



@server

iSeries

OS/400 PASE APIs

*Version 5 Release 3*







@server

iSeries

OS/400 PASE APIs

*Version 5 Release 3*

**Note**

Before using this information and the product it supports, be sure to read the information in "Notices," on page 91.

**Sixth Edition (August 2005)**

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>OS/400 PASE APIs . . . . .</b>	<b>1</b>
APIs . . . . .	1
OS/400 PASE Callable Program APIs . . . . .	1
QP2SHELL() and QP2SHELL2()—Run an OS/400 PASE Shell Program . . . . .	2
Parameters . . . . .	2
Authorities . . . . .	2
Return Value . . . . .	3
Error Messages . . . . .	3
Usage Notes . . . . .	3
Related Information . . . . .	4
QP2TERM()—Run an OS/400 PASE Terminal Session	5
Parameters . . . . .	5
Authorities . . . . .	5
Return Value . . . . .	5
Error Messages . . . . .	5
Usage Notes . . . . .	6
Related Information . . . . .	6
OS/400 PASE ILE Procedure APIs . . . . .	6
Qp2dlclose()—Close a Dynamically Loaded OS/400 PASE Module . . . . .	7
Parameters . . . . .	7
Authorities . . . . .	7
Return Value . . . . .	7
Usage Notes . . . . .	8
Related Information . . . . .	8
Qp2CallPase()—Call an OS/400 PASE Procedure . . . . .	8
Parameters . . . . .	8
Authorities . . . . .	10
Return Value . . . . .	10
Usage Notes . . . . .	10
Related Information . . . . .	11
Qp2dlerror()—Retrieve OS/400 PASE Dynamic Load Error Information . . . . .	12
Parameters . . . . .	12
Authorities . . . . .	12
Return Value . . . . .	12
Usage Notes . . . . .	12
Related Information . . . . .	12
Qp2dlopen()—Dynamically Load an OS/400 PASE Module. . . . .	13
Parameters . . . . .	13
Authorities . . . . .	14
Return Value . . . . .	14
Usage Notes . . . . .	14
Related Information . . . . .	14
Qp2dlsym()—Find an Exported OS/400 PASE Symbol. . . . .	14
Parameters . . . . .	15
Authorities . . . . .	15
Return Value . . . . .	15
Usage Notes . . . . .	15
Related Information . . . . .	15
Qp2EndPase()—End an OS/400 PASE Program . . . . .	16
Parameters . . . . .	16
Authorities . . . . .	16

Return Value . . . . .	16
Usage Notes . . . . .	16
Related Information . . . . .	16
Qp2errnop()—Retrieve OS/400 PASE errno Pointer	17
Parameters . . . . .	17
Authorities . . . . .	17
Return Value . . . . .	17
Usage Notes . . . . .	17
Related Information . . . . .	17
Qp2free()—Free OS/400 PASE Heap Memory . . . . .	17
Parameters . . . . .	18
Authorities . . . . .	18
Return Value . . . . .	18
Usage Notes . . . . .	18
Related Information . . . . .	18
Qp2jobCCSID()—Retrieve Job CCSID for OS/400 PASE . . . . .	18
Parameters . . . . .	19
Authorities . . . . .	19
Return Value . . . . .	19
Usage Notes . . . . .	19
Related Information . . . . .	19
Qp2malloc()—Allocate OS/400 PASE Heap Memory	19
Parameters . . . . .	20
Authorities . . . . .	20
Return Value . . . . .	20
Usage Notes . . . . .	20
Related Information . . . . .	20
Qp2paseCCSID()—Retrieve OS/400 PASE CCSID. . . . .	20
Parameters . . . . .	21
Authorities . . . . .	21
Return Value . . . . .	21
Usage Notes . . . . .	21
Related Information . . . . .	21
Qp2ptrsize()—Retrieve OS/400 PASE Pointer Size	21
Parameters . . . . .	21
Authorities . . . . .	22
Return Value . . . . .	22
Usage Notes . . . . .	22
Related Information . . . . .	22
Qp2RunPase()—Run an OS/400 PASE Program . . . . .	22
Parameters . . . . .	22
Authorities . . . . .	23
Return Value . . . . .	23
Error Messages . . . . .	24
Usage Notes . . . . .	24
Related Information . . . . .	26
Qp2SignalPase()—Post an OS/400 PASE Signal . . . . .	26
Parameters . . . . .	27
Authorities . . . . .	27
Return Value . . . . .	27
Usage Notes . . . . .	27
Related Information . . . . .	27
Runtime Functions For Use by OS/400 PASE Programs . . . . .	27

build_ILEarglist()—Build an ILE Argument List for OS/400 PASE. . . . .	30	Usage Notes . . . . .	45
Parameters . . . . .	30	systemCL()—Run a CL Command for OS/400 PASE	46
Authorities . . . . .	31	Parameters . . . . .	46
Return Value . . . . .	31	Authorities . . . . .	47
Usage Notes . . . . .	31	Return Value . . . . .	47
Related Information . . . . .	31	Usage Notes . . . . .	47
fork400()and f_fork400()—Create A New Process with OS/400 PASE Options . . . . .	32	_CVTERRNO()—Convert ILE errno to OS/400 PASE	
Parameters . . . . .	32	errno . . . . .	48
Authorities . . . . .	32	Parameters . . . . .	48
Return Value . . . . .	33	Authorities . . . . .	48
Error Conditions. . . . .	33	Return Value . . . . .	48
Usage Notes . . . . .	33	Usage Notes . . . . .	48
Related Information . . . . .	33	Related Information . . . . .	49
QMHRMVM()—Receive Nonprogram Message for OS/400 PASE. . . . .	34	_CVTSPP()—Convert Space Pointer for OS/400 PASE . . . . .	49
Parameters . . . . .	34	Parameters . . . . .	49
Authorities . . . . .	35	Authorities . . . . .	49
Return Value . . . . .	35	Return Value . . . . .	49
Related Information . . . . .	35	Error Conditions. . . . .	49
QMHRMCPM()—Receive Program Message for OS/400 PASE. . . . .	36	Usage Notes . . . . .	49
Parameters . . . . .	37	Related Information . . . . .	50
Authorities . . . . .	37	_CVTTS64()—Convert Teraspace Address for OS/400 PASE. . . . .	50
Return Value . . . . .	37	Parameters . . . . .	50
Usage Notes . . . . .	37	Authorities . . . . .	50
Related Information . . . . .	37	Return Value . . . . .	50
QMHSNDM()—Send Nonprogram Message for OS/400 PASE. . . . .	38	Error Conditions. . . . .	50
Parameters . . . . .	38	Usage Notes . . . . .	50
Authorities . . . . .	39	Related Information . . . . .	51
Return Value . . . . .	39	_GETTS64() and _GETTS64_SPP()—Get Teraspace Address for OS/400 PASE . . . . .	51
Related Information . . . . .	39	Parameters . . . . .	51
QMHSNDPM()—Send Program Message for OS/400 PASE . . . . .	40	Authorities . . . . .	51
Parameters . . . . .	40	Return Value . . . . .	51
Authorities . . . . .	41	Error Conditions. . . . .	52
Return Value . . . . .	41	Usage Notes . . . . .	52
Usage Notes . . . . .	41	Related Information . . . . .	52
Related Information . . . . .	41	_GETTS64M()—Get Multiple Teraspace Pointers for OS/400 PASE. . . . .	52
Qp2setenv_ile()—Set ILE environment variables for OS/400 PASE. . . . .	42	Parameters . . . . .	52
Parameters . . . . .	42	Authorities . . . . .	53
Authorities . . . . .	42	Return Value . . . . .	53
Return Value . . . . .	42	Error Conditions. . . . .	53
Error Conditions. . . . .	42	Usage Notes . . . . .	53
Usage Notes . . . . .	43	Related Information . . . . .	53
Related Information . . . . .	43	_ILECALLX()—Call an ILE Procedure for OS/400 PASE . . . . .	54
size_ILEarglist()—Compute ILE Argument List Size for OS/400 PASE . . . . .	43	Parameters . . . . .	54
Parameters . . . . .	43	Authorities . . . . .	56
Authorities . . . . .	44	Return Value . . . . .	56
Return Value . . . . .	44	Usage Notes . . . . .	56
Related Information . . . . .	44	Related Information . . . . .	57
SQLOverrideCCSID400()—Override SQL CLI CCSID for OS/400 PASE . . . . .	45	_ILELOADX()—Load an ILE Bound Program for OS/400 PASE. . . . .	58
Parameters . . . . .	45	Parameters . . . . .	58
Authorities . . . . .	45	Authorities . . . . .	59
Return Value . . . . .	45	Return Value . . . . .	59
Error Conditions. . . . .	45	Error Conditions. . . . .	59
		Related Information . . . . .	59
		_ILESVMX()—Find an Exported ILE Symbol for OS/400 PASE. . . . .	60

Parameters . . . . .	60	Authorities . . . . .	70
Authorities . . . . .	60	Return Value . . . . .	70
Return Value . . . . .	60	Error Conditions. . . . .	70
Error Conditions. . . . .	61	Usage Notes . . . . .	70
Related Information . . . . .	61	Related Information . . . . .	71
<b>_MEMCPY_WT()—Copy Memory With Tags for</b>		<b>_SETSPPM()—Set Multiple Space Pointers for</b>	
<b>OS/400 PASE. . . . .</b>	<b>61</b>	<b>OS/400 PASE. . . . .</b>	<b>71</b>
Parameters . . . . .	62	Parameters . . . . .	72
Authorities . . . . .	62	Authorities . . . . .	72
Return Value . . . . .	62	Return Value . . . . .	72
Error Conditions. . . . .	62	Error Conditions. . . . .	72
Usage Notes . . . . .	62	Usage Notes . . . . .	72
Related Information . . . . .	62	Related Information . . . . .	72
<b>_PGMCALL()—Call an OS/400 Program for OS/400</b>		<b>_STRLEN_SPP()—Determine Character String</b>	
<b>PASE . . . . .</b>	<b>63</b>	<b>Length for OS/400 PASE . . . . .</b>	<b>73</b>
Parameters . . . . .	63	Parameters . . . . .	73
Authorities . . . . .	64	Authorities . . . . .	73
Return Value . . . . .	64	Return Value . . . . .	73
Error Conditions. . . . .	64	Error Conditions. . . . .	73
Usage Notes . . . . .	64	Usage Notes . . . . .	73
Related Information . . . . .	65	Related Information . . . . .	73
<b>_RETURN()—Return Without Exiting OS/400 PASE</b>	<b>65</b>	<b>_STRNCPY_SPP()—Copy Character String for</b>	
Parameters . . . . .	65	<b>OS/400 PASE. . . . .</b>	<b>74</b>
Authorities . . . . .	65	Parameters . . . . .	74
Return Value . . . . .	65	Authorities . . . . .	74
Error Conditions. . . . .	65	Error Conditions. . . . .	74
Usage Notes . . . . .	66	Usage Notes . . . . .	74
Related Information . . . . .	66	Related Information . . . . .	74
<b>_RSLOBJ()—Resolve to an OS/400 Object for</b>		Concepts . . . . .	75
<b>OS/400 PASE. . . . .</b>	<b>66</b>	OS/400 PASE Runtime Libraries . . . . .	75
Parameters . . . . .	66	OS/400 PASE Locales . . . . .	76
Authorities . . . . .	67	OS/400 PASE Environment Variables. . . . .	85
Return Value . . . . .	67	Overview . . . . .	85
Error Conditions. . . . .	67	Special OS/400 PASE Environment Variables . . . . .	86
Usage Notes . . . . .	68	OS/400 PASE Signal Handling . . . . .	88
Related Information . . . . .	68	OS/400 PASE Signals and ILE Signals . . . . .	88
<b>_SETCCSID()—Set OS/400 PASE CCSID. . . . .</b>	<b>68</b>	OS/400 Messages and OS/400 PASE Programs . . . . .	88
Parameters . . . . .	68	OS/400 Exceptions and OS/400 PASE Signals . . . . .	88
Authorities . . . . .	69		
Return Value . . . . .	69		
Error Conditions. . . . .	69		
Usage Notes . . . . .	69		
Related Information . . . . .	69		
<b>_SETSP() and _SETSP_TS64()—Set Space Pointer</b>			
<b>for OS/400 PASE . . . . .</b>	<b>70</b>		
Parameters . . . . .	70		

**Appendix. Notices . . . . . 91**

Trademarks . . . . .	92
Terms and conditions for downloading and printing publications . . . . .	93
Code disclaimer information. . . . .	94





---

## OS/400 PASE APIs

Portable Application Solutions Environment (OS/400<sup>(R)</sup> PASE) is an integrated runtime environment for AIX<sup>(R)</sup> applications. OS/400 PASE supports the same binary executable format as AIX for PowerPC<sup>(R)</sup> 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:

- “OS/400 PASE Callable Program APIs”
- “OS/400 PASE ILE Procedure APIs” on page 6
- “Runtime Functions For Use by OS/400 PASE Programs” on page 27

See also:

- OS/400 PASE for information about creating OS/400 PASE programs.
- “OS/400 PASE Runtime Libraries” on page 75 for information about OS/400 PASE interfaces that are also supported on AIX.
- “OS/400 PASE Locales” on page 76 for information about OS/400 PASE locales.
- “OS/400 PASE Environment Variables” on page 85 for information about OS/400 PASE environment variables.
- “OS/400 PASE Signal Handling” on page 88 for information about OS/400 PASE signals and how they relate to OS/400 exception messages.

[Top](#) | [APIs by category](#)

---

## APIs

These are the APIs for this category.

---

### OS/400 PASE Callable Program APIs

The callable program APIs run an OS/400<sup>(R)</sup> PASE program. They are:

- “QP2SHELL() and QP2SHELL2()—Run an OS/400 PASE Shell Program” on page 2 (Run an OS/400 PASE Shell Program) runs an OS/400 PASE program in the job that calls the API.
- “QP2TERM()—Run an OS/400 PASE Terminal Session” on page 5 (Run an OS/400 PASE Terminal Session) 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\_CCSID.

**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

<b>Object Referred to</b>	<b>Authority Required</b>
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

- QP2SHELL and QP2SHELL2 provide callable program interfaces to ILE procedure Qp2RunPase. See "Qp2RunPase()—Run an OS/400 PASE Program" on page 22 for details about running an OS/400 PASE program.
- 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.
- 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.
- 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.
- 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).
- 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.
- 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 QP2SHELL2 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_CCSD</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_CCSD=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" on page 76 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: based on the OS/400 job TIMZON attribute) «
	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 can set ILE environment variable PASE_TZ at the system level to provide a default timezone other than the one determined from the job TIMZON attribute. For example, this CL command sets the default timezone to US Central time: <pre>ADDENVVAR ENVVAR(PASE_TZ) VALUE('CST6CDT') LEVEL(*SYS)</pre>
<i>QIBM_IFS_OPEN_MAX</i>	(Default: » "66000" «) 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 65 534 files and the system requires an open file for each OS/400 PASE executable it loads, so the default of 66 000 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" on page 22
- "Qp2SignalPase()—Post an OS/400 PASE Signal" on page 26
- "QP2TERM()—Run an OS/400 PASE Terminal Session" on page 5

API introduced: V4R5

[Top](#) | ["OS/400 PASE APIs," on page 1](#) | [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
CPFB9C4 E	Error running OS/400 PASE terminal session, reason code &1, errno &2.
CPFB9C9 E	Terminal session already in use.
CPFB9CA 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() and QP2SHELL2()—Run an OS/400 PASE Shell Program” on page 2 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.

<code>COLUMNS</code>	If COLUMNS is not already set, QP2TERM sets it to the number of columns available for program output on the interactive display.
<code>ROWS</code>	If ROWS is not already set, QP2TERM sets it to the number of rows available for program output on the interactive display.
<code>QIBM_USE_DESCRIPTOR_STDIO=I</code>	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.
<code>QIBM_PASE_DESCRIPTOR_STDIO=T</code>	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() and QP2SHELL2()—Run an OS/400 PASE Shell Program” on page 2

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## OS/400 PASE ILE Procedure APIs

The ILE procedure APIs run an OS/400<sup>(R)</sup> 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:

- “Qp2dlclose()—Close a Dynamically Loaded OS/400 PASE Module” on page 7 (Close a Dynamically Loaded OS/400 PASE Module) closes and unloads an OS/400 PASE module previously opened by the Qp2dlopen API (or the OS/400 PASE dlopen function).
- “Qp2CallPase()—Call an OS/400 PASE Procedure” on page 8 (Call an OS/400 PASE Procedure) calls a procedure in an OS/400 PASE program that is already running in the job that calls the API.
- “Qp2dlerror()—Retrieve OS/400 PASE Dynamic Load Error Information” on page 12 (Retrieve OS/400 PASE Dynamic Load Error Information) returns a pointer to a string that provides error information for the most recent dynamic load function (Qp2dlopen, Qp2dlsym, or Qp2dlclose API).
- “Qp2dlopen()—Dynamically Load an OS/400 PASE Module” on page 13 (Dynamically Load an OS/400 PASE Module) dynamically loads an OS/400 PASE module by calling the OS/400 PASE dlopen() function.
- “Qp2dlsym()—Find an Exported OS/400 PASE Symbol” on page 14 (Find an Exported OS/400 PASE Symbol) finds an exported OS/400 PASE symbol by calling the OS/400 PASE dlsym() function.
- “Qp2EndPase()—End an OS/400 PASE Program” on page 16 (End an OS/400 PASE Program) ends any OS/400 PASE program currently running in the job.



- “Qp2errnop()—Retrieve OS/400 PASE errno Pointer” on page 17 (Retrieve OS/400 PASE errno Pointer) returns a pointer to the OS/400 PASE errno variable for the current thread.
- “Qp2free()—Free OS/400 PASE Heap Memory” on page 17 (Free OS/400 PASE Heap Memory) frees an OS/400 PASE heap memory allocation by calling the OS/400 PASE free() function.
- “Qp2jobCCSID()—Retrieve Job CCSID for OS/400 PASE” on page 18 (Retrieve Job CCSID for OS/400 PASE) returns the job default CCSID from the last time the OS/400 PASE CCSID was set.
- “Qp2malloc()—Allocate OS/400 PASE Heap Memory” on page 19 (Allocate OS/400 PASE Heap Memory) allocates memory from the OS/400 PASE heap by calling the OS/400 PASE malloc() function.
- “Qp2paseCCSID()—Retrieve OS/400 PASE CCSID” on page 20 (Retrieve OS/400 PASE CCSID) returns the OS/400 PASE CCSID from the last time the OS/400 PASE CCSID was set.
- “Qp2ptrsize()—Retrieve OS/400 PASE Pointer Size” on page 21 (Retrieve OS/400 PASE Pointer Size) returns the pointer size, in bytes, for the OS/400 Application Solutions Environment (OS/400 PASE) program currently running in the job.
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22 (Run an OS/400 PASE Program) runs an OS/400 PASE program in the job that calls the API.
- “Qp2SignalPase()—Post an OS/400 PASE Signal” on page 26 (Post an OS/400 PASE Signal) posts an OS/400 PASE signal to an OS/400 PASE program that is already running in the job that calls the API.

Top | “OS/400 PASE APIs,” on page 1 | 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” on page 12
- “Qp2errno()—Retrieve OS/400 PASE errno Pointer” on page 17
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | 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:

<code>QP2_ARG_END (0)</code>	The end of the list of argument type values.
<code>QP2_ARG_WORD (-1)</code>	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.
<code>QP2_ARG_DWORD (-2)</code>	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.
<code>QP2_ARG_FLOAT32 (-3)</code>	A 4-byte floating point number.
<code>QP2_ARG_FLOAT64 (-4)</code>	An 8-byte floating point number.
<code>QP2_ARG_PTR32 (-5)</code>	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 <code>QP2_ARG_PTR_TOSTACK (0x0fff0000)</code> . 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. <code>QP2_ARG_PTR32</code> is allowed only when calling a procedure in a 32-bit OS/400 PASE program.
<code>QP2_ARG_PTR64 (-6)</code>	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 <code>QP2_ARG_PTR_TOSTACK (0x00000000fff0000)</code> . 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. <code>QP2_ARG_PTR64</code> 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:

<code>QP2_RESULT_VOID (0)</code>	No function result returned.
<code>QP2_RESULT_WORD (-1)</code>	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.
<code>QP2_RESULT_DWORD (-2)</code>	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.
<code>QP2_RESULT_FLOAT64 (-4)</code>	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.

*QP2CALLPASE\_ENVIRON\_ERROR* (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.

*QP2CALLPASE\_ARG\_ERROR* (4)

One or more values in the signature array are not valid.

*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.

*QP2CALLPASE\_RETURN\_NOEXIT* (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 `_SETSPP` 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” on page 22—Run an OS/400 PASE Program

API introduced: V4R5

Top | “OS/400 PASE APIs,” on page 1 | 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” on page 17
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | 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:

<i>QP2_RTLD_NOW</i> (0x00000002)	Load all dependents of the module being loaded and resolve all symbols. Either QP2_RTLD_NOW or QP2_RTLD_LAZY must be specified.
<i>QP2_RTLD_LAZY</i> (0x00000004)	Allow the system to defer loading dependent modules. Either QP2_RTLD_NOW or QP2_RTLD_LAZY must be specified.
<i>QP2_RTLD_GLOBAL</i> (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.
<i>QP2_RTLD_LOCAL</i> (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.
<i>QP2_RTLD_MEMBER</i> (0x00040000)	Specifies that the <i>path</i> argument string may contain the name of a member in an archive (shared library).
<i>QP2_RTLD_NOAUTODEFER</i> (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” on page 12
- “Qp2errnop()—Retrieve OS/400 PASE errno Pointer” on page 17
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | 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” on page 12
- “Qp2errnop()—Retrieve OS/400 PASE errno Pointer” on page 17
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | 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” on page 22
- “\_RETURN()—Return Without Exiting OS/400 PASE” on page 65

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | 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” on page 22

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## 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:

<i>EPERM</i>	An error occurred attempting to call an OS/400 PASE function.
<i>ETERM</i>	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” on page 17
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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” on page 22
- “`Qp2paseCCSID()`—Retrieve OS/400 PASE CCSID” on page 20

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## 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” on page 17
- “`Qp2RunPase()`—Run an OS/400 PASE Program” on page 22

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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” on page 22
- “`Qp2jobCCSID()`—Retrieve Job CCSID for OS/400 PASE” on page 18

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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,” on page 1 | 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)  This argument must be a null pointer.



### symbolData

(Input) This argument is ignored.

### symbolDataLen

(Input) This argument is ignored.

**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:

<i>QP2RUNPASE_ERROR</i> (-1)	An internal error occurred during Qp2RunPase processing.
<i>QP2RUNPASE_RETURN_NOEXIT</i> (-2)	The OS/400 PASE program returned without exiting (by calling the OS/400 PASE <b>_RETURN</b> 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:

<i>WIFEXITED(stat_val)</i>	Evaluates to a nonzero value if OS/400 PASE program ended normally.
<i>WEXITSTATUS(stat_val)</i>	If the value of the <i>WIFEXITED(stat_val)</i> is nonzero, evaluates to the low-order 8 bits of the value the OS/400 PASE program specified as the argument to <i>exit</i> or the function result returned by <i>main</i> .
<i>WIFSIGNALED(stat_val)</i>	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.
<i>WTERMSIG(stat_val)</i>	If the value of <i>WIFSIGNALED(stat_val)</i> 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.
» CPFB9CB E	Qp2RunPase second argument must be a null pointer. «

## 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" on page 85 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:

<i>QIBM_USE_DESCRIPTOR_STDIO</i>	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 <i>stdin</i> , <i>stdout</i> , and <i>stderr</i> . Otherwise, OS/400 PASE file descriptors 0, 1, and 2 are mapped to ILE C runtime files <i>stdin</i> , <i>stdout</i> , and <i>stderr</i> (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 stdout, 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\_FLUSH\_STDIO*

This ILE environment variable controls whether OS/400 PASE runtime flushes every write to a standard output stream attached to a Data Management file (such as a spooled printer file) or to the Dynamic Screen Manager in an interactive job. *QIBM\_PASE\_FLUSH\_STDIO* must be set before starting OS/400 PASE, and only applies when OS/400 PASE is NOT using IFS descriptors for standard I/O (*QIBM\_USE\_DESCRIPTOR\_STDIO* is not set). It is usually only needed for interactive programs that require immediate display of output that does not end with newline. These values are supported:

Y flush both stdout and stderr

1 flush only stdout (OS/400 PASE descriptor 1)

2 flush only stderr (OS/400 PASE descriptor 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 QP0ZSPWT (used by fork) and QP0ZSPWP (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” on page 85 for more information, including OS/400 PASE environment variables you can set to control runtime behaviors that differ from AIX.
6. 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” on page 76 to determine what locales are supported by OS/400 PASE.
7. 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.
8. 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.
9. 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.
10. » Time-of-day information in an OS/400 PASE program depends on the value of OS/400 PASE environment variable *TZ*, which provides information about timezone name and offset from UTC

(Universal Coordinated Time). For example, the correct TZ setting for Central Time in the USA is TZ=CST6CDT. See AIX documentation for more information about environment variable TZ.



11. Any credential 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.
12. 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.
13. 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.
14. Older versions of **Qp2RunPase** used *symbolName*, *symbolData*, and *symbolDataLen* to pass inputs other than character string arguments and environment variables to the OS/400 PASE program. An OS/400 PASE program can retrieve any inputs that cannot be expressed as null-terminated strings (such as tagged pointers) by calling ILE or OPM code (using `_ILECALL` or `_PGMCALL`) with by-address arguments.

## 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() and QP2SHELL2()—Run an OS/400 PASE Shell Program” on page 2
- “QP2TERM()—Run an OS/400 PASE Terminal Session” on page 5

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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:

QP2CALLPASE_NORMAL(0)	An OS/400 PASE signal was posted successfully.
QP2CALLPASE_ENVIRON_ERROR(2)	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.
QP2CALLPASE_ARG_ERROR(4)	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

- “Qp2CallPase()—Call an OS/400 PASE Procedure” on page 8
- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## Runtime Functions For Use by OS/400 PASE Programs

OS/400<sup>(R)</sup> PASE runtime includes interfaces supported on AIX<sup>(R)</sup> and interfaces unique to OS/400 PASE. They are unique to OS/400 PASE

The runtime functions are:

- “build\_ILEarglist()—Build an ILE Argument List for OS/400 PASE” on page 30 (Build an ILE Argument List for OS/400 PASE) builds an ILE argument list using argument values copied from an OS/400 PASE function with the same signature.
- **»** “fork400()and f\_fork400()—Create A New Process with OS/400 PASE Options” on page 32 (Create a New Process with OS/400 PASE Options) creates a new (child) process that is an almost exact copy of the calling (parent) process. **«**
- **»** “fork400()and f\_fork400()—Create A New Process with OS/400 PASE Options” on page 32 (Create a New Process with OS/400 PASE Options) creates a new (child) process that is an almost exact copy of the calling (parent) process. **«**
- “QMHRVCVM()—Receive Nonprogram Message for OS/400 PASE” on page 34 (Receive Nonprogram Message for OS/400 PASE) allows an OS/400 PASE program to receive a message from a nonprogram message queue.
- “QMHRCVPM()—Receive Program Message for OS/400 PASE” on page 36 (Receive Program Message for OS/400 PASE) allows an OS/400 PASE program to receive a message from a program call message queue or from the job external message queue.
- “QMHSNDM()—Send Nonprogram Message for OS/400 PASE” on page 38 (Send Nonprogram Message for OS/400 PASE) allows an OS/400 PASE program to send a message to a nonprogram message queue so it can communicate with another job or user.
- “QMHSNDPM()—Send Program Message for OS/400 PASE” on page 40 (Send Program Message for OS/400 PASE) allows an OS/400 PASE program to send a message to a program call message queue or to the job external message queue.
- “Qp2jobCCSID()—Retrieve Job CCSID for OS/400 PASE” on page 18 (Retrieve Job CCSID for OS/400 PASE) returns the job default CCSID from the last time the OS/400 PASE CCSID was set.
- “Qp2paseCCSID()—Retrieve OS/400 PASE CCSID” on page 20 (Retrieve OS/400 PASE CCSID) returns the OS/400 PASE CCSID from the last time the OS/400 PASE CCSID was set.
- **»** “Qp2setenv\_ile()—Set ILE environment variables for OS/400 PASE” on page 42 (Set ILE Environment Variables for OS/400 PASE) allows an OS/400 PASE program to set ILE environment variables. **«**
- “size\_ILEarglist()—Compute ILE Argument List Size for OS/400 PASE” on page 43 (Compute ILE Argument List Size for OS/400 PASE) computes the number of bytes of memory required to build an ILE argument list.
- “SQLOverrideCCSID400()—Override SQL CLI CCSID for OS/400 PASE” on page 45 (Override SQL CLI CCSID for OS/400 PASE) allows an OS/400 PASE program to specify a CCSID for character arguments and results on SQL runtime functions.
- “systemCL()—Run a CL Command for OS/400 PASE” on page 46 (Run a CL Command for OS/400 PASE) allows an OS/400 PASE program to run a CL command.
- “\_CVTERRNO()—Convert ILE errno to OS/400 PASE errno” on page 48 (Convert ILE errno to OS/400 PASE errno) converts an ILE errno value to a corresponding OS/400 PASE errno value.
- “\_CVTSPP()—Convert Space Pointer for OS/400 PASE” on page 49 (Convert Space Pointer for OS/400 PASE) converts a tagged space pointer value to an equivalent OS/400 PASE memory address.
- **»** “\_CVTTS64()—Convert Teraspace Address for OS/400 PASE” on page 50 (Convert Teraspace Address for OS/400 PASE) converts a 64-bit teraspace address to an equivalent OS/400 PASE memory address. **«**
- **»** “\_GETTS64() and \_GETTS64\_SPP()—Get Teraspace Address for OS/400 PASE” on page 51 (Get Teraspace Address for OS/400 PASE) returns the 64-bit teraspace address equivalent of an OS/400 PASE memory address. **«**
- **»** “\_GETTS64M()—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52 (Get Multiple Teraspace Pointers for OS/400 PASE) retrieves teraspace address equivalents for a set of OS/400 PASE memory addresses. **«**

- [» “\\_GETTS64\(\) and \\_GETTS64\\_SPP\(\)—Get Teraspace Address for OS/400 PASE” on page 51](#) (Get Teraspace Address (from Space Pointer) for OS/400 PASE) returns the 64-bit teraspace address stored in a 16-byte space pointer. [«](#)
- [“\\_ILECALLX\(\)—Call an ILE Procedure for OS/400 PASE” on page 54](#) (Call an ILE Procedure for OS/400 PASE) allows an OS/400 PASE program to call an ILE procedure.
- [“\\_ILELOADX\(\)—Load an ILE Bound Program for OS/400 PASE” on page 58](#) (Load an ILE Bound Program for OS/400 PASE) allows an OS/400 PASE program to load (activate) an ILE bound program.
- [» “\\_ILELOADX\(\)—Load an ILE Bound Program for OS/400 PASE” on page 58](#) (Load an ILE Bound Program for OS/400 PASE) loads (activates) an ILE-bound program. [«](#)
- [“\\_ILESYMX\(\)—Find an Exported ILE Symbol for OS/400 PASE” on page 60](#) (Find Exported ILE Symbol for OS/400 PASE) 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.
- [» “\\_ILESYMX\(\)—Find an Exported ILE Symbol for OS/400 PASE” on page 60](#) (Find an Exported ILE Symbol for OS/400 PASE) finds an exported ILE symbol in the activation of an ILE-bound program. [«](#)
- [“\\_MEMCPY\\_WT\(\)—Copy Memory With Tags for OS/400 PASE” on page 61](#) (Copy Memory With Tags for OS/400 PASE) allows an OS/400 PASE program to copy memory with tagged pointers.
- [“\\_PGM\\_CALL\(\)—Call an OS/400 Program for OS/400 PASE” on page 63](#) (Call an OS/400 Program for OS/400 PASE) calls an OS/400 program (object type \*PGM) from an OS/400 PASE program.
- [“\\_RETURN\(\)—Return Without Exiting OS/400 PASE” on page 65](#) (Return without Exiting OS/400 PASE) returns to the ILE called that called OS/400 PASE in this job, without exiting the OS/400 PASE program.
- [“\\_RSLOBJ\(\)—Resolve to an OS/400 Object for OS/400 PASE” on page 66](#) (Resolve to an OS/400 Object for OS/400 PASE) resolves to an OS/400 object.
- [“\\_SETCCSID\(\)—Set OS/400 PASE CCSID” on page 68](#) (Set OS/400 PASE CCSID) retrieves and sets the OS/400 PASE Coded Character Set Identifier (CCSID) value.
- [» “\\_SETSPP\(\) and \\_SETSPP\\_TS64\(\)—Set Space Pointer for OS/400 PASE” on page 70](#) (Set Space Pointer for OS/400 PASE) sets a tagged space pointer to the teraspace equivalent of an OS/400 PASE memory address. [«](#)
- [“\\_SETSPPM\(\)—Set Multiple Space Pointers for OS/400 PASE” on page 71](#) (Set Multiple Space Pointers for OS/400 PASE) Sets multiple space pointers for OS/400 PASE.
- [» “\\_SETSPP\(\) and \\_SETSPP\\_TS64\(\)—Set Space Pointer for OS/400 PASE” on page 70](#) (Set Space Pointer for OS/400 PASE) sets a space pointer from teraspace address for OS/400 PASE. [«](#)
- [“\\_STRLEN\\_SPP\(\)—Determine Character String Length for OS/400 PASE” on page 73](#) (Determine Character String Length for OS/400 PASE) determines the length of a null-terminated character string.
- [“\\_STRNCPY\\_SPP\(\)—Copy Character String for OS/400 PASE” on page 74](#) (Copy Character String for OS/400 PASE) copies a null-terminated character string.

Top | “OS/400 PASE APIs,” on page 1 | APIs by category



## build\_ILEarglist()—Build an ILE Argument List for OS/400 PASE

### Syntax

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

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

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 **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:

<code>ARG_END(0)</code>	Specifies the end of the signature list.
<code>ARG_INT8 (-1)</code>	Signed 1-byte integer argument.
<code>ARG_UINT8 (-2)</code>	Unsigned 1-byte integer argument.
<code>ARG_INT16 (-3)</code>	Signed 2-byte integer argument.
<code>ARG_UINT16 (-4)</code>	Unsigned 2-byte integer argument.
<code>ARG_INT32 (-5)</code>	Signed 4-byte integer argument.
<code>ARG_UINT32 (-6)</code>	Unsigned 4-byte integer argument.
<code>ARG_INT64 (-7)</code>	Signed 8-byte integer argument.
<code>ARG_UINT64 (-8)</code>	Unsigned 8-byte integer argument.
<code>ARG_FLOAT32 (-9)</code>	4-byte floating-point argument.
<code>ARG_FLOAT64 (-10)</code>	8-byte floating-point argument.
<code>ARG_MEMPTR (-11)</code>	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 “ <code>_ILECALLX()</code> —Call an ILE Procedure for OS/400 PASE” on page 54 ( <code>_ILECALLX</code> ) for more information about how <code>ARG_MEMPTR</code> arguments are handled.

- » **ARG\_MEMTS64** (-14)      The argument is a memory address. The OS/400 PASE procedure argument is an OS/400 PASE memory address that `build_ILEarglist` copies into the `ts64_t` type value in the ILE argument list. See “`_ILECALLX()`—Call an ILE Procedure for OS/400 PASE” on page 54 (`_ILECALLX`) for more information about how `ARG_MEMTS64` arguments are handled. «
- » **ARG\_TS64PTR** (-15)      The argument is a 64-bit teraspace pointer. «  
*Any positive number*      The argument is an aggregate (structure or union). The value in the signature list is the  
 (1-32767)                      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`) 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” on page 54
- “`size_ILEarglist()`—Compute ILE Argument List Size for OS/400 PASE” on page 43

API introduced: V4R5

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## fork400() and f\_fork400()—Create A New Process with OS/400 PASE Options

### Syntax

```
#include <as400_protos.h>

pid_t fork400(const char *jobname,
             unsigned int resourceID);

pid_t f_fork400(const char *jobname,
               unsigned int resourceID);
```

Default Public 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 **fork400()** function creates a new process. The new process (the child process) is an almost exact copy of the calling process (the parent process). **fork400()** is called once (by the parent process), but returns twice (once in the parent process and once in the child process). **fork400()** is the same as the **fork()** function plus it allows additional OS/400 PASE unique options to be specified.

**f\_fork400()** function is a similarly enhanced version of the **f\_fork()** function. When **f\_fork400()** (or **f\_fork()**) is used, one of the **exec** functions must be called in the child process immediately after it is created. **f\_fork400()** does not call the fork handlers so the application data, mutexes and the locks are all undefined in the child process.

## Parameters

### jobname

(Input) Pointer to a null-terminated string in the OS/400 PASE CCSID that specifies the OS/400 job name of the new process.

The job name specified must begin with an alphabetic character [A-Z] or the characters [ \$#@]. The remaining characters must be alphanumeric [A-Z] or [0-9] or [ \$#@\_]. The string should not be longer than 10 characters (not including the terminating null character). If the specified jobname is invalid, the jobname parameter value is ignored.

### resourceID

(Input) A positive integer value specifying the resources affinity identifier for the new process.

Use the value of 0 to let the operating system select the resources affinity identifier value automatically.

## Authorities

**fork400()** and **f\_fork400()** require no authority.



## Return Value

Upon successful completion, the **fork400()** or **f\_fork400()** function returns a value of 0 to the child process and the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and the `errno` global variable is set to indicate the error.

## Error Conditions

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

<code>[EAGAIN]</code>	Exceeds the limit on the total number of processes running or the system does not have the resources necessary to create another process.
<code>[ENOMEM]</code>	Not enough space exists for this process.
<code>[EINVAL]</code>	An invalid argument value was specified.

## Usage Notes

1. Consult the AIX documentation for **fork()** and **f\_fork()** for additional details regarding attributes of the parent process inherited by the child process and differences between **fork()** and **f\_fork()**.
2. The OS/400 PASE environment specification `QIBM_PASE_USE_PRESTART_JOBS=Y` will be ignored when the `fork400()` or `f_fork400()` functions are used with a non-null `jobname` or a non-zero `resourceID` value.

## Related Information

- See the “OS/400 PASE Environment Variables” on page 85 documentation for information about the `PASE_FORK_JOBNAME` environment variable that can be used to specify the OS/400 job name for new processes created using the **fork()** or **f\_fork()** functions.



API introduced: V5R3

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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);
```

Default Public 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 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 ERRC0200 format. The QMHRVCVM 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 QMHRVCVM()—Receive Nonprogram Message for further description of the arguments for the QMHRVCVM and QMHRVCVM1 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” on page 34
- “QMHSNDPM()—Send Program Message for OS/400 PASE” on page 40
- “QMHRCVPM()—Receive Program Message for OS/400 PASE” on page 36

API introduced: V5R1

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

## 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         ccid);
```

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 (QMHRCVPM, QMHRCVPM1, and QMHRCVPM2) 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 (QMHRVCVPM) 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 QMHRVCVPM and QMHRVCVPM1 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 QMHRVCVPM()—Receive Program Message for further description of the arguments for the QMHRVCVPM, QMHRVCVPM1, and QMHRVCVPM2 OS/400 PASE runtime functions.

## Authorities

See QMHRVCVPM()—Receive Program Message for information about authorities required for the QMHRVCVPM, QMHRVCVPM1, and QMHRVCVPM2 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 QMHRVCVPM API, or if the QMHRVCVPM 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 a for storage reference error).

## Related Information

- QMHRVCVPM()—Receive Program Message (system API)
- "QMHRVCVPM()—Receive Program Message for OS/400 PASE" on page 36
- "QMHSNDM()—Send Nonprogram Message for OS/400 PASE" on page 38
- "QMHRVCVM()—Receive Nonprogram Message for OS/400 PASE" on page 34

API introduced: V5R1

Top | "OS/400 PASE APIs," on page 1 | 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 ERRRC0200 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)
- “QMHRCSV()—Receive Nonprogram Message for OS/400 PASE” on page 34
- “QMHSNDPM()—Send Program Message for OS/400 PASE” on page 40
- “QMHRCSVPM()—Receive Program Message for OS/400 PASE” on page 36

API introduced: V5R1

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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 ERRRC0200 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)
- "QMHRCVPM()—Receive Program Message for OS/400 PASE" on page 36
- "QMHSNDM()—Send Nonprogram Message for OS/400 PASE" on page 38
- "QMHRVCM()—Receive Nonprogram Message for OS/400 PASE" on page 34

API introduced: V5R1

[Top](#) | ["OS/400 PASE APIs," on page 1](#) | [APIs by category](#)

---

## Qp2setenv\_ile()—Set ILE environment variables for OS/400 PASE

### Syntax

```
#include <as400_protos.h>

int Qp2setenv_ile(const char *const *env,
                 const char *conflict);
```

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 **Qp2setenv\_ile()** function sets one or more ILE environment variables, with special support to resolve conflicts between ILE and OS/400 PASE variables that have the same name but require different values.

## Parameters

**env** (Input) Address of a list of pointers to null-terminated character strings that specify ILE environment variables to set. Each character string should have the form "NAME=value", where NAME is the environment variable name. The first null pointer indicates the end of the list. ILE environment variables are stored in EBCDIC, so the system converts the character strings from the (ASCII) OS/400 PASE CCSID to the job default CCSID.

### conflict

(Input) Pointer to a character string that specifies a colon-delimited list of environment variable names that have conflicting use between OS/400 PASE and ILE. If *conflict* is a null pointer, the system uses a default string of "SHELL:PATH:LANG:NLSPATH".

## Authorities

None

## Return Value

The function result is zero for normal completion. A 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]	Input arguments were changed during processing in a way that does not allow the function to complete normally.
[ENOMEM]	Insufficient OS/400 PASE heap memory is available to complete the request.

## Usage Notes

1. **Qp2setenv\_ILE** sets an ILE environment variable with the same name as the value specified in the *env* string in most cases, but the system adds a prefix "PASE\_" to the name of the ILE version of some environment variables. The *conflict* argument controls what variables add the name prefix, which lets you pass the current OS/400 PASE environment (runtime variable *environ*) to **Qp2setenv\_ile** without removing or changing variables that have conflicting use between OS/400 PASE and ILE. You can specify the address of a null string for the *conflict* argument to avoid any conflict-resolution processing.
2. Any OS/400 PASE environment variable name with a prefix "ILE\_" is copied to the ILE environment twice. The first copy uses the same variable name, and the second copy uses the name without the prefix. 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. This lets you store ILE environment variable values in the OS/400 PASE environment without conflict.

## Related Information

- “\_PGMCALL()—Call an OS/400 Program for OS/400 PASE” on page 63
- “systemCL()—Run a CL Command for OS/400 PASE” on page 46

◀ API introduced: V5R3

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## size\_ILEarglist()—Compute ILE Argument List Size for OS/400 PASE

Syntax

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

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

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 **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.
» <i>ARG_MEMTS64</i> (-14)	The argument is a field of type ts64_t. «
» <i>ARG_TS64PTR</i> (-15)	The argument is a field of type ts64_t. «
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” on page 54
- “`build_ILEarglist()`—Build an ILE Argument List for OS/400 PASE” on page 30

API introduced: V4R5

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

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

### Syntax

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

```
int SQLOverrideCCSID400(int newCCSID);
```

Default Public Authority: \*USE

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.

---

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

**Syntax**

```
#include <as400_protos.h>

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

Default Public Authority: \*USE

Library: Standard C Library (libc.a)

Threadsafe: Conditional. See "Usage Notes" on page 47.

**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:

<i>SYSTEMCL_MSG_STDOUT</i> (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).
<i>SYSTEMCL_MSG_STDERR</i> (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.

<code>SYSTEMCL_FILTER_STDERR</code> (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.
<code>SYSTEMCL_SPAWN</code> (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 <code>systemCL</code> function.
<code>SYSTEMCL_SPAWN_JOBLOG</code> (0x00000200)	Forces the system to generate an OS/400 job log for the job submitted using option <code>SYSTEMCL_SPAWN</code> .
<code>SYSTEMCL_ENVIRON</code> (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 <code>systemCL</code> function, regardless of whether the command runs in this process or a child process (created for option <code>SYSTEMCL_SPAWN</code> ).

## 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

- `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`.
- 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`.
- 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.
- 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.
- 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).
- 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," on page 1 | APIs by category

---

## **\_CVTERRNO()—Convert ILE errno to OS/400 PASE errno**

Syntax

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

```
int _CVTERRNO(int errno_ile);
```

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 **\_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” on page 22—Run an OS/400 PASE Program

API introduced: V5R1

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **\_CVTSPP()—Convert Space Pointer for OS/400 PASE**

Syntax

```
#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 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” on page 88 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

- “\_CVTTS64()—Convert Teraspace Address for OS/400 PASE”
- “\_GETTS64() and \_GETTS64\_SPP()—Get Teraspace Address for OS/400 PASE” on page 51
- “\_GETTS64M()—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52
- “\_SETSPP() and \_SETSPP\_TS64()—Set Space Pointer for OS/400 PASE” on page 70
- “\_SETSPPM()—Set Multiple Space Pointers for OS/400 PASE” on page 71

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## **\_CVTTS64()—Convert Teraspace Address for OS/400 PASE**

Syntax

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

```
void* _CVTTS64(ts64_t 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 **\_CVTTS64()** function converts a 64-bit teraspace address to an equivalent OS/400 PASE memory address.

## Parameters

**source** (Input) A 64-bit teraspace address.

## Authorities

**\_CVTTS64** requires no authority.

## Return Value

**\_CVTTS64** returns the OS/400 PASE memory address equivalent of the 64-bit teraspace address. The result is zero (null) if the input is either zero or an address that cannot contain OS/400 PASE memory.

## 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” on page 88 for information about handling OS/400 exceptions.

## Usage Notes

1. **\_CVTTS64** returns an OS/400 PASE memory address regardless of whether there is currently any memory at that address.

## Related Information

- “\_CVTSPP()—Convert Space Pointer for OS/400 PASE” on page 49
- “\_GETTS64() and \_GETTS64\_SPP()—Get Teraspace Address for OS/400 PASE”
- “\_GETTS64M()—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52
- “\_SETSPP() and \_SETSPP\_TS64()—Set Space Pointer for OS/400 PASE” on page 70
- “\_SETSPPM()—Set Multiple Space Pointers for OS/400 PASE” on page 71



API introduced: V5R3

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **\_GETTS64() and \_GETTS64\_SPP()—Get Teraspace Address for OS/400 PASE**

### Syntax

```
#include <as400_protos.h>

ts64_t _GETTS64(const void *memory);

ts64_t _GETTS64_SPP(const ILEpointer *source);
```

Default Public 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 `_GETTS64()` function returns the 64-bit teraspace address equivalent of an OS/400 PASE memory address. The `_GETTS64_SPP()` function returns the 64-bit teraspace address stored in a 16-byte space pointer.

## Parameters

### **memory**

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

**source** (Input) Pointer to a 16-byte tagged space pointer or 16-byte null pointer.

## Authorities

`_GETTS64` and `_GETTS64_SPP` require no authority.

## Return Value

`_GETTS64` and `_GETTS64_SPP` return a 64-bit teraspace address or zero.

## 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” on page 88 for information about handling OS/400 exceptions.

## Usage Notes

1. `_GETTS64` returns zero if the input *memory* address is null (zero) or points to a location that cannot contain OS/400 PASE memory.
2. `_GETTS64_SPP` returns zero if *source* is not a tagged space pointer or contains an address that is outside teraspace.
3. `_GETTS64` and `_GETTS64_TS64` return a teraspace address regardless of whether there is currently any memory at the result location.

## Related Information

- “`_CVTSPP()`—Convert Space Pointer for OS/400 PASE” on page 49
- “`_CVTTS64()`—Convert Teraspace Address for OS/400 PASE” on page 50
- “`_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE”
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71

◀ API introduced: V5R3

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## `_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE

Syntax

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

```
void _GETTS64M(ts64_t  *list,  
              unsigned count);
```

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 `_GETTS64M()` function retrieves teraspace address equivalents for a set of OS/400 PASE memory addresses.

## Parameters

**list** (Input/Output) Pointer to an array of type `ts64_t` into which the caller has stored OS/400 PASE memory addresses. `_GETTS64M` replaces each OS/400 PASE memory address with an equivalent 64-bit teraspace address.

**count** (Input) Specifies the number of entries in the *list* array.

## Authorities

`_GETTS64M` requires no authority.

## Return Value

`_GETTS64M` 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” on page 88 for information about handling OS/400 exceptions.

## Usage Notes

1. `_GETTS64M` returns null (zero) for any input address that is either zero or 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. `_GETTS64M` returns (non-null) teraspace pointers regardless of whether there is currently any memory at the OS/400 PASE addresses.

## Related Information

- “`_CVTSPP()`—Convert Space Pointer for OS/400 PASE” on page 49
- “`_CVTTS64()`—Convert Teraspace Address for OS/400 PASE” on page 50
- “`_GETTS64()` and `_GETTS64_SPP()`—Get Teraspace Address for OS/400 PASE” on page 51
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52



API introduced: V5R3

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **\_ILECALLX()—Call an ILE Procedure for OS/400 PASE**

### Syntax

```
#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);
```

Default Public 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.

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:

<b>Argument Length</b>	<b>Alignment</b>
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:

<code>ARG_END (0)</code>	Specifies the end of the signature list.
<code>ARG_INT8 (-1)</code>	Signed 1-byte integer argument.
<code>ARG_UINT8 (-2)</code>	Unsigned 1-byte integer argument.
<code>ARG_INT16 (-3)</code>	Signed 2-byte integer argument.
<code>ARG_UINT16 (-4)</code>	Unsigned 2-byte integer argument.
<code>ARG_INT32 (-5)</code>	Signed 4-byte integer argument.
<code>ARG_UINT32 (-6)</code>	Unsigned 4-byte integer argument.
<code>ARG_INT64 (-7)</code>	Signed 8-byte integer argument.
<code>ARG_UINT64 (-8)</code>	Unsigned 8-byte integer argument.
<code>ARG_FLOAT32 (-9)</code>	4-byte floating-point argument.
<code>ARG_FLOAT64 (-10)</code>	8-byte floating-point argument.
<code>ARG_MEMPTR (-11)</code>	The argument is a field of type <code>ILEpointer</code> into which the caller has stored an OS/400 PASE memory address (in member address). <code>_ILECALLX</code> and <code>_ILECALL</code> 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 <code>ILEpointer</code> argument value in memory so it contains a tagged space pointer, but the memory may not be updated if the target ILE procedure uses <code>ARGOPT</code> linkage.
<code>ARG_SPCPTR (-12)</code>	The argument is a field of type <code>ILEpointer</code> where the OS/400 PASE program has stored a tagged space pointer (or an untagged or null pointer).
<code>ARG_OPENPTR (-13)</code>	The argument is a field of type <code>ILEpointer</code> where the OS/400 PASE program has stored a 16-byte pointer of any type (including possibly an untagged or null pointer).
» <code>ARG_MEMTS64 (-14)</code>	The argument is a field of type <code>ts64_t</code> into which the caller has stored an OS/400 PASE memory address. <code>_ILECALLX</code> and <code>_ILECALL</code> convert the OS/400 PASE memory address to an equivalent 64-bit teraspace pointer, except that null (address zero) is unchanged. The converted result is passed as the argument value to the target ILE procedure. The <code>ts64_t</code> argument value in memory may or may not be updated. «
» <code>ARG_TS64PTR (-15)</code>	The argument is a field of type <code>ts64_t</code> where the OS/400 PASE program has stored a 64-bit teraspace address. «
<i>Any positive number (1-32767)</i>	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:

<code>RESULT_VOID(0)</code>	No function result.
<code>RESULT_INT8 (-1)</code>	Signed 1-byte integer result, returned in field <code>result.s_int8.r_int8</code> in the <code>ILEarglist</code> argument.
<code>RESULT_UINT8 (-2)</code>	Unsigned 1-byte integer result, returned in field <code>result.s_uint8.r_uint8</code> in the <code>ILEarglist</code> argument.
<code>RESULT_INT16 (-3)</code>	Signed 2-byte integer result, returned in field <code>result.s_int16.r_int16</code> in the <code>ILEarglist</code> argument.
<code>RESULT_UINT16 (-4)</code>	Unsigned 2-byte integer result, returned in field <code>result.s_uint16.r_uint16</code> in the <code>ILEarglist</code> argument.
<code>RESULT_INT32 (-5)</code>	Signed 4-byte integer result, returned in field <code>result.s_int32.r_int32</code> in the <code>ILEarglist</code> argument.

<i>RESULT_UINT32</i> (-6)	Unsigned 4-byte integer result, returned in field <code>result.s_uint32.r_uint32</code> in the <code>ILEarglist</code> argument.
<i>RESULT_INT64</i> (-7)	Signed 8-byte integer result, returned in field <code>result.r_int64</code> in the <code>ILEarglist</code> argument.
<i>RESULT_UINT64</i> (-8)	Unsigned 8-byte integer result, returned in field <code>result.r_uint64</code> in the <code>ILEarglist</code> argument.
<i>RESULT_FLOAT64</i> (-10) Any positive number (1-32767)	8-byte floating-point result, returned in field <code>result.r_float64</code> in the <code>ILEarglist</code> argument. The function result is an aggregate (structure or union). <code>result_type</code> 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 <code>result.r_aggregate.addr</code> in type <code>ILEarglist_base</code> to the OS/400 PASE memory address of the result buffer before calling an ILE procedure that returns an aggregate result. <code>_ILECALLX</code> and <code>_ILECALL</code> convert the OS/400 PASE memory address to a teraspace address the same way they convert <code>ARG_MEMPTR</code> 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:

<i>ILECALL_NOINTERRUPT</i> (0x00000004)	Specifies that OS/400 PASE signals will not interrupt the called ILE procedure. Some system functions (such as <code>select</code> ) can be interrupted by signals. Normally either an ILE signal or an OS/400 PASE signal can interrupt such an operation, but <code>ILECALL_NOINTERRUPT</code> 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” on page 88 for information about handling OS/400 exceptions.

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

<i>ILECALL_NOERROR</i> (0)	The target ILE procedure was called and returned normally.
<i>ILECALL_INVALID_ARG</i> (1)	An invalid value was found in the signature list.
<i>ILECALL_INVALID_RESULT</i> (2)	The <code>result_type</code> value is invalid.
<i>ILECALL_INVALID_FLAGS</i> (3)	The flags value is invalid.

## Usage Notes

- `_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 `_PGMCALL` function or use the `systemCL` function to run the `CL CALL` command.
- 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 `_ILELOADX` to load the bound program (creating a new activation in the child process) and then use `_ILESYMx` to obtain ILE procedure pointers into the new activation.
6. See “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70 (`_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” on page 63
- “`_ILESYMx()`—Find an Exported ILE Symbol for OS/400 PASE” on page 60
- “`_ILELOADX()`—Load an ILE Bound Program for OS/400 PASE” on page 58
- “`size_ILEarglist()`—Compute ILE Argument List Size for OS/400 PASE” on page 43
- “`build_ILEarglist()`—Build an ILE Argument List for OS/400 PASE” on page 30
- “`_CVTSPP()`—Convert Space Pointer for OS/400 PASE” on page 49
- “`_CVTTS64()`—Convert Teraspace Address for OS/400 PASE” on page 50
- “`_GETTS64()` and `_GETTS64_SPP()`—Get Teraspace Address for OS/400 PASE” on page 51
- “`_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71
- “`systemCL()`—Run a CL Command for OS/400 PASE” on page 46

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## **\_ILELOADX()—Load an ILE Bound Program for OS/400 PASE**

### Syntax

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

```
unsigned long long _ILELOADX(const void *id,  
                             unsigned int flags);
```

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

Default Public 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 **\_ILELOADX()** and **\_ILELOAD()** functions load 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

`_ILELOADX` and `_ILELOAD` call the ILE `QleActBndPgmLong` API to activate the bound program. See `QleActBndPgmLong()`—Activate Bound Program Long for information about authorities required to use `_ILELOADX` and `_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 `_ILELOADX` or `_ILELOAD` ran), the function result is an activation mark that uniquely identifies the activation within the process. 64-bit ILE activation mark values can only be returned using `_ILELOADX`.

## 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” on page 88 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`):

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

## Related Information

- “`_ILECALLX()`—Call an ILE Procedure for OS/400 PASE” on page 54
- “`_ILESYMXX()`—Find an Exported ILE Symbol for OS/400 PASE” on page 60
- “`_RSLOBJ()`—Resolve to an OS/400 Object for OS/400 PASE” on page 66

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## **\_ILESYMXX()—Find an Exported ILE Symbol for OS/400 PASE**

### Syntax

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

```
int _ILESYMXX(ILEpointer      *export,  
              unsigned long long actmark,  
              const char      *symbol);
```

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

Default Public 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 **\_ILESYMXX()** and **\_ILESYM()** functions find an exported symbol in the activation of an ILE bound program and return 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 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 **\_ILELOADX** and **\_ILELOAD** functions return an activation mark when they load a bound program. 64-bit activation mark values can only be handled by **\_ILESYMXX** and **\_ILELOADX**.

#### **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**

**\_ILESYMXX** and **\_ILESYM** call the ILE **QleGetExpLong** API to find the exported symbol. See **QleGetExpLong()—Get Export Long** for information about authorities required to use **\_ILESYMXX** and **\_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` 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” on page 88 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

- “`_ILELOADX()`—Load an ILE Bound Program for OS/400 PASE” on page 58
- “`_ILECALLX()`—Call an ILE Procedure for OS/400 PASE” on page 54

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## `_MEMCPY_WT()`—Copy Memory With Tags for OS/400 PASE

```
Syntax
#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);
```

Default Public 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 `_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” on page 88 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” on page 49
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71

API introduced: V4R5

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## **\_PGMCALL()—Call an OS/400 Program for OS/400 PASE**

### Syntax

```
#include <as400_protos.h>

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

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 **\_PGMCALL()** 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 (usually) 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 **PGMCALL\_MAXARGS** (255) arguments can be passed to an OS/400 program  
» unless **PGMCALL\_NOMAXARGS** is specified.



**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 <b>PGMCALL_DIRECT_ARGS</b> 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 <b>select</b> ) can be interrupted by signals. Normally either an ILE signal or an OS/400 PASE signal can interrupt such an operation, but <b>PGMCALL_NOINTERRUPT</b> delays OS/400 PASE signal processing until control returns from the called OS/400 program. This option has no effect on ILE signal handling.

- » Specifies that more than 255 arguments may be passed to the OS/400 program. Current system architecture limits the number of arguments to 16383 when this flag is set. Specifying **PGMCALL\_NOMAXARGS** slightly increases system resource requirements even when no more than 255 arguments are provided. «

*PGMCALL\_NOMAXARGS*  
(0x00000008)
- » Specifies that all arguments are null-terminated ASCII character strings. The system converts argument strings from the OS/400 PASE CCSID to the job default CCSID and passes the converted (EBCDIC) strings to the OS/400 program. The converted strings are stored in an internal system buffer, so any changes made by the OS/400 program are **not** returned to the caller (by-value argument behavior). «

*PGMCALL\_ASCII\_STRINGS*  
(0x00000010)

## 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” on page 88 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 too many 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 usually 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

» unless **PGMCALL\_ASCII\_STRINGS** is specified.

«



5. `_PGMCALL` does no character encoding conversions unless `PGMCALL_ASCII_STRINGS` is specified, 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” on page 66
- “`_ILECALLX()`—Call an ILE Procedure for OS/400 PASE” on page 54
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71
- “`Qp2setenv_ile()`—Set ILE environment variables for OS/400 PASE” on page 42

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## `_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” on page 16 for more information.

## Related Information

- “`Qp2RunPase()`—Run an OS/400 PASE Program” on page 22—Run an OS/400 PASE Program
- “`Qp2EndPase()`—End an OS/400 PASE Program” on page 16—End an OS/400 PASE Program

API introduced: V5R2

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **`_RSLOBJ()`—Resolve to an OS/400 Object for OS/400 PASE**

Syntax

```
#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);
```

Default Public 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 `_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:

<i>RSLOBJ_TS_PGM</i> (0x0201)	Specifies the MI type and subtype for an OS/400 program (object type *PGM).
<i>RSLOBJ_TS_SRVPGM</i> (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” on page 88 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.

[EINTER]	An signal interrupted the operation.
[ENAMETOOLONG]	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.
[ENOENT]	The specified OS/400 object was not found.
[ENOTDIR]	A qualifier part of the <i>path</i> is not a directory.
[ELOOP]	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

- “`_ILELOADX()`—Load an ILE Bound Program for OS/400 PASE” on page 58
- “`_PGMCALL()`—Call an OS/400 Program for OS/400 PASE” on page 63

API introduced: V5R2

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [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” on page 76 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 “Qp2RunPase()—Run an OS/400 PASE Program” on page 22 (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 `ccsidtocs` convert between AIX Character Set names and CCSID values.

## Related Information

- “Qp2RunPase()—Run an OS/400 PASE Program” on page 22—Run an OS/400 PASE Program

API introduced: V4R5

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **\_SETSPP() and \_SETSPP\_TS64()—Set Space Pointer for OS/400 PASE**

### Syntax

```
#include <as400_protos.h>

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

void _SETSPP_TS64(ILEpointer *target,
                 ts64_t      ts64);
```

Default Public 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 `_SETSPP()` function sets a tagged space pointer to the teraspace equivalent of an OS/400 PASE memory address. [»](#) The `_SETSPP_TS64()` function sets a tagged space pointer to the memory identified by a 64-bit teraspace 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).

[»](#) **ts64** (Input) 64-bit teraspace address, or null (zero). [«](#)

## **Authorities**

`_SETSPP` and `_SETSPP_TS64` require no authority.

## **Return Value**

`_SETSPP` and `_SETSPP_TS64` return 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” on page 88 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_TS64` returns a 16-byte null pointer if the input *ts64* value is zero or outside the range of teraspace.
 

«
3. **»** `_SETSPP` and `_SETSPP_TS64` return *target* a space pointer regardless of whether there is currently any memory at the target address.
 

«
4. 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.
5. 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” on page 49
- “`_CVTTS64()`—Convert Teraspace Address for OS/400 PASE” on page 50
- “`_GETTS64()` and `_GETTS64_SPP()`—Get Teraspace Address for OS/400 PASE” on page 51
- “`_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE”

API introduced: V4R5 for `_SETSPP`, V5R3 for `_SETSPP_TS64`

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)

---

## **`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE**

Syntax

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

```
void _SETSPPM(ILEpointer *const *target);
```

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 `_SETSPPM()` function sets multiple tagged space pointers to the teraspace equivalents of OS/400 PASE memory addresses.

## Parameters

**target** (Input/Output) Pointer to a list of pointers (of type ILEpointer), with a null pointer marking the end of the list. `_SETSPPM` updates each ILEpointer with a tagged space pointer to the teraspace equivalent address of the OS/400 PASE memory address input through the *addr* field of the ILEpointer.

## Authorities

`_SETSPPM` requires no authority.

## Return Value

`_SETSPPM` 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” on page 88 for information about handling OS/400 exceptions.

## Usage Notes

1. `_SETSPPM` returns a 16-byte null ILEpointer if the OS/400 PASE memory address is null (zero) or 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. `_SETSPPM` returns space pointers regardless of whether there is currently any memory at the OS/400 PASE addresses.
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” on page 49
- “`_CVTTS64()`—Convert Teraspace Address for OS/400 PASE” on page 50
- “`_GETTS64()` and `_GETTS64_SPP()`—Get Teraspace Address for OS/400 PASE” on page 51
- “`_GETTS64M()`—Get Multiple Teraspace Pointers for OS/400 PASE” on page 52
- “`_SETSPPP()` and `_SETSPPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70



API introduced: V5R3

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)



---

## **\_STRLEN\_SPP()—Determine Character String Length for OS/400 PASE**

### Syntax

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

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

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 `_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” on page 88 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” on page 49
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71

API introduced: V4R5

Top | “OS/400 PASE APIs,” on page 1 | 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);
```

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 `_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” on page 88 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” on page 49
- “`_SETSPP()` and `_SETSPP_TS64()`—Set Space Pointer for OS/400 PASE” on page 70
- “`_SETSPPM()`—Set Multiple Space Pointers for OS/400 PASE” on page 71

---

## Concepts

These are the concepts for this 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," on page 1 for information about interfaces that are unique to OS/400 PASE:

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

Library	Description
libm.a	IEEE Math library
libMrm.a	Motif Runtime library for UIL
↳ libnsl.a	Transport Independent Remote Procedure Call (TI-RPC) ⏪
libpthdebug.a	Threads debug support
libpthread.a	Threads runtime
libpthread_compat.a	Old threads compatibility
libPW.a	Programmers Workbench library
librtl.a	Runtime linking runtime
libSM.a	X Session Management library
↳ libtli.a	Transport Library Interface ⏪
libUil.a	Motif User Interface Language library
libxcurses.a	Curses library
↳ libxti.a	X/Open Transport Library Interface ⏪
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
libxlf90_r.a	FORTTRAN runtime
libxlfpthrds_compat.a	Old FORTRAN threads compatibility
libxlomp_ser.a	Open mp (multi-processing) support
libxlsmp.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 Intrinsics
liby.a	yacc support

Top | "OS/400 PASE APIs," on page 1 | APIs by category

---

## 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. ↳ Any locale with a name ending in "@preeuro" uses the traditional 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
	» id_ID	id_ID.8859-15	Indonesian	Indonesia	923 «
	» ID_ID	ID_ID.UTF-8	Indonesian	Indonesia	1208 «
	» ms_MY	ms_MY.8859-15	Malay	Malaysia	923 «
	» MS_MY	MS_MY.UTF-8	Malay	Malaysia	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
2909	en_BE	en_BE.8859-15	English	Belgium	923
		» en_BE.8859-15@preuro	English	Belgium	923 «
	EN_BE	EN_BE.UTF-8	English	Belgium	1208
		» EN_BE.UTF-8@preuro	English	Belgium	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_SP	SH_SP.UTF-8	Serbian Latin	Yugoslavia	1208
	sh_YU	sh_YU.ISO8859-2	Serbian Latin	Yugoslavia	912
	SH_YU	SH_YU.UTF-8	Serbian Latin	Yugoslavia	1208
	sr_SP	sr_SP.ISO8859-5	Serbian Cyrillic	Yugoslavia	915
	SR_SP	SR_SP.UTF-8	Serbian Cyrillic	Yugoslavia	1208
	sr_YU	sr_YU.ISO8859-5	Serbian Cyrillic	Yugoslavia	915
	SR_YU	SR_YU.UTF-8	Serbian Cyrillic	Yugoslavia	1208
2922	pt_PT	pt_PT.ISO8859-1	Portuguese	Portugal	819
		pt_PT.IBM-1252	Portuguese	Portugal	1252
		» pt_PT.IBM-1252@preuro	Portuguese	Portugal	1252 «
		pt_PT.8859-15	Portuguese	Portugal	923
		» pt_PT.8859-15@preuro	Portuguese	Portugal	923 «
	PT_PT	PT_PT.UTF-8	Portuguese	Portugal	1208
		» PT_PT.UTF-8@preuro	Portuguese	Portugal	1208 «

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2923	nl_NL	nl_NL.ISO8859-1	Dutch	Netherlands	819
		nl_NL.IBM-1252	Dutch	Netherlands	1252
		»nl_NL.IBM-1252@preeuro	Dutch	Netherlands	1252 «
		nl_NL.8859-15	Dutch	Netherlands	923
		»nl_NL.8859-15@preeuro	Dutch	Netherlands	923 «
	NL_NL	NL_NL.UTF-8	Dutch	Netherlands	1208
		»NL_NL.UTF-8@preeuro	Dutch	Netherlands	1208 «
2924	en_AU	en_AU.8859-15	English	Australia	923
	EN_AU	EN_AU.UTF-8	English	Australia	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_HK	en_HK.8859-15	English	Hong Kong	923 «
	»EN_HK	EN_HK.UTF-8	English	Hong Kong	1208 «
	en_IE	en_IE.8859-15	English	Ireland	923
		»en_IE.8859-15@preeuro	English	Ireland	923 «
	EN_IE	EN_IE.UTF-8	English	Ireland	1208
		»EN_IE.UTF-8@preeuro	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_PH	en_PH.8859-15	English	Philippines	923 «
	»EN_PH	EN_PH.UTF-8	English	Philippines	1208 «
	»en_SG	en_SG.8859-15	English	Singapore	923 «
	»EN_SG	EN_SG.UTF-8	English	Singapore	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

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2925	fi_FI	fi_FI.ISO8859-1	Finnish	Finland	819
		fi_FI.IBM-1252	Finnish	Finland	1252
		» fi_FI.IBM-1252@preeuro	Finnish	Finland	1252 «
		fi_FI.8859-15	Finnish	Finland	923
		» fi_FI.8859-15@preeuro	Finnish	Finland	923 «
	FI_FI	FI_FI.UTF-8	Finnish	Finland	1208
		» FI_FI.UTF-8@preeuro	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@preeuro	French	France	1252 «
		fr_FR.8859-15	French	France	923
		» fr_FR.8859-15@preeuro	French	France	923 «
	FR_FR	FR_FR.UTF-8	French	France	1208
		» FR_FR.UTF-8@preeuro	French	France	1208 «
2929	de_AT	de_AT.8859-15	German	Austria	923
		» de_AT.8859-15@preeuro	German	Austria	923 «
	DE_AT	DE_AT.UTF-8	German	Austria	1208
		» DE_AT.UTF-8@preeuro	German	Austria	1208 «
	de_DE	de_DE.ISO8859-1	German	Germany	819
		de_DE.IBM-1252	German	Germany	1252
		» de_DE.IBM-1252@preeuro	German	Germany	1252 «
		de_DE.8859-15	German	Germany	923
		» de_DE.8859-15@preeuro	German	Germany	923 «
	DE_DE	DE_DE.UTF-8	German	Germany	1208
	» DE_DE.UTF-8@preeuro	German	Germany	1208 «	

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2931 (part 1)	ca_ES	ca_ES.ISO8859-1	Catalan	Spain	819
		ca_ES.IBM-1252	Catalan	Spain	1252
		» ca_ES.IBM-1252@preeuro	Catalan	Spain	1252 «
		ca_ES.8859-15	Catalan	Spain	923
		» ca_ES.8859-15@preeuro	Catalan	Spain	923 «
	CA_ES	CA_ES.UTF-8	Catalan	Spain	1208
		» CA_ES.UTF-8@preeuro	Catalan	Spain	1208 «
	es_AR	es_AR.8859-15	Spanish	Argentina	923
	ES_AR	ES_AR.UTF-8	Spanish	Argentina	1208
	» es_BO	es_BO.8859-15	Spanish	Bolivia	923 «
	» ES_BO	ES_BO.UTF-8	Spanish	Bolivia	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_CR	es_CR.8859-15	Spanish	Costa Rica	923 «
	» ES_CR	ES_CR.UTF-8	Spanish	Costa Rica	1208 «
	» es_DO	es_DO.8859-15	Spanish	Dominican Republic	923 «
	» ES_DO	ES_DO.UTF-8	Spanish	Dominican Republic	1208 «
	» es_EC	es_EC.8859-15	Spanish	Ecuador	923 «
	» ES_EC	ES_EC.UTF-8	Spanish	Ecuador	1208 «
	es_ES	es_ES.ISO8859-1	Spanish	Spain	819
		es_ES.IBM-1252	Spanish	Spain	1252
		» es_ES.IBM-1252@preeuro	Spanish	Spain	1252 «
		es_ES.8859-15	Spanish	Spain	923
		» es_ES.8859-15@preeuro	Spanish	Spain	923 «



Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2931 (part 2)	ES_ES	ES_ES.UTF-8	Spanish	Spain	1208
		» ES_ES.UTF-8@preeuro	Spanish	Spain	1208 <<
	» es_GT	es_GT.8859-15	Spanish	Guatemala	923 <<
	» ES_GT	ES_GT.UTF-8	Spanish	Guatemala	1208 <<
	» es_HN	es_HN.8859-15	Spanish	Honduras	923 <<
	» ES_HN	ES_HN.UTF-8	Spanish	Honduras	1208 <<
	es_MX	es_MX.8859-15	Spanish	Mexico	923
	ES_MX	ES_MX.UTF-8	Spanish	Mexico	1208
	» es_NI	es_NI.8859-15	Spanish	Nicaragua	923 <<
	» ES_NI	ES_NI.UTF-8	Spanish	Nicaragua	1208 <<
	» es_PA	es_PA.8859-15	Spanish	Panama	923 <<
	» ES_PA	ES_PA.UTF-8	Spanish	Panama	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	Puerto Rico	923
	ES_PR	ES_PR.UTF-8	Spanish	Puerto Rico	1208
	» es_PY	es_PY.8859-15	Spanish	Paraguay	923 <<
	» ES_PY	ES_PY.UTF-8	Spanish	Paraguay	1208 <<
	» es_SV	es_SV.8859-15	Spanish	El Salvador	923 <<
	» ES_SV	ES_SV.UTF-8	Spanish	El Salvador	1208 <<
	» es_US	es_US.8859-15	Spanish	United States	923 <<
	» ES_US	ES_US.UTF-8	Spanish	United States	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@preeuro	Italian	Italy	1252 <<
		it_IT.8859-15	Italian	Italy	923
		» it_IT.8859-15@preeuro	Italian	Italy	923 <<
	IT_IT	IT_IT.UTF-8	Italian	Italy	1208
		» IT_IT.UTF-8@preeuro	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

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2939	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
	de_LU	de_LU.8859-15	German	Luxembourg	923
		» de_LU.8859-15@preeuro	German	Luxembourg	923 «
	DE_LU	DE_LU.UTF-8	German	Luxembourg	1208
		» DE_LU.UTF-8@preeuro	German	Luxembourg	1208 «
2940	fr_CH	fr_CH.ISO8859-1	French	Switzerland	819
		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

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
2954	ar_AA	ar_AA.ISO8859-6	Arabic	Arabic Countries	1089
	AR_AA	AR_AA.UTF-8	Arabic	Arabic Countries	1208
	ar_AE	ar_AE.ISO8859-6	Arabic	United Arab Emirates	1089
	AR_AE	AR_AE.UTF-8	Arabic	United Arab Emirates	1208
	ar_BH	ar_BH.ISO8859-6	Arabic	Bahrain	1089
	AR_BH	AR_BH.UTF-8	Arabic	Bahrain	1208
	» ar_DZ	ar_DZ.ISO8859-6	Arabic	Algeria	1089 <<
	» AR_DZ	AR_DZ.UTF-8	Arabic	Algeria	1208 <<
	ar_EG	ar_EG.ISO8859-6	Arabic	Egypt	1089
	AR_EG	AR_EG.UTF-8	Arabic	Egypt	1208
	ar_JO	ar_JO.ISO8859-6	Arabic	Jordan	1089
	AR_JO	AR_JO.UTF-8	Arabic	Jordan	1208
	ar_KW	ar_KW.ISO8859-6	Arabic	Kuwait	1089
	AR_KW	AR_KW.UTF-8	Arabic	Kuwait	1208
	ar_LB	ar_LB.ISO8859-6	Arabic	Lebanon	1089
	AR_LB	AR_LB.UTF-8	Arabic	Lebanon	1208
	» ar_MA	ar_MA.ISO8859-6	Arabic	Morocco	1089 <<
	» AR_MA	AR_MA.UTF-8	Arabic	Morocco	1208 <<
	ar_OM	ar_OM.ISO8859-6	Arabic	Oman	1089
	AR_OM	AR_OM.UTF-8	Arabic	Oman	1208
	ar_QA	ar_QA.ISO8859-6	Arabic	Qatar	1089
	AR_QA	AR_QA.UTF-8	Arabic	Qatar	1208
	ar_SA	ar_SA.ISO8859-6	Arabic	Saudi Arabia	1089
	AR_SA	AR_SA.UTF-8	Arabic	Saudi Arabia	1208
	ar_SY	ar_SY.ISO8859-6	Arabic	Syrian Arab Republic	1089
	AR_SY	AR_SY.UTF-8	Arabic	Syrian Arab Republic	1208
	ar_TN	ar_TN.ISO8859-6	Arabic	Tunisia	1089
	AR_TN	AR_TN.UTF-8	Arabic	Tunisia	1208
» ar_YE	ar_YE.ISO8859-6	Arabic	Yemen	1089 <<	
» AR_YE	AR_YE.UTF-8	Arabic	Yemen	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

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
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@preeuro	Dutch	Belgium	1252 «
		nl_BE.8859-15	Dutch	Belgium	923
		» nl_BE.8859-15@preeuro	Dutch	Belgium	923 «
	NL_BE	NL_BE.UTF-8	Dutch	Belgium	1208
		» NL_BE.UTF-8@preeuro	Dutch	Belgium	1208 «
2966	fr_BE	fr_BE.ISO8859-1	French	Belgium	819
		fr_BE.IBM-1252	French	Belgium	1252
		» fr_BE.IBM-1252@preeuro	French	Belgium	1252 «
		fr_BE.8859-15	French	Belgium	923
		» fr_BE.8859-15@preeuro	French	Belgium	923 «
	FR_BE	FR_BE.UTF-8	French	Belgium	1208
		» FR_BE.UTF-8@preeuro	French	Belgium	1208 «
	fr_LU	fr_LU.8859-15	French	Luxembourg	923
		» fr_LU.8859-15@preeuro	French	Luxembourg	923 «
	FR_LU	FR_LU.UTF-8	French	Luxembourg	1208
		» FR_LU.UTF-8@preeuro	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

Feature	Locale Names		Language	Region	CCSID
	Short Name	Full Name			
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.GB18030	Simplified Chinese	People's Republic of China	1386 «
	ZH_CN	ZH_CN.UTF-8	Simplified Chinese	People's Republic of China	1208
	» ZH_HK	ZH_HK.UTF-8	Simplified Chinese	Hong Kong	1208 «
	» ZH_SG	ZH_SG.UTF-8	Simplified Chinese	Singapore	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

Top | "OS/400 PASE APIs," on page 1 | APIs by category

## 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 "Qp2RunPase()—Run an OS/400 PASE Program" on page 22 (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 "QP2SHELL() and QP2SHELL2()—Run an OS/400 PASE Shell Program" on page 2 (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 "systemCL()—Run a CL Command for OS/400 PASE" on page 46 (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\_FORK\_JOBNAME

**PASE\_FORK\_JOBNAME** specifies the OS/400 job name for a new process created with the **fork()** or **f\_fork()** function. Only the first 10 characters in the string are used, and lowercase characters are converted to uppercase. The specified value is ignored and a default job name is used if the string does not follow OS/400 simple name rules (first character alphabetic and subsequent characters alphameric or underscore). Prestarted job are never used when a fork job name is specified. See “fork400() and f\_fork400()—Create A New Process with OS/400 PASE Options” on page 32 (fork400 or f\_fork400) for information about specifying the OS/400 job name for specific fork operations.

Some OS/400 PASE shells (including the default Korn shell) do not set environment variables for exported variables in the shell process itself. Setting **PASE\_FORK\_JOBNAME** in such a shell does not control job names for first-order utility processes created by that shell, but can control job names for processes forked by a utility started by the shell.



### 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`.

Top | “OS/400 PASE APIs,” on page 1 | APIs by category

---

## **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 “`Qp2SignalPase()`—Post an OS/400 PASE Signal” on page 26 (`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 “QMHRCVPM()—Receive Program Message for OS/400 PASE” on page 36) for more details about the error.

[Top](#) | [“OS/400 PASE APIs,” on page 1](#) | [APIs by category](#)



---

## Appendix. Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

#### COPYRIGHT LICENSE:

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

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

---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Advanced 36  
Advanced Function Printing  
Advanced Peer-to-Peer Networking  
AFP  
AIX  
AS/400  
COBOL/400  
CUA  
DB2  
DB2 Universal Database  
Distributed Relational Database Architecture  
Domino  
DPI

DRDA  
eServer  
GDDM  
IBM  
Integrated Language Environment  
Intelligent Printer Data Stream  
IPDS  
iSeries  
Lotus Notes  
MVS  
Netfinity  
Net.Data  
NetView  
Notes  
OfficeVision  
Operating System/2  
Operating System/400  
OS/2  
OS/400  
PartnerWorld  
PowerPC  
PrintManager  
Print Services Facility  
RISC System/6000  
RPG/400  
RS/6000  
SAA  
SecureWay  
System/36  
System/370  
System/38  
System/390  
VisualAge  
WebSphere  
xSeries

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

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

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

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

---

## Terms and conditions for downloading and printing publications

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

**Personal Use:** You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM<sup>(R)</sup>.

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

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

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

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

All material copyrighted by IBM Corporation.

By downloading or printing information from this site, you have indicated your agreement with these terms and conditions.

---

## **Code disclaimer information**

This document contains programming examples.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM<sup>(R)</sup>, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.





Printed in USA