PASE CCSID was left unchanged.

Error Conditions

The only error condition that causes a function result of -1 is that the new *ccsid* cannot be converted to or from the OS/400 job default CCSID.

Usage Notes

1. The initial OS/400 PASE CCSID value is specified as a parameter on the Qp2RunPase API. The OS/400 PASE CCSID has two primary uses:
 - It is used to set the the CCSID attribute of any bytestream file created in the Integrate File System by an OS/400 PASE program.
 - It is used by OS/400 PASE runtime functions to convert character arguments and results between the OS/400 PASE CCSID and whatever encoding is required by the OS/400 service used to implement the function.
2. The OS/400 PASE CCSID should generally be the CCSID equivalent of the code set for the current locale. See [OS/400 PASE Locales](#) to determine what locales are supported by OS/400 PASE.
3. Character arguments and results for OS/400 PASE runtime functions that use OS/400 services are almost always automatically converted using the OS/400 PASE CCSID. For example, the name of a bytestream file passed to the OS/400 PASE open function is converted from the OS/400 PASE CCSID to the internal encoding required by the OS/400 Integrated File System.
4. Any data an OS/400 PASE program writes to or reads from a file descriptor for an open bytestream file, socket, FIFO, or pipe is generally *not* converted. The only exception is for the initial file descriptors 0, 1, and 2 provided when the Qp2RunPase API is called to start an OS/400 PASE program, which default to converting file data between the OS/400 PASE CCSID and the job default CCSID (see [Run an OS/400 PASE Program](#) (Qp2RunPase) for more information).
5. Other than special support for file descriptors 0, 1, and 2, OS/400 PASE runtime does no CCSID conversion of file data. This differs from ILE runtime, which does CCSID conversion between the file CCSID and job default CCSID for any file opened in text mode. OS/400 PASE runtime sets the CCSID attribute of any file it creates to the OS/400 PASE CCSID, so an ILE program that uses text mode to open an ASCII file created by an OS/400 PASE program can read and write EBCDIC data.
6. The OS/400 PASE runtime functions cstoccsid and ccsidtoocs convert between AIX Character Set names and CCSID values.

Related Information

- [Qp2RunPase\(\)--Run an OS/400 PASE Program](#)

API Introduced: V4R5

`_SETSPP()`--Set Space Pointer for OS/400 PASE

Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

void _SETSPP(ILEpointer *target,
             const void *memory);
```

Default Public Authority: *USE

Library: Standard C Library (libc.a)

Threadsafe: Yes

Note: This function can only be used in an OS/400 PASE program. See [OS/400 PASE](#) for more information about creating OS/400 PASE programs.

The `_SETSPP()` function sets a tagged space pointer to the teraspace equivalent of an OS/400 PASE memory address.

Parameters

target

(Output) Pointer to a 16-byte aligned buffer where the tagged space pointer (or null pointer) is returned.

memory

(Input) Pointer containing either an OS/400 PASE memory address, or a null pointer (zero).

Authorities

`_SETSPP` requires no authority.

Return Value

`_SETSPP` returns no function result. A tagged space pointer or 16-byte null pointer is returned in the *target* buffer.

Error Conditions

Any error is reported with an OS/400 exception message that the system converts to an OS/400 PASE signal. See [OS/400 PASE Signal Handling](#) for information about handling OS/400 exceptions.

Usage Notes

1. `_SETSPP` returns a 16-byte null pointer if the input OS/400 PASE *memory* address is null (zero) or if a 64-bit *memory* value points to a location that cannot contain OS/400 PASE memory. OS/400 PASE memory is allocated from teraspace, but teraspace has a limited capacity smaller than 64-bits, so OS/400 PASE can only provide addressability to a subset of a 64-bit address space.
2. `_SETSPP` returns a *target* space pointer regardless of whether there is currently any memory at the OS/400 PASE *memory* address. The tagged space pointer returned by `_SETSPP` can reference any memory later mapped to the OS/400 PASE *memory* address, until the OS/400 PASE program either calls `exec` or ends.
3. A tagged space pointer to a teraspace location must only be used by the process that owns the teraspace, although the current system implementation does not reliably enforce this restriction. Applications must not assume that a process can reference memory in the teraspace of another process because future system implementations may make this impossible. Tagged space pointers to teraspace memory that were either inherited by the child process of a fork or stored in shared memory by another process should be considered unusable.
4. Tagged (16-byte) pointers must not be stored in memory mapped from a bytestream file (by either `mmap` or `shmat`, although the current system implementation does not reliably enforce this restriction. Tagged pointers can be stored in shared memory objects (created by `shmget` and mapped by `shmat`), but a tagged space pointer to teraspace memory cannot be reliably used by a process other than the one that owns the teraspace.

Related Information

- [_CVTSPP\(\)--Convert Space Pointer for OS/400 PASE](#)

API Introduced: V4R5

[Top](#) | [OS/400 PASE APIs](#) | [APIs by category](#)

OS/400 PASE Runtime Libraries

OS/400 PASE runtime supports a large subset of the interfaces provided by AIX runtime. Most runtime interfaces supported by OS/400 PASE provide the same options and behavior as AIX. The latest information about what AIX runtime interfaces are supported by OS/400 PASE can found at the [PartnerWorld for Developers, iSeries](#) web site.

OS/400 PASE interfaces for Structured Query Language (SQL) Call Level Interface (CLI) are somewhat different from any AIX database. OS/400 PASE library **libdb400.a** handles (ASCII/EBCDIC) character encoding conversions, but supports only the options and behaviors provided by DB2 Universal Database for iSeries. An OS/400 PASE program that uses SQL CLI must compile using OS/400 header file **sqlcli.h**. See [OS/400 PASE](#) for more information.

OS/400 PASE runtime includes the following libraries, installed (as symbolic links) in /usr/lib. See [AIX documentation](#) for information about most of the interfaces exported by these libraries, DB2 Universal Database for iSeries documentation for information about SQL CLI interfaces, and [OS/400 PASE APIs](#) for information about interfaces that are unique to OS/400 PASE:

Library	Description
libbsd.a	BSD UNIX(TM) equivalence runtime
libc.a	C runtime
libC.a	C++ runtime
libc128.a	C 128-bit (type long double) runtime
libC128.a	C++ 128-bit (type long double) runtime
libcrypt.a	C runtime cryptographic interfaces
libcur.a	AIX legacy Curses library
libdb400.a	DB2 Universal Database SQL CLI runtime
libdbm.a	New Database Manager (NDBM) interfaces
libdbx.a	dbx (debugger) utility support
libdl.a	Dynamic load runtime
libg.a	Debug support
libgaimisc.a	Internal X Windows support
libgair4.a	Internal X Windows support
libi18n.a	Internationalization runtime
libICE.a	Inter-Client Exchange library
libiconv.a	Character conversion runtime

libIM.a	Input method library
» libl.a	lex support«
» libld.a	Object File Access Routine library«
» libm.a	IEEE Math library«
libMrm.a	Motif Runtime library for UIL
libpthdebug.a	Threads debug support
libpthread.a	Threads runtime
libpthread_compat.a	Old threads compatability
» libPW.a	Programmers Workbench library«
librtl.a	Runtime linking runtime
libSM.a	X Session Management library
libUil.a	Motif User Interface Language library
» libxcurses.a	Curses library«
libX11.a	C interface for the X Window System protocol
libXaw.a	Athena Widget Set
libXext.a	Interfaces to X windows extensions
» libXi.a	X Windows input processing«
libxf90_r.a	FORTTRAN runtime
libxfpthrds_compat.a	Old FORTRAN threads compatability
libxlomp_ser.a	Open mp (multi-processing) support
libxlomp.a	Symmetric mp (multiprocessing) support
libXm.a	Motif widget library
libXmu.a	Miscellaneous X Windows utility functions
» libXtst.a	X Windows testing support«
libXt.a	X Toolkit Intrinsic
» liby.a	yacc support«

OS/400 PASE Locales

OS/400 PASE includes a subset of the locales provided by AIX, supporting both 32-bit and 64-bit applications. OS/400 PASE locales are installed as symbolic links in directory /usr/lib/nls/loc.

The full name of any OS/400 PASE locale includes a code set name, which equates to the Coded Character Set Identifier (CCSID) shown in the table. Some locales also have a short name that exclude the code set part of the name. Any locale with a name ending in "@euro" uses the Euro as the currency symbol.

➤ Most OS/400 PASE locales are shipped with OS/400 language feature codes. Only locales in the base *CODE load and locales for installed language feature codes will exist on a particular OS/400 system.

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
*CODE	be_BY	be_BY.ISO8859-5	Byelorussian	Byelorussian SSR	915
	BE_BY	BE_BY.UTF-8	Byelorussian	Byelorussian SSR	1208
	ET_EE	ET_EE.UTF-8	Estonian	Estonia	1208
	UK_UA	UK_UA.UTF-8	Ukrainian	Ukraine	1208
2903	LT_LT	LT_LT.UTF-8	Lithuanian	Lithuania	1208
2904	LV_LV	LV_LV.UTF-8	Latvian	Latvia	1208
2905	VI_VN	VI_VN.UTF-8	Vietnamese	Vietnam	1208
2911	sl_SI	sl_SI.ISO8859-2	Slovene	Slovenia	912
	SL_SI	SL_SI.UTF-8	Slovene	Slovenia	1208
2912	hr_HR	hr_HR.ISO8859-2	Croatian	Croatia	912
	HR_HR	HR_HR.UTF-8	Croatian	Croatia	1208
2913	mk_MK	mk_MK.ISO8859-5	Macedonian	Macedonia	915
	MK_MK	MK_MK.UTF-8	Macedonian	Macedonia	1208
2914	sh_SP	sh_SP.ISO8859-2	Serbian Latin	Yugoslavia	912
	sh_YU	sh_YU.ISO8859-2	Serbian Latin	Yugoslavia	912
	sr_YU	sr_YU.ISO8859-2	Serbian	Yugoslavia	912
	SH_SP	SH_SP.UTF-8	Serbian Latin	Yugoslavia	1208
	SH_YU	SH_YU.UTF-8	Serbian Latin	Yugoslavia	1208
	SR_YU	SR_YU.UTF-8	Serbian	Yugoslavia	1208
	sr_SP	sr_SP.ISO8859-5	Serbian Latin	Yugoslavia	915

	SR_SP	SR_SP.UTF-8	Serbian Latin	Yugoslavia	1208
2922	pt_PT	pt_PT.ISO8859-1	Portuguese	Portugal	819
		pt_PT.IBM-1252	Portuguese	Portugal	1252
		pt_PT.IBM-1252@euro	Portuguese	Portugal	1252
		pt_PT.8859-15	Portuguese	Portugal	923
		pt_PT.8859-15@euro	Portuguese	Portugal	923
	PT_PT	PT_PT.UTF-8	Portuguese	Portugal	1208
		PT_PT.UTF-8@euro	Portuguese	Portugal	1208
2923	nl_NL	nl_NL.ISO8859-1	Dutch	Netherlands	819
		nl_NL.IBM-1252	Dutch	Netherlands	1252
		nl_NL.IBM-1252@euro	Dutch	Netherlands	1252
		nl_NL.8859-15	Dutch	Netherlands	923
		nl_NL.8859-15@euro	Dutch	Netherlands	923
	NL_NL	NL_NL.UTF-8	Dutch	Netherlands	1208
		NL_NL.UTF-8@euro	Dutch	Netherlands	1208
2924	en_AU	en_AU.8859-15	English	Australia	923
	EN_AU	EN_AU.UTF-8	English	Australia	1208
	en_BE	en_BE.8859-15	English	Belgium	923
		en_BE.8859-15@euro	English	Belgium	923
	EN_BE	EN_BE.UTF-8	English	Belgium	1208
		EN_BE.UTF-8@euro	English	Belgium	1208
	en_CA	en_CA.8859-15	English	Canada	923
	EN_CA	EN_CA.UTF-8	English	Canada	1208
	en_GB	en_GB.ISO8859-1	English	Great Britain	819
		en_GB.IBM-1252	English	Great Britain	1252
		en_GB.IBM-1252@euro	English	Great Britain	1252
		en_GB.8859-15	English	Great Britain	923
		en_GB.8859-15@euro	English	Great Britain	923

	EN_GB	EN_GB.UTF-8	English	Great Britain	1208
	en_IE	en_IE.8859-15	English	Ireland	923
		en_IE.8859-15@euro	English	Ireland	923
	EN_IE	EN_IE.UTF-8	English	Ireland	1208
		EN_IE.UTF-8@euro	English	Ireland	1208
	en_IN	en_IN.8859-15	English	India	923
	EN_IN	EN_IN.UTF-8	English	India	1208
	en_NZ	en_NZ.8859-15	English	New Zealand	923
	EN_NZ	EN_NZ.UTF-8	English	New Zealand	1208
	en_US	en_US.ISO8859-1	English	United States	819
		en_US.8859-15	English	United States	923
	EN_US	EN_US.UTF-8	English	United States	1208
	en_ZA	en_ZA.8859-15	English	South Africa	923
	EN_ZA	EN_ZA.UTF-8	English	South Africa	1208
	HI_IN	HI_IN.UTF-8	Hindi	India	1208
2925	fi_FI	fi_FI.ISO8859-1	Finnish	Finland	819
		fi_FI.IBM-1252	Finnish	Finland	1252
		fi_FI.IBM-1252@euro	Finnish	Finland	1252
		fi_FI.8859-15	Finnish	Finland	923
		fi_FI.8859-15@euro	Finnish	Finland	923
	FI_FI	FI_FI.UTF-8	Finnish	Finland	1208
		FI_FI.UTF-8@euro	Finnish	Finland	1208
2926	da_DK	da_DK.ISO8859-1	Danish	Denmark	819
		da_DK.8859-15	Danish	Denmark	923
	DA_DK	DA_DK.UTF-8	Danish	Denmark	1208
2928	fr_FR	fr_FR.ISO8859-1	French	France	819
		fr_FR.IBM-1252	French	France	1252
		fr_FR.IBM-1252@euro	French	France	1252

		fr_FR.8859-15	French	France	923
		fr_FR.8859-15@euro	French	France	923
	FR_FR	FR_FR.UTF-8	French	France	1208
		FR_FR.UTF-8@euro	French	France	1208
2929	de_AT	de_AT.8859-15	German	Austria	923
		de_AT.8859-15@euro	German	Austria	923
	DE_AT	DE_AT.UTF-8	German	Austria	1208
		DE_AT.UTF-8@euro	German	Austria	1208
	de_DE	de_DE.ISO8859-1	German	Germany	819
		de_DE.IBM-1252	German	Germany	1252
		de_DE.IBM-1252@euro	German	Germany	1252
		de_DE.8859-15	German	Germany	923
		de_DE.8859-15@euro	German	Germany	923
	DE_DE	DE_DE.UTF-8	German	Germany	1208
		DE_DE.UTF-8@euro	German	Germany	1208
	2931	ca_ES	ca_ES.ISO8859-1	Catalan	Spain
		ca_ES.IBM-1252	Catalan	Spain	1252
		ca_ES.IBM-1252@euro	Catalan	Spain	1252
		ca_ES.8859-15	Catalan	Spain	923
		ca_ES.8859-15@euro	Catalan	Spain	923
CA_ES		CA_ES.UTF-8	Catalan	Spain	1208
		CA_ES.UTF-8@euro	Catalan	Spain	1208
es_AR		es_AR.8859-15	Spanish	Argentina	923
ES_AR		ES_AR.UTF-8	Spanish	Argentina	1208
es_CL		es_CL.8859-15	Spanish	Chile	923
ES_CL		ES_CL.UTF-8	Spanish	Chile	1208
es_CO		es_CO.8859-15	Spanish	Columbia	923
ES_CO		ES_CO.UTF-8	Spanish	Columbia	1208

	es_ES	es_ES.ISO8859-1	Spanish	Spain	819
		es_ES.IBM-1252	Spanish	Spain	1252
		es_ES.IBM-1252@euro	Spanish	Spain	1252
		es_ES.8859-15	Spanish	Spain	923
		es_ES.8859-15@euro	Spanish	Spain	923
	ES_ES	ES_ES.UTF-8	Spanish	Spain	1208
		ES_ES.UTF-8@euro	Spanish	Spain	1208
	es_MX	es_MX.8859-15	Spanish	Mexico	923
	ES_MX	ES_MX.UTF-8	Spanish	Mexico	1208
	es_PE	es_PE.8859-15	Spanish	Peru	923
	ES_PE	ES_PE.UTF-8	Spanish	Peru	1208
	es_PR	es_PR.8859-15	Spanish	Paraguay	923
	ES_PR	ES_PR.UTF-8	Spanish	Paraguay	1208
	es_UY	es_UY.8859-15	Spanish	Uruguay	923
	ES_UY	ES_UY.UTF-8	Spanish	Uruguay	1208
	es_VE	es_VE.8859-15	Spanish	Venezuela	923
	ES_VE	ES_VE.UTF-8	Spanish	Venezuela	1208
2932	it_IT	it_IT.ISO8859-1	Italian	Italy	819
		it_IT.IBM-1252	Italian	Italy	1252
		it_IT.IBM-1252@euro	Italian	Italy	1252
		it_IT.8859-15	Italian	Italy	923
		it_IT.8859-15@euro	Italian	Italy	923
	IT_IT	IT_IT.UTF-8	Italian	Italy	1208
		IT_IT.UTF-8@euro	Italian	Italy	1208
2933	no_NO	no_NO.ISO8859-1	Norwegian	Norway	819
		no_NO.8859-15	Norwegian	Norway	923
	NO_NO	NO_NO.UTF-8	Norwegian	Norway	1208
2937	sv_SE	sv_SE.ISO8859-1	Swedish	Sweden	819

		sv_SE.8859-15	Swedish	Sweden	923
	SV_SE	SV_SE.UTF-8	Swedish	Sweden	1208
2939	de_LU	de_LU.8859-15	German	Luxembourg	923
		de_LU.8859-15@euro	German	Luxembourg	923
	DE_LU	DE_LU.UTF-8	German	Luxembourg	1208
		DE_LU.UTF-8@euro	German	Luxembourg	1208
	de_CH	de_CH.ISO8859-1	German	Switzerland	819
		de_CH.8859-15	German	Switzerland	923
	DE_CH	DE_CH.UTF-8	German	Switzerland	1208
	2940	fr_CH	fr_CH.ISO8859-1	French	Switzerland
		fr_CH.8859-15	French	Switzerland	923
FR_CH		FR_CH.UTF-8	French	Switzerland	1208
2942	it_CH	it_CH.8859-15	Italian	Switzerland	923
	IT_CH	IT_CH.UTF-8	Italian	Switzerland	1208
2954	ar_AA	ar_AA.ISO8859-6	Arabic	Arabic Countries	1089
	ar_AE	ar_AE.ISO8859-6	Arabic	United Arab Emirates	1089
	ar_BH	ar_BH.ISO8859-6	Arabic	Bahrain	1089
	ar_EG	ar_EG.ISO8859-6	Arabic	Egypt	1089
	ar_JO	ar_JO.ISO8859-6	Arabic	Jordan	1089
	ar_KW	ar_KW.ISO8859-6	Arabic	Kuwait	1089
	ar_LB	ar_LB.ISO8859-6	Arabic	Lebanon	1089
	ar_OM	ar_OM.ISO8859-6	Arabic	Oman	1089
	ar_QA	ar_QA.ISO8859-6	Arabic	Qatar	1089
	ar_SA	ar_SA.ISO8859-6	Arabic	Saudi Arabia	1089
	ar_SY	ar_SY.ISO8859-6	Arabic	Syrian Arab Republic	1089
	AR_AA	AR_AA.UTF-8	Arabic	Arabic Countries	1208
	AR_AE	AR_AE.UTF-8	Arabic	United Arab Emirates	1208

	AR_BH	AR_BH.UTF-8	Arabic	Bahrain	1208
	AR_EG	AR_EG.UTF-8	Arabic	Egypt	1208
	AR_JO	AR_JO.UTF-8	Arabic	Jordan	1208
	AR_KW	AR_KW.UTF-8	Arabic	Kuwait	1208
	AR_LB	AR_LB.UTF-8	Arabic	Lebanon	1208
	AR_OM	AR_OM.UTF-8	Arabic	Oman	1208
	AR_QA	AR_QA.UTF-8	Arabic	Qatar	1208
	AR_SA	AR_SA.UTF-8	Arabic	Saudi Arabia	1208
	AR_SY	AR_SY.UTF-8	Arabic	Syrian Arab Republic	1208
2956	tr_TR	tr_TR.ISO8859-9	Turkish	Turkey	920
	TR_TR	TR_TR.UTF-8	Turkish	Turkey	1208
2957	el_GR	el_GR.ISO8859-7	Greek	Greece	813
	EL_GR	EL_GR.UTF-8	Greek	Greece	1208
2958	is_IS	is_IS.ISO8859-1	Icelandic	Iceland	819
		is_IS.8859-15	Icelandic	Iceland	923
	IS_IS	IS_IS.UTF-8	Icelandic	Iceland	1208
2961	iw_IL	iw_IL.ISO8859-8	Hebrew	Israel	916
	HE_IL	HE_IL.UTF-8	Hebrew	Israel	1208
2962	ja_JP	ja_JP.IBM-eucJP	Japanese	Japan	33722
	Ja_JP	ja_JP.IBM-943	Japanese	Japan	943
	JA_JP	JA_JP.UTF-8	Japanese	Japan	1208
2963	nl_BE	nl_BE.ISO8859-1	Dutch	Belgium	819
		nl_BE.IBM-1252	Dutch	Belgium	1252
		nl_BE.IBM-1252@euro	Dutch	Belgium	1252
		nl_BE.8859-15	Dutch	Belgium	923
		nl_BE.8859-15@euro	Dutch	Belgium	923
	NL_BE	NL_BE.UTF-8	Dutch	Belgium	1208
		NL_BE.UTF-8@euro	Dutch	Belgium	1208

2966	fr_BE	fr_BE.ISO8859-1	French	Belgium	819
		fr_BE.IBM-1252	French	Belgium	1252
		fr_BE.IBM-1252@euro	French	Belgium	1252
		fr_BE.8859-15	French	Belgium	923
		fr_BE.8859-15@euro	French	Belgium	923
	FR_BE	FR_BE.UTF-8	French	Belgium	1208
		FR_BE.UTF-8@euro	French	Belgium	1208
	fr_LU	fr_LU.8859-15	French	Luxembourg	923
		fr_LU.8859-15@euro	French	Luxembourg	923
	FR_LU	FR_LU.UTF-8	French	Luxembourg	1208
		FR_LU.UTF-8@euro	French	Luxembourg	1208
2972	th_TH	TH_TH.TIS-620	Thai	Thailand	874
	TH_TH	TH_TH.UTF-8	Thai	Thailand	1208
2974	bg_BG	bg_BG.ISO8859-5	Bulgarian	Bulgaria	915
	BG_BG	BG_BG.UTF-8	Bulgarian	Bulgaria	1208
2975	cs_CZ	cs_CZ.ISO8859-2	Czech	Czech Republic	912
	CS_CZ	CS_CZ.UTF-8	Czech	Czech Republic	1208
2976	hu_HU	hu_HU.ISO8859-2	Hungarian	Hungary	912
	HU_HU	HU_HU.UTF-8	Hungarian	Hungary	1208
2978	pl_PL	pl_PL.ISO8859-2	Polish	Poland	912
	PL_PL	PL_PL.UTF-8	Polish	Poland	1208
2979	ru_RU	ru_RU.ISO8859-5	Russian	Russia	915
	RU_RU	RU_RU.UTF-8	Russian	Russia	1208
2980	pt_BR	pt_BR.ISO8859-1	Portuguese	Brazil	819
		pt_BR.8859-15	Portuguese	Brazil	923
	PT_BR	PT_BR.UTF-8	Portuguese	Brazil	1208
2981	fr_CA	fr_CA.ISO8859-1	French	Canada	819
		fr_CA.8859-15	French	Canada	923

	FR_CA	FR_CA.UTF-8	French	Canada	1208
2986	ko_KR	ko_KR.IBM-eucKR	Korean	Korea	970
	KO_KR	KO_KR.UTF-8	Korean	Korea	1208
2987	zh_TW	zh_TW.IBM-eucTW	Traditional Chinese	Taiwan	964
	Zh_TW	ZH_TW.big5	Traditional Chinese	Taiwan	950
	zh_TW	ZH_TW.UTF-8	Traditional Chinese	Taiwan	1208
2989	zh_CN	zh_CN.IBM-eucCN	Simplified Chinese	People's Republic of China	1383
	Zh_CN	zh_CN.GBK	Simplified Chinese	People's Republic of China	1386
	ZH_CN	ZH_CN.UTF-8	Simplified Chinese	People's Republic of China	1208
2992	ro_RO	ro_RO.ISO8859-2	Romanian	Romania	912
	RO_RO	RO_RO.UTF-8	Romanian	Romania	1208
2994	sk_SK	sk_SK.ISO8859-2	Slovak	Slovakia	912
	SK_SK	SK_SK.UTF-8	Slovak	Slovakia	1208
2995	sq_AL	sq_AL.ISO8859-1	Serbian Cyrillic	Yugoslavia	915
		sq_AL.8859-15	Serbian Cyrillic	Yugoslavia	923
	SQ_AL	SQ_AL.UTF-8	Serbian Cyrillic	Yugoslavia	1208



OS/400 PASE Environment Variables

Overview

OS/400 PASE environment variables are independent of ILE environment variables. Setting a variable in one environment has no effect on the other environment, but several system interfaces allow you to copy variables between environments:

- The **Qp2RunPase** API lets you specify any list of environment variables you want to initialize for the OS/400 PASE program. See [Run an OS/400 PASE Program](#) (Qp2RunPase) documentation for more information.
- The **QP2SHELL** and **QP2TERM** APIs initialize the OS/400 PASE environment with a copy of nearly all ILE environment variables. See [Run an OS/400 PASE Shell Program](#) (QP2SHELL) documentation for more information.
- The **systemCL** OS/400 PASE runtime function copies nearly all OS/400 PASE environment variables to the ILE environment for option **SYSTEMCL_ENVIRON**. See [Run a CL Command for OS/400 PASE](#) (systemCL) documentation for more information.
- The OS/400 PASE **system** utility copies nearly all OS/400 PASE environment variables to the ILE environment for option **-e**. See [Run a CL Command](#) (OS/400 PASE **system** utility) documentation for more information.

Special OS/400 PASE Environment Variables

Some OS/400 PASE runtime behaviors are different from AIX because of differences between the two operating systems. You can use these OS/400 PASE environment variables to control some of the differences:

» PASE_EXEC_QOPENSYS

PASE_EXEC_QOPENSYS can be used to prevent the system from searching the /QOpenSys file system for an absolute path (starting with "/") specified as an argument to **exec** or **Qp2RunPase**, or in the first line of a shell script. The system normally searches the /QOpenSys file system if the absolute path name for an OS/400 PASE program or script cannot be opened or is not a regular bytestream file. OS/400 directory /usr/bin contains links to QShell utilities that cannot run as OS/400 PASE programs, so searching /QOpenSys allows more AIX programs and shell scripts to run unchanged (using OS/400 PASE utilities in directory /QOpenSys/usr/bin). The system does not do an extended search in the /QOpenSys file system if the OS/400 PASE shell or other program that calls **exec** or **Qp2RunPase** has changed credentials (**setuid** or **setgid**) or if the OS/400 PASE environment specifies **PASE_EXEC_QOPENSYS=N**.◀

PASE_MAXDATA64

PASE_MAXDATA64 specifies the maximum number of 256MB segments provided for brk (heap) storage in a 64-bit OS/400 PASE program. If **PASE_MAXDATA64** is omitted or contains an invalid value (either non-numeric or less than one), a default of 256 segments (64GB) is used. **PASE_MAXDATA64** has no effect on 32-bit OS/400 PASE programs, and it must be set either in the initial environment passed to **Qp2RunPase** or before running **exec** for a 64-bit OS/400 PASE program.

PASE_MAXSHR64

PASE_MAXSHR64 specifies the maximum number of 256MB segments provided for shared memory (shmat and mmap) in a 64-bit OS/400 PASE program. If **PASE_MAXSHR64** is omitted or contains an invalid value (either non-numeric or less than one), a default of 256 segments (64GB) is used. **PASE_MAXSHR64** has no effect on 32-bit OS/400 PASE programs, and it must be set either in the initial environment passed to **Qp2RunPase** or before running **exec** for a 64-bit OS/400 PASE program.

PASE_STDIO_ISATTY

The default behavior of the OS/400 PASE **isatty** runtime function returns true for file descriptors 0, 1, and 2 (stdin, stdout, and stderr), regardless of whether the open file is a tty device. Setting OS/400 PASE environment variable **PASE_STDIO_ISATTY** to N, either in the initial environment passed to **Qp2RunPase** or before the first invocation of **isatty**, causes **isatty** to return an accurate indication of whether the open file is a tty device.

PASE_SYSCALL_NOSIGILL

The OS/400 PASE kernel exports some system calls that are implemented by the AIX kernel but are unsupported by OS/400 PASE. **»**The default behavior for any unsupported syscall is to send exception message MCH3204, which the system converts to OS/400 PASE signal **SIGILL**. The unsupported syscall returns a function result of -1 with OS/400-unique errno EUNKNOWN (3474) if the signal is ignored or the handler returns. Message MCH3204 appears in the OS/400 job log to provide the name of the unsupported system call and the OS/400 PASE instruction address that caused the error. The message may also include the internal dump identifier for a VLOG entry that contains this information:
««

```
syscall number (GPR2 value)
OS/400 PASE instruction address
Link register value
GPR3-10 values (if available, or zero otherwise)
syscall name (if known, converted to uppercase)
```

OS/400 PASE programs can suppress the exception message and **SIGILL** signal for unsupported system calls by setting environment variable **PASE_SYSCALL_NOSIGILL** either in the initial environment passed to **Qp2RunPase** or before running **exec**. **PASE_SYSCALL_NOSIGILL** is ignored if the OS/400 PASE program has the S_ISUID or S_ISGID attribute, but otherwise is interpreted as a list of syscall function names with optional errno values, delimited by colons. The colon-delimited values must take one of these forms:

```
syscall_name
syscall_name=errno_name      (errno_name is EINVAL, EPERM, and so on)
syscall_name=errno_number    (errno_number is 0-127)
```

SIGILL is suppressed for any *syscall_name* in the list that is recognized as an OS/400 PASE system call. The first or only entry in the list may use a special *syscall_name* of "ALL" to set a default behavior for all unsupported syscalls. Any entry in the list that is not an OS/400 PASE syscall name is ignored, and specifying the name of a syscall that is supported by the OS/400 PASE kernel has no effect on the operation of that syscall.

Any syscall in the **PASE_SYSCALL_NOSIGILL** list that is unsupported by the OS/400 PASE kernel returns a function result of -1 with the specified errno value (defaulting to ENOSYS) except that specifying *errno_number* of 0 causes the unsupported syscall to return a function result of zero (without setting *errno*). An invalid *errno_name* or *errno_number* defaults to ENOSYS.

For example, the following **PASE_SYSCALL_NOSIGILL** value suppresses **SIGILL** for all unsupported syscalls. "quotactl" returns EPERM and "audit" returns function result of zero, while all other unsupported syscalls return ENOSYS:

```
export PASE_SYSCALL_NOSIGILL=ALL:quotactl=EPERM:audit=0
```

Note: **PASE_SYSCALL_NOSIGILL** is not intended for production programs. It is provided as a convenience for feasibility testing using unchanged AIX binaries that need to be modified for production.

»PASE_THREAD_ATTACH

If OS/400 PASE environment variable **PASE_THREAD_ATTACH** is set to Y when an OS/400 PASE program runs `libpthreads.a` initialization (usually at program startup), an ILE thread that was not started by OS/400 PASE will be attached to OS/400 PASE when it calls an OS/400 PASE procedure (using **Qp2CallPase** or **Qp2CallPase2**). Once an ILE thread has attached to OS/400 PASE, that thread is subject to asynchronous interruption for OS/400 PASE functions such as signal handling and thread cancellation. In particular, the thread will be canceled as part of ending the OS/400 PASE program (when **exit** runs or OS/400 PASE processing terminates for a signal).«

PASE_UNLIMITED_PATH_MAX

The OS/400 Integrated File System supports longer path names than the value of **PATH_MAX** (1023) in AIX header file `<limits.h>`. Setting OS/400 PASE environment variable **PASE_UNLIMITED_PATH_MAX** to Y, either in the initial environment passed to **Qp2RunPase** or before running **exec**, allows an OS/400 PASE program to access objects with long path names. OS/400 PASE loader functions and some library runtime functions can fail with path names longer than AIX **PATH_MAX**.

»PASE_USRGRP_LOWERCASE

OS/400 user names and group names are case-insensitive, but the system stores and returns them in uppercase. OS/400 PASE runtime functions that return user names and group names (`getpwnam`, `getpwuid`, `getgrnam`, and `getgrgid`) default to converting them to lowercase unless OS/400 PASE environment variable **PASE_USRGRP_LOWERCASE** is set to N.«

OS/400 PASE Signal Handling

OS/400 PASE Signals and ILE Signals


OS/PASE signals and POSIX/ILE signals are independent, so it is not possible to directly call a handler for one signal type by raising the other type of signal. However, the [Post an OS/400 PASE Signal](#) (Qp2SignalPase) API can be used as the handler for any ILE signal to post a corresponding OS/400 PASE signal. An OS/400 PASE program can also define handlers for OS/400 PASE signals that call ILE procedures to post equivalent ILE signals. Program **QP2SHELL** and the OS/400 PASE **fork** function always setup handlers to map every ILE signal to a corresponding OS/400 PASE signal.

OS/400 Messages and OS/400 PASE Programs

Many OS/400 applications and system functions report errors with exception messages sent to program call message queues. See [Message Handling Terms and Concepts](#) for information about exception messages and program call message queues.

The system only creates program call message queues for ILE procedures and OMI programs. Any machine exception caused by an operation inside an OS/400 PASE program (such as MCH0601 for a storage reference error) is sent to the program call message queue for an (internal) ILE procedure in service program QP2USER. This ILE procedure is also the apparent caller of any ILE procedure the OS/400 PASE program calls directly (using `_ILECALLX` or `_ILECALL`), so any OS/400 message the called procedure sends to its caller goes to the same message queue used for machine exceptions.

OS/400 Exceptions and OS/400 PASE Signals

The ILE procedure in service program QP2USER that runs OS/400 PASE programs handles any exception and converts it to an OS/400 PASE signal, the same way POSIX/ILE C runtime converts exceptions to ILE signals. The specific signal used depends on the OS/400 message identifier for the exception. OS/400 PASE and ILE use different signal numbers, but both map any specific message identifier to the same signal name (such as SIGSEGV). See the [WebSphere Development Studio: ILE C/C++ Programmer's Guide](#)  for details.

➤ An OS/400 PASE signal handler can determine whether a signal is associated with an exception message by inspecting field `msgkey` in the `ucontext_t_os400` structure (declared in header file `as400_types.h`) that is passed as an argument to the handler. ⚡ A non-zero value is the message reference key for the OS/400 message that caused the signal. Zero indicates the signal is not associated with an OS/400 message (which is always true for asynchronous signals). The OS/400 PASE program can use the message reference key to receive the exception message (see [Receive Program Message for OS/400 PASE](#)) for more details about the error.