

PXROS-HR Kernel v8.2.0

provided by HighTec EDV-Systeme GmbH



Reference Manual

Copyright © 2020 HighTec EDV-Systeme GmbH

This manual contains descriptions for copyrighted products that are not explicitly indicated as such. The absence of the TM symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, HighTec EDV-Systeme GmbH assumes no responsibility for any inaccuracies. HighTec EDV-Systeme GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. HighTec EDV-Systeme GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

Additionally, HighTec EDV-Systeme GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. HighTec EDV-Systeme GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

Copyright © 2020 HighTec EDV-Systeme GmbH, D-66113 Saarbrücken. Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may occur without the express written consent from HighTec EDV-Systeme GmbH.

HighTec EDV-Systeme GmbH
Stammsitz & Entwicklung
Europaallee 19
D-66113 Saarbrücken
Tel.: 0681/92613-16 Fax: -26

www.hightec-rt.com

Contents

1	Abort Mechanism Services	1
1.1	PxGetAbortFrameSize	1
2	Application Information Services	3
2.1	PxGetAppinfo	3
2.2	PxSetAppinfo	3
3	Delay Job Services	5
3.1	PxDelayRelease	5
3.2	PxDelayRequest	5
3.3	PxDelaySched	6
4	Error Handling Services	9
4.1	PxAbort	9
4.2	PxGetError	9
4.3	PxMessage	9
4.4	PxMessageFunDefault	10
4.5	PxPanic	11
4.6	PxSetError	11
4.7	PxSetMessageFun	11
5	Event Handling Services	13
5.1	PxAwaitEvents	13
5.2	PxExpectAbort	14
5.3	PxGetAbortingEvents	15
5.4	PxGetSavedEvents	16
5.5	PxResetEvents	16
5.6	PxTaskSignalEvents	17
6	Interrupt and Trap Services	19
6.1	PxIntInitVectab	19
6.2	PxIntInstallFastContextHandler	19
6.3	PxIntInstallFastHandler	20
6.4	PxIntInstallHandler	20
6.5	PxIntInstallService	21
6.6	PxInterruptRelease	22
6.7	PxInterruptRequest	23
6.8	PxTrapAbort	24
6.9	PxTrapInitVectab	24
6.10	PxTrapInstallHandler	25
7	Mailbox Services	27
7.1	PxMbxCheck	27

7.2	PxMbxInstallHnd	27
7.3	PxMbxRelease	29
7.4	PxMbxRequest	29
8	Memory Management	31
8.1	PxMcGetSize	31
8.2	PxMcGetType	31
8.3	PxMcGetVarsizedOverhead	32
8.4	PxMcInsertBlk	33
8.5	PxMcRelease	34
8.6	PxMcRequest	34
8.7	PxMcResolveDefault	36
8.8	PxMcReturnAllBlks	36
8.9	PxMcReturnBlk	37
8.10	PxMcTakeBlk	37
9	Message-related Services	39
9.1	PxMsgAwaitRel	39
9.2	PxMsgEnvelop	40
9.3	PxMsgForceRelease	41
9.4	PxMsgGetBufferSize	42
9.5	PxMsgGetData	42
9.6	PxMsgGetMetadata	43
9.7	PxMsgGetOwner	44
9.8	PxMsgGetProtection	45
9.9	PxMsgGetSender	45
9.10	PxMsgGetSize	46
9.11	PxMsgInstallRelmbx	47
9.12	PxMsgReceive	47
9.13	PxMsgRelDataAccess	48
9.14	PxMsgRelease	49
9.15	PxMsgReleaseAllMsg	50
9.16	PxMsgRequest	51
9.17	PxMsgSend	53
9.18	PxMsgSetData	54
9.19	PxMsgSetMetadata	55
9.20	PxMsgSetProtection	55
9.21	PxMsgSetSize	56
9.22	PxMsgSetToAwaitRel	57
10	Object-related Services	59
10.1	PxDelayIdError	59
10.2	PxDelayIdGet	59
10.3	PxDelayIdInvalidate	59
10.4	PxDelayIdsValid	60
10.5	PxDelayIdResetError	60
10.6	PxDelayIdSet	60

10.7 PxGetObjsize	61
10.8 PxInterruptIdError	61
10.9 PxInterruptIdGet	62
10.10PxInterruptIdInvalidate	62
10.11PxInterruptIdsValid	62
10.12PxInterruptIdResetError	63
10.13PxInterruptIdSet	63
10.14PxMbxIdError	63
10.15PxMbxIdGet	64
10.16PxMbxIdInvalidate	64
10.17PxMbxIdsValid	64
10.18PxMbxIdResetError	65
10.19PxMbxIdSet	65
10.20PxMcidError	65
10.21PxMcidGet	66
10.22PxMcidInvalidate	66
10.23PxMcidIsValid	67
10.24PxMcidResetError	67
10.25PxMcidSet	67
10.26PxMsgIdError	68
10.27PxMsgIdGet	68
10.28PxMsgIdInvalidate	68
10.29PxMsgIdsValid	69
10.30PxMsgIdResetError	69
10.31PxMsgIdSet	69
10.32PxObjGetName	70
10.33PxObjSetName	71
10.34PxOpoolIdError	71
10.35PxOpoolIdGet	72
10.36PxOpoolIdInvalidate	72
10.37PxOpoolIdsValid	72
10.38PxOpoolIdResetError	73
10.39PxOpoolIdSet	73
10.40PxPeldError	73
10.41PxPeldGet	74
10.42PxPeldInvalidate	74
10.43PxPeldIsValid	74
10.44PxPeldResetError	75
10.45PxPeldSet	75
10.46PxSysObjReleaseAllObjects	76
10.47PxTaskIdError	76
10.48PxTaskIdGet	76
10.49PxTaskIdInvalidate	77
10.50PxTaskIdsValid	77
10.51PxTaskIdResetError	77
10.52PxTaskIdSet	78
10.53PxToldError	78

10.54PxToldGet	78
10.55PxToldInvalidate	79
10.56PxToldIsValid	79
10.57PxToldResetError	80
10.58PxToldSet	80
11 Object Pool	81
11.1 PxOpoolGetCurrentCapacity	81
11.2 PxOpoolGetType	81
11.3 PxOpoolRelease	82
11.4 PxOpoolRequest	82
11.5 PxOpoolResolveDefault	83
12 TriCore™ Implementation To Access Peripheral Registers	85
12.1 PxRegisterRead	85
12.2 PxRegisterSetMask	85
12.3 PxRegisterWrite	86
13 PxInit	89
13.1 PxInit	89
13.2 PxInitializeBeforePxInit	91
13.3 _PxInit_Start_Cores	91
13.4 _PxInitcall	92
14 Special PXROS Services	93
14.1 PxServiceTaskInit	93
15 System Information Functions	95
15.1 PxCsInfoGetDelayInfo	95
15.2 PxCsInfoGetInterruptInfo	96
15.3 PxCsInfoGetMCInfo	97
15.4 PxCsInfoGetMbxInfo	98
15.5 PxCsInfoGetMsgInfo	99
15.6 PxCsInfoGetMsgsInMbx	100
15.7 PxCsInfoGetNumberOfObjects	101
15.8 PxCsInfoGetObjType	101
15.9 PxCsInfoGetOpoolInfo	102
15.10PxCsInfoGetPeInfo	103
15.11PxCsInfoGetTaskInfo	104
15.12PxCsInfoGetTolInfo	106
16 PXROS Internal System Functions	107
16.1 PxGetCoreId	107
16.2 PxGetGlobalServerMbx	107
16.3 PxMbxRegisterMbx	108
16.4 PxMbxRequestMbx	108
16.5 PxVersion	109
16.6 _PxHndcall	109

17 Task Manipulation Services	111
17.1 PxDie	111
17.2 PxDieService	111
17.3 PxGetId	112
17.4 PxGetPrivileges	112
17.5 PxGetTimeslices	113
17.6 PxRemoveAccessRights	113
17.7 PxRestoreAccessRights	113
17.8 PxSetPrivileges	114
17.9 PxSetTimeslices	114
17.10 PxTaskCheck	115
17.11 PxTaskCreate	115
17.12 PxTaskForceTermination	120
17.13 PxTaskGetAccessRights	121
17.14 PxTaskGetMbx	121
17.15 PxTaskGetPrio	122
17.16 PxTaskGetSize	123
17.17 PxTaskResume	123
17.18 PxTaskSetPrio	124
17.19 PxTaskSuspend	125
17.20 PxTerminate	125
18 Task Mode Manipulation Services	127
18.1 PxClearModebits	127
18.2 PxSetModebits	127
18.3 PxTaskGetModebits	128
19 Time Management	131
19.1 PxPeChange	131
19.2 PxPeRelease	131
19.3 PxPeRequest	132
19.4 PxPeStart	133
19.5 PxPeStop	134
19.6 PxTickDefine_Hnd	135
19.7 PxTickGetCount	135
19.8 PxTickGetTicksFromMilliseconds	136
19.9 PxTickGetTimeInMilliseconds	136
19.10 PxTickSetTicksPerSecond	137
19.11 PxToChange	137
19.12 PxToRelease	138
19.13 PxToRequest	139
19.14 PxToStart	140
19.15 PxToStop	141
20 Trace Services	143
20.1 PxSetTraceFunc	143
20.2 PxTraceAssignBuffer	143

20.3 PxTraceCtrl	144
20.4 PxTraceGetBuffer	144
21 Access Rights	147
21.1 Access Rights	147
21.2 Functions needing Access Rights	150

1 Abort Mechanism Services

1.1 PxGetAbortFrameSize

NAME

PxGetAbortFrameSize() - return the size of an abort frame

SYNOPSIS

```
#include <pxdef.h>
```

```
PxSize_t
```

```
PxGetAbortFrameSize(void);
```

RETURN VALUES

- size of an abort frame

SEE ALSO

- PxExpectAbort()
- PxTaskCreate()

DESCRIPTION

PxGetAbortFrameSize returns the size of an abort frame.

2 Application Information Services

2.1 PxGetAppinfo

NAME

PxGetAppinfo() - return the calling task's application information

SYNOPSIS

```
#include <pxdef.h>
```

```
PxArg_t  
PxGetAppinfo(void);
```

RETURN VALUES

- application information

DESCRIPTION

PxGetAppinfo returns the application information info from the calling task's task control block.

2.2 PxSetAppinfo

NAME

PxSetAppinfo() - set the calling task's application information

SYNOPSIS

```
#include <pxdef.h>
```

```
void  
PxSetAppinfo(PxArg_t info);
```

PARAMETERS

`info` the application info

DESCRIPTION

PxSetAppinfo puts the application information info in the calling task's task control block. It overwrites any information previously stored in the control block.

3 Delay Job Services

3.1 PxDelayRelease

NAME

PxDelayRelease() - release a delay object

SYNOPSIS

```
#include <pxdef.h>
```

```
PxDelay_t
```

```
PxDelayRelease(PxDelay_t Delay);
```

PARAMETERS

Delay delay object to be released

RETURN VALUES

- invalid delay handle on success
- Delay on failure

ERROR CODES

PXERR_DELAY_ILLDELAY - Delay is not a valid delay object

SEE ALSO

- PxDelayRequest()
- Error Handling Services, see [chapter 4](#) on page 9
- Time Management, see [chapter 19](#) on page 131

DESCRIPTION

PxDelayRelease releases the delay job handle Delay by converting it into a generic object and releasing this object. If Delay is in use, the corresponding job is cancelled.

3.2 PxDelayRequest

NAME

PxDelayRequest() - request a delay job handle

PxDelayRequest_EvWait() - request a delay job handle while waiting for events

PxDelayRequest_NoWait() - request a delay job handle with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxDelay_t
```

```
PxDelayRequest(PxOpool_t opoolid);
```

```
PxDelay_t
```

```
PxDelayRequest_EvWait (PxOpool_t opoolid,
                      PxEvents_t events);

PxDelay_t
PxDelayRequest_NoWait (PxOpool_t opoolid);
```

PARAMETERS

`opoolid` the object pool, where the delay object is requested from.

Parameters of PxDelayRequest_EvWait()

`events` event mask with events making the call return

RETURN VALUES

- invalid delay handle on failure
- delay on success

Returnvalues of PxDelayRequest_EvWait()

- events, if request aborted by an event

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool

Exceptions of PxDelayRequest_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

PXERR_OBJ_ABORTED - request aborted by an event

Exceptions of PxDelayRequest_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- [PxDelayRelease\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9
- Time Management, see [chapter 19](#) on page 131

DESCRIPTION

PxDelayRequest... functions create a delay job handle by converting a generic object from object pool opool. The handle's identifier is returned

The functions act differently if there is no object available. In such a case PxDelayRequest_NoWait fails, PxDelayRequest waits until a free object is available, and PxDelayRequest_EvWait waits until either there is a free object or an event specified in the set events occurs.

3.3 PxDelaySched

NAME

PxDelaySched() - schedule a delay job (task service)

PxDelaySched_Hnd() - schedule a delay job (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxDelaySched(PxDelay_t delayId,
             PxTicks_t ticks,
             void (*handler) (PxArg_t),
             PxArg_t arg);

PxError_t
PxDelaySched_Hnd(PxDelay_t delayId,
                 PxTicks_t ticks,
                 void (*handler) (PxArg_t),
                 PxArg_t arg);
```

PARAMETERS

<code>delayId</code>	delay job handle
<code>ticks</code>	PXROS ticks, when the job is executed
<code>*handler</code>	handler function to be executed
<code>arg</code>	argument for the handler function

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_DELAY_ILLDELAY - delayId is not a valid delay handler

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the delayId object

PXERR_TASK_ILLCALL - Task service called by handler

SEE ALSO

- [PxTickSetTicksPerSecond\(\)](#)
- Time Management, see [chapter 19](#) on page [131](#)

DESCRIPTION

PxDelaySched... cancels a potential delay job associated with delayId. If ticks != 0, PXROS schedules activation of the handler call handler(arg) after ticks PXROS ticks and associates this delay job with delayId.

4 Error Handling Services

4.1 PxAbort

NAME

PxAbort() - Abort routine

SYNOPSIS

```
#include <pxdef.h>

void
PxAbort (PxError_t);
```

SEE ALSO

- PxMessageFunDefault()

DESCRIPTION

PxAbort tries to do something sensible in case of a fatal error. Its exact behavior depends on the processor. Usually, PxAbort performs a breakpoint or illegal instruction, hoping that it is caught by either a monitor or by the application itself and performs an endless loop.

4.2 PxGetError

NAME

PxGetError() - return remembered error code

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxGetError (void);
```

RETURN VALUES

- the remembered error code.

SEE ALSO

- PxSetError()

DESCRIPTION

PxGetError returns the last error remembered.

4.3 PxMessage

NAME

PxMessage() - message by current message function

SYNOPSIS

```
#include <pxdef.h>
```

```
void
PxMessage(PxMessageClass_t cls,
          PxError_t err,
          PxArg_t arg1,
          PxArg_t arg2);
```

PARAMETERS

<code>cls</code>	error class
<code>err</code>	PXROS error code
<code>arg1</code>	first argument to error message format string
<code>arg2</code>	second argument to error message format string

SEE ALSO

- [PxMessageFunDefault\(\)](#)
- [PxSetMessageFun\(\)](#)

DESCRIPTION

PxMessage reports problem `err` at class `cls` through the current PXROS error reporting function. `cls` may be chosen as `PXWarning`, `PXLogError`, `PXError` or `PXFatal`.

4.4 PxMessageFunDefault

NAME

`PxMessageFunDefault()` - default message handling routine

SYNOPSIS

```
#include <pxdef.h>

void
PxMessageFunDefault(PxMessageClass_t msg_class,
                   PxError_t err,
                   PxArg_t arg1,
                   PxArg_t arg2);
```

PARAMETERS

<code>msg_class</code>	PXROS error class
<code>err</code>	PXROS error number
<code>arg1</code>	optional argument
<code>arg2</code>	optional argument

SEE ALSO

- [PxAbort\(\)](#)
- [PxMessage\(\)](#)
- [PxSetMessageFun\(\)](#)

DESCRIPTION

`PxMessageFunDefault` is the default message handling routine. This service calls `PxAbort` for messages of class `PXFatal` and ignores other classes.

4.5 PxPanic

NAME

PxPanic() - Panic routine

SYNOPSIS

```
#include <pxdef.h>
```

```
void
PxPanic(void);
```

SEE ALSO

- PxSetMessageFun()

DESCRIPTION

PxPanic tries to do something sensible in case of a fatal error. Its exact behavior depends on the processor. Usually, PxPanic performs a breakpoint or illegal instruction, hoping that it is caught by either a monitor or by the application itself.

4.6 PxSetError

NAME

PxSetError() - remember specified error code

SYNOPSIS

```
#include <pxdef.h>
```

```
void
PxSetError(PxError_t error);
```

PARAMETERS

`error` the error code

SEE ALSO

- PxGetError()

DESCRIPTION

PxSetError sets error as the task's last remembered error.

4.7 PxSetMessageFun

NAME

PxSetMessageFun() - set the PXROS error reporting function

SYNOPSIS

```
#include <pxdef.h>
```

```
void
PxSetMessageFun(PxMessageFun_t messagefun);
```

PARAMETERS

`messagefun` the new PXROS error reporting function

ERROR CODES

PXERR_ACCESS_RIGHT - calling task does not have the right to change the error reporting function

SEE ALSO

- [PxMessage\(\)](#)
- [PxMessageFunDefault\(\)](#)
- [PxPanic\(\)](#)

DESCRIPTION

PxSetMessageFun sets the PXROS error reporting function to messagefun

5 Event Handling Services

5.1 PxAwaitEvents

NAME

PxAwaitEvents() - await specified events

SYNOPSIS

```
#include <pxdef.h>
```

```
PxEvents_t
```

```
PxAwaitEvents(PxEvents_t events);
```

PARAMETERS

`events` event mask to wait for

RETURN VALUES

- events that caused the return

SEE ALSO

- [PxClearModebits\(\)](#)
- [PxPeChange\(\)](#)
- [PxPeRelease\(\)](#)
- [PxPeRequest\(\)](#)
- [PxPeStart\(\)](#)
- [PxPeStop\(\)](#)
- [PxSetModebits\(\)](#)
- [PxTaskSignalEvents\(\)](#)
- [PxToChange\(\)](#)
- [PxToRelease\(\)](#)
- [PxToRequest\(\)](#)
- [PxToStart\(\)](#)
- [PxToStop\(\)](#)
- Event Handling Services, see [chapter 5](#) on page 13

DESCRIPTION

PxAwaitEvents waits until one or more of the events specified in `events` are signalled.

It returns the events that caused the return. If an event from `events` was already signalled, the call returns immediately. If the events overlap with a previous mask of a

PxExpectAbort the behavior of the call depends on the right `PXACCESS_OVERRIDE_ABORT_EVENTS`

If the task has the right, all events are handled by PxAwaitEvents. Otherwise the events to PxExpectAbort have a higher priority and are captured by PxExpectAbort. The events are not counted, if an event is sent multiply it is only received once. If events is zero then the task waits forever.

5.2 PxExpectAbort

NAME

PxExpectAbort() - call a function and expect abort during the call

SYNOPSIS

```
#include <pxdef.h>

PxEvents_t
PxExpectAbort (PxEvents_t ev,
               void func,
               parms...);
```

PARAMETERS

ev	events that make the call return
func	function to call
parms...	parameters to function

RETURN VALUES

- events that made the call return.

SEE ALSO

- PxClearModebits()
- PxGetAbortFrameSize()
- PxGetAbortingEvents()
- PxGetSavedEvents()
- PxSetModebits()
- PxTaskCreate()
- PxTaskGetModebits()
- PxTaskSignalEvents()
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxExpectAbort enables the abort mechanism and calls the function func() with the specified argument. This call is aborted and PxExpectAbort returns prematurely provided that:

1. one of the events specified in events is saved for task and there is no intervening active call to PxExpectAbort, PxAwaitEvents or a PxResetEvents service or a call with the __EvWait suffix (nested in func).

and

2. the abort mechanism is enabled.

otherwise, PxExpectAbort returns when the call returns. If the call has been aborted, PxExpectAbort returns the events which led to abortion and clears them from the saved events (as they have been handled). If the call was not aborted, PxExpectAbort returns 0.

If the function call is aborted, PxExpectAbort must restore the context to its state before activation. Thus PxExpectAbort saves the context in an abort frame taken from the tasks abort stack. This abort frame is released when PxExpectAbort returns. As this implies, the task must provide one abort frame for each nested PxExpectAbort call. The size of the abort frames is processor dependent and defined in the constant PXAbortFrameSize.

During their activation, nested PxExpectAbort calls disable the effects of an enclosing PxExpectAbort call. The current effective aborting events can be obtained via a call to PxGetAbortingEvents. On return from the PxExpectAbort also the task mode is restored and there with the state of the abort mechanism (enabled/disabled).

func is a procedure or PXROS call, with arbitrary arguments; however, any potential return value is lost.

The abort mechanism may be disabled with PxSetModebits and enabled with PxClearModebits or with a new PxExpectAbort call.

5.3 PxGetAbortingEvents

NAME

PxGetAbortingEvents() - return the event mask of the currently active events of the calling task

SYNOPSIS

```
#include <pxdef.h>

PxEvents_t
PxGetAbortingEvents(void);
```

RETURN VALUES

- event mask of calling task

SEE ALSO

- [PxClearModebits\(\)](#)
- [PxExpectAbort\(\)](#)
- [PxSetModebits\(\)](#)
- Task Mode Manipulation Services, see [chapter 18](#) on page [127](#)

DESCRIPTION

PxGetAbortingEvents returns the events able to abort the current activity (when the abort mechanism is enabled).

5.4 PxGetSavedEvents

NAME

PxGetSavedEvents() - return events saved for the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxEvents_t  
PxGetSavedEvents(void);
```

RETURN VALUES

- saved events of calling task

SEE ALSO

- PxClearModebits()
- PxExpectAbort()
- PxSetModebits()
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxGetSavedEvents returns the events saved for the calling task.

5.5 PxResetEvents

NAME

PxResetEvents() - reset events

SYNOPSIS

```
#include <pxdef.h>
```

```
PxEvents_t  
PxResetEvents(PxEvents_t events);
```

PARAMETERS

`events` events to be reset

RETURN VALUES

- the events that are reset by the call

SEE ALSO

- PxClearModebits()
- PxPeChange()
- PxPeRelease()
- PxPeRequest()
- PxPeStart()
- PxPeStop()
- PxSetModebits()

- [PxToChange\(\)](#)
- [PxToRelease\(\)](#)
- [PxToRequest\(\)](#)
- [PxToStart\(\)](#)
- [PxToStop\(\)](#)
- Task Mode Manipulation Services, see [chapter 18](#) on page [127](#)

DESCRIPTION

PxResetEvents resets the events specified in events. The events actually reset (i.e. those that were set before the call) are returned.

5.6 PxTaskSignalEvents

NAME

PxTaskSignalEvents() - signal events to a task (task service)

PxTaskSignalEvents_Hnd() - signal events to a task (handler service)

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
PxTaskSignalEvents(PxTask_t taskid,
                  PxEvents_t events);

PxError_t
PxTaskSignalEvents_Hnd(PxTask_t taskid,
                      PxEvents_t events);
```

PARAMETERS

taskid task to send the events to

events events to send

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - task has not the right to allocate from PXOpoolGlobal-Systemdefault

PXERR_INTERNAL_INCONSISTENCY - allocated object not convertible

PXERR_OBJ_NOOBJ - no free objects available to send the request

PXERR_TASK_ILLTASK - taskid is not a valid task object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxClearModebits\(\)](#)

- [PxExpectAbort\(\)](#)
- [PxSetModebits\(\)](#)
- Event Handling Services, see [chapter 5](#) on page 13

DESCRIPTION

PxTaskSignalEvents signals the events specified in events to taskid. There they are saved until handled by task.

If taskid waits for any of the events (with a PxAwaitEvents call or a call with the _EvWait suffix), the task is readied and the service returns. If some of the task's aborting events occur (see PxExpectAbort) and task's abort mechanism is enabled, the PxExpectAbort call returns prematurely. If the events are signaled to a task on an other core, PXROS will allocate an object for intercore communication to send this request to the other core for execution.

PxTaskSignalEvents_Hnd is the equivalent handler service. The handler service can not send events to tasks on other cores, because it can not allocate an objects for the requested intercore communication.

6 Interrupt and Trap Services

6.1 PxIntInitVectab

NAME

PxIntInitVectab() - initialize the C interrupt interface

SYNOPSIS

```
#include <pxdef.h>

void
PxIntInitVectab (void);
```

SEE ALSO

- PxTrapInitVectab()

DESCRIPTION

PxIntInitVectab initializes the C interrupt interface.

6.2 PxIntInstallFastContextHandler

NAME

PxIntInstallFastContextHandler() - install a fast interrupt handler handled in the context of the task

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxIntInstallFastContextHandler(PxUInt_t intno,
                               PxIntHandler_t inthandler,
                               PxArg_t arg);
```

PARAMETERS

intno	the number of the interrupt for which the fast context handler is installed
inthandler	the interrupt handler to be installed
arg	argument for interrupt handler

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install handlers

PXERR_INTR_ILL - handler already installed from another task

PXERR_REQUEST_INVALID_PARAMETER - intno out of specification

SEE ALSO

- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

Installation of a C-function as fast context interrupt handler. This handler will be called in user mode with task protection of the installing task.

6.3 PxIntInstallFastHandler

NAME

PxIntInstallFastHandler () - install a fast interrupt handler

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxIntInstallFastHandler(PxUInt_t intno,
                        void (* inthandler) (PxArg_t),
                        PxArg_t arg);
```

PARAMETERS

<code>intno</code>	the number of the interrupt for which the fast handler is installed
<code>inthandler</code>	the fast interrupt handler to be installed
<code>arg</code>	argument for fast interrupt handler

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install fast handlers

PXERR_INTR_ILL - handler already installed from another task

PXERR_REQUEST_INVALID_PARAMETER - intno out of specification

SEE ALSO

- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

Installation of a C-function as fast interrupt handler. This handler will be called in supervisor mode with supervisor protection.

6.4 PxIntInstallHandler

NAME

PxIntInstallHandler () - install an interrupt handler

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxIntInstallHandler(PxUInt_t intno,
                    PxInterrupt_t intObj,
```

```
void (* inthandler) (PxArg_t),
PxArg_t arg);
```

PARAMETERS

<code>intno</code>	the number of the interrupt for which the handler is installed
<code>intObj</code>	the interrupt object
<code>inthandler</code>	the interrupt handler to be installed
<code>arg</code>	argument for interrupt handler

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install handlers

PXERR_INTERRUPT_ILLINTERRUPT - `intObj` not valid interrupt object

PXERR_INTR_ILL - handler already installed from another task

PXERR_REQUEST_INVALID_PARAMETER - `intno` out of specification


SEE ALSO


- [PxTaskCreate\(\)](#)
- [PxTrapInstallHandler\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

Installs the C-function handler as interrupt handler for interrupt number `intno`. Whenever this interrupt occurs, handler is queued into the PXROS sysjob list via the interrupt object `intObj`. When the interrupt level is left to system level, handler is executed with argument `arg` on the installing task's interrupt stack.

 handler MUST FOLLOW the standard GNU C calling conventions!

 The function is not protected against the abort mechanism!

 The installation is NOT ATOMIC! The application MUST ENSURE, that no `intno` interrupts occur during a call to `PxIntInstallHandler (intno ,....)` !

6.5 PxIntInstallService

NAME

`PxIntInstallService ()` - install a PXROS service call as an interrupt handler

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxIntInstallService(PxUInt_t intno,
```

```
PxUInt_t service,
PxArg_t arg,
PxEvents_t events);
```

PARAMETERS

<code>intno</code>	the number of the interrupt for which the PXROS service is installed
<code>service</code>	the PXROS service
<code>arg</code>	argument for PXROS service
<code>events</code>	events sent to installing task

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install services

PXERR_ILLEGAL_SERVICE_CALLED - service invalid

PXERR_INTR_ILL - handler already installed from another task

PXERR_REQUEST_INVALID_PARAMETER - intno out of specification

SEE ALSO

- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

Installs a PXROS handler service directly as an interrupt handler.

6.6 PxInterruptRelease

NAME

PxInterruptRelease() - release an interrupt object

SYNOPSIS

```
#include <pxdef.h>
```

```
PxInterrupt_t
PxInterruptRelease(PxInterrupt_t Interrupt);
```

PARAMETERS

<code>Interrupt</code>	interrupt object to be released
------------------------	---------------------------------

RETURN VALUES

- invalid interrupt object on success
- interrupt object on failure

ERROR CODES

PXERR_INTERRUPT_ILLINTERRUPT - Interrupt is not a valid interrupt object

SEE ALSO

- PxInterruptRequest()

- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxBInterruptRelease releases the interrupt object Interrupt by converting it into a generic object and releasing this object.

6.7 PxBInterruptRequest

NAME

PxBInterruptRequest() - request an interrupt object

PxBInterruptRequest_EvWait() - request an interrupt object while waiting for events

PxBInterruptRequest_NoWait() - request an interrupt object with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxBInterrupt_t
PxBInterruptRequest(PxBOpool_t opoolid);

PxBInterrupt_t
PxBInterruptRequest_EvWait(PxBOpool_t opoolid,
                           PxBEvents_t events);

PxBInterrupt_t
PxBInterruptRequest_NoWait(PxBOpool_t opoolid);
```

PARAMETERS

opoolid the object pool, where the interrupt object is requested from.

Parameters of PxBInterruptRequest_EvWait()

events event mask with events making the call return

RETURN VALUES

- invalid interrupt handle on failure
- interrupt object on success

Returnvalues of PxBInterruptRequest_EvWait()

- invalid interrupt object on failure
- events, if request aborted by an event

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OBJ_ABORTED - request aborted by an event

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool

Exceptions of PxBInterruptRequest_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

Exceptions of PxBInterruptRequest_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- [PxInterruptRelease\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxInterruptRequest ... functions create a interrupt job handle by converting a generic object from object pool opool. The handles identifier is returned

The functions act differently if there is no object available. In such a case PxInterruptRequest_NoWait fails, PxInterruptRequest waits until a free object is available, and PxInterruptRequest_EvWait waits until either there is a free object or an event specified in the set events occurs.

6.8 PxTrapAbort

NAME

PxTrapAbort() - default trap handler

SYNOPSIS

```
#include <pxdef.h>

void
PxTrapAbort(int trapno,
            int tin,
            TC_CSA_t *csa);
```

PARAMETERS

trapno	trap number
tin	trap identification number
*csa	pointer to context save area

DESCRIPTION

This function will be called as the default trap handling routine. Since it is called for all traps this function should avoid to use additional CSAs; therefore it is called through a jump (`__attribute__((interrupt_handler))`) and should return with the standard return sequence of interrupt handler.

6.9 PxTrapInitVectab

NAME

PxTrapInitVectab() - initialize trap vector table

SYNOPSIS

```
#include <pxdef.h>

void
PxTrapInitVectab(void);
```

SEE ALSO

- [PxIntInitVectab\(\)](#)

DESCRIPTION

This function initializes the PXROS trap interface.

6.10 PxTrapInstallHandler

NAME

PxTrapInstallHandler() - install a traphandler for a trap

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxTrapInstallHandler(PxUInt_t trapno,
                    PxBool_t (* traphandler)(PxTrapTin_t, PxUInt_t, PxUInt_t, PxUInt_t,
                    PxUInt_t arg);
```

PARAMETERS

<code>trapno</code>	trap number
<code>traphandler</code>	the traphandler
<code>arg</code>	traphandler's argument

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install handlers

PXERR_REQUEST_INVALID_PARAMETER - invalid trap number

SEE ALSO


- [PxIntInstallHandler\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION


Installs the C-function handler as trap handler for trap class trapno. Whenever this trap occurs, handler is called with 6 arguments.

- the trap number and TIN (Trap Indication Number)
- the user defined argument
- the identifier of the task, that caused the trap
- the contents of DSTR (Data Synchronous Error Trap Register)
- the contents of DEADD (Data Error Address Register)
- a pointer to the saved CSA

The trap handler should return 1 if the trap could be handled, 0 otherwise.

 handler MUST FOLLOW the standard GNU C calling conventions!

 The function is not protected against the abort mechanism!

 The installation is NOT ATOMIC! The application MUST ENSURE, that no traps of class trapno occur during a call to PxDTrapInstallHandler (trapno ,....) !

7 Mailbox Services

7.1 PxMbxCheck

NAME

PxMbxCheck() - check the validity of a mailbox

SYNOPSIS

```
#include <pxdef.h>
```

```
PxBool_t  
PxMbxCheck(PxMbx_t mbxid);
```

PARAMETERS

`mbxid` the mailbox object

RETURN VALUES

- true if `mbxid` is a mailbox object
- false if `mbxid` is not a mailbox object

DESCRIPTION

PxMbxCheck checks the validity of a mailbox object. The function returns true if the parameter is a valid mailbox object, else false.

7.2 PxMbxInstallHnd

NAME

PxMbxInstallHnd() - install or remove a mailbox handler.

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxMbxInstallHnd(PxMbx_t mbx,  
                PxMsg_t (*hnd)(PxMsg_t , PxMsgType_t, PxArg_t),  
                PxMsgType_t mode,  
                PxArg_t arg);
```

PARAMETERS

`mbx` mailbox where to install the handler

`hnd` pointer to the handler

`mode` activating messages

`arg` argument (private data) for the handler

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to install handlers

PXERR_MBX_HNDINSTALLED - there is already a handler installed

PXERR_MBX_ILLMBX - mbx is not a valid mailbox

PXERR_MBX_ILLMODE - mode is not a known mode for a mailbox handler

SEE ALSO

- [PxMsgReceive\(\)](#)
- [PxMsgSend\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMbxInstallHnd installs or removes a mailbox handler. If hnd is zero, a possibly existing handler at the mailbox specified in mbx is removed. If hnd is nonzero, PxMbxInstallHnd installs the handler hnd at the mailbox mbx with private data arg. msgtype specifies the type of messages activating the handler. msgtype may be one of the following values:


- PXMsgNormalMsg for normal messages
- PXMsgPrioMsg for prioritized messages, or
- PXMsgAnyMsg for both normal and prioritized messages.

When a message msg of type msgtype is sent to the mailbox, PXROS activates the mailbox handler assigned to the mailbox with the call:

```
hnd(&msg, msgtype, arg) .
```

PXROS expects diagnostic information from the handler:

- If the handler returns PxMsgId(id == _PXIllegalObjId, error == PXERR_NOERROR) PXROS assumes that the message has been completely processed by the handler. The application's send call then returns successfully, and the message is not placed in the mailbox.
- If the handler returns PxMsgId(id != _PXIllegalObjId, error == PXERR_NOERROR) PXROS places the message in the mailbox. The application's send call returns successfully.
- If the handler returns PxMsgId(id != _PXIllegalObjId, error != PXERR_NOERROR), PXROS assumes that the handler has refused the message by returning an error indicator. The application's send call then fails by propagating the handler's error code.

 When the mailbox handler is called the message's user is set to the installer task of the handler, so the mailbox handler may call all message-related functions that require a valid user.

7.3 PxBxRelease

NAME

PxBxRelease() - release a mailbox object

SYNOPSIS

```
#include <pxdef.h>
```

```
PxBx_t
PxBxRelease (PxBx_t Mbx) ;
```

PARAMETERS

Mbx mailbox object to be released

RETURN VALUES

- invalid mailbox handle on success
- mailbox on failure

ERROR CODES

PXERR_MBX_ILLMBX - Mbx is not a valid mailbox object

SEE ALSO

- PxBxRequest()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxBxRelease releases the mailbox handle Mbx by converting it into a generic object and releasing this object.

7.4 PxBxRequest

NAME

PxBxRequest() - request a mailbox handle

PxBxRequest_EvWait() - request a mailbox handle while waiting for events

PxBxRequest_NoWait() - request a mailbox handle with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxBx_t
PxBxRequest (PxOpool_t opoolid) ;

PxBx_t
PxBxRequest_EvWait (PxOpool_t opoolid,
                    PxEvents_t events) ;
```

```
PxBx_t
PxBxRequest_NoWait (PxOpool_t opoolid) ;
```

PARAMETERS

opoolid the object pool, where the mailbox object is requested from.

Parameters of `PxMbxRequest_EvWait()`

`events` eventmask, that makes the call return

RETURN VALUES

- invalid mailbox handle on failure
- mailbox on success

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool

Exceptions of `PxMbxRequest_EvWait()`

PXERR_EVENT_ZERO - the given event mask is zero

PXERR_OBJ_ABORTED - request aborted by an event

Exceptions of `PxMbxRequest_NoWait()`

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- `PxMbxRelease()`
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxMbxRequest...` functions create a mailbox handle by converting a generic object from object pool `opoolid`. The handle's identifier is returned

The functions act differently if there is no object available. In such a case `PxMbxRequest_NoWait` fails, `PxMbxRequest` waits until a free object is available, and `PxMbxRequest_EvWait` waits until either there is a free object or an event specified in the set events occurs.

8 Memory Management

8.1 PxMcGetSize

NAME

PxMcGetSize() - get size of a memory class

SYNOPSIS

```
#include <pxdef.h>
```

```
PxSize_t  
PxMcGetSize(PxMc_t mcid);
```

PARAMETERS

mcid the memory class object

RETURN VALUES

- 0 if mcid is varsize
- blocksize if mcid is fixsize

ERROR CODES

PXERR_MC_ILLMC - mc is not a valid memory class

SEE ALSO

- PxMcGetType()
- PxMcGetVarsizedOverhead()
- PxMcInsertBlk()
- PxMcRelease()
- PxMcRequest()
- PxMcResolveDefault()
- PxMcReturnBlk()
- PxMcTakeBlk()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMcGetSize returns 0 if mcid is varsize or blocksize if mcid is fixsize .

8.2 PxMcGetType

NAME

PxMcGetType() - get type of a memory class

SYNOPSIS

```
#include <pxdef.h>

PxMcType_t
PxMcGetType(PxMc_t mcid);
```

PARAMETERS

`mcid` the memory class object

RETURN VALUES

- the memory classes type

ERROR CODES

PXERR_MC_ILLMC - `mcid` is not a valid memory class

SEE ALSO

- [PxMcGetSize\(\)](#)
- [PxMcGetVarsizedOverhead\(\)](#)
- [PxMcInsertBlk\(\)](#)
- [PxMcRelease\(\)](#)
- [PxMcRequest\(\)](#)
- [PxMcResolveDefault\(\)](#)
- [PxMcReturnBlk\(\)](#)
- [PxMcTakeBlk\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxMcGetType` returns the memory classes type:

- `PXMcVarsized`
- `PXMcVarsizedAdjusted`
- `PXMcVarsizedAligned`
- `PXMcFixsized`
- `PXMcTypeLast` if an invalid memory class is given.

8.3 PxMcGetVarsizedOverhead

NAME

`PxMcGetVarsizedOverhead()` - return the overhead of a varsized memory class

SYNOPSIS

```
#include <pxdef.h>

PxSize_t
PxMcGetVarsizedOverhead(PxMc_t mcid);
```


PARAMETERS

`mcid` the memory class object

RETURN VALUES

- overhead per block of a var-sized memory class
- `PXMAXUINT` in case of error

ERROR CODES

`PXERR_MC_ILLMC` - `mcid` is not a valid memory class

SEE ALSO

- [PxMcGetSize\(\)](#)
- [PxMcGetType\(\)](#)

DESCRIPTION

`PxMcGetVarsizedOverhead` returns the overhead per block of a var-sized memory class.

8.4 PxmclInsertBlk

NAME

`PxmclInsertBlk()` - insert a new block into a memory class

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMcInsertBlk(PxMc_t mcId,
              PxMptr_t blk,
              PxSize_t size);
```

PARAMETERS

`mcId` memory class

`blk` the memory block to insert

`size` size of the memory block

RETURN VALUES

- PXROS error code

ERROR CODES

`PXERR_ACCESS_RIGHT` - the calling task has not the right to access the memory class

`PXERR_MC_ILLMC` - `mc` is not a valid memory class object

SEE ALSO

- [PxMcGetSize\(\)](#)
- [PxMcGetType\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxmclInsertBlk` inserts a new block of the size `size` into the memory class `mc`.

8.5 PxMcRelease

NAME

PxMcRelease() - release a memory class object

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMc_t
PxMcRelease (PxMc_t Mc);
```

PARAMETERS

Mc memory class object to be released

RETURN VALUES

- invalid memory class handle on success
- Mc on failure

ERROR CODES

PXERR_MC_ILLMC - Mc is not a valid memory class object

SEE ALSO

- PxMcGetSize()
- PxMcGetType()
- PxMcRequest()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMcRelease releases the memory class handle Mc by converting it into a generic object and releasing this object.

8.6 PxMcRequest

NAME

PxMcRequest() - request a memory class

PxMcRequest_EvWait() - request a memory class while waiting for events

PxMcRequest_NoWait() - request a memory class with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMc_t
PxMcRequest (PxMcType_t mctype,
             PxSize_t size,
             PxOpool_t opoolid);
```

```
PxMc_t
PxMcRequest_EvWait (PxMcType_t mctype,
                   PxSize_t size,
                   PxOpool_t opoolid,
```

```

        PxEvents_t events);

PxMc_t
PxMcRequest_NoWait (PxMcType_t mctype,
                   PxSize_t size,
                   PxOpool_t opoolid);

```

PARAMETERS

mctype the memory class type

size the memory class size

opoolid the object pool, where the memory class object is requested from.

Parameters of PxMcRequest_EvWait()

events eventmask, that make the call return

RETURN VALUES

- invalid memory class handle on failure
- memory class on success

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool

Exceptions of PxMcRequest_EvWait()

PXERR_OBJ_ABORTED - request aborted by an event

Exceptions of PxMcRequest_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- PxMcGetSize()
- PxMcGetType()
- PxMcRelease()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMcRequest... functions create a memory class handle by converting a generic object from object pool opoolid. The handle's identifier is returned

The following memory class types exist:

- PXMcFixsized
- PXMcVarsized
- PXMcVarsizedAligned
- PXMcVarsizedAdjusted

The functions act differently if there is no object available. In such a case `PxMcRequest_NoWait` fails, `PxMcRequest` waits until a free object is available, and `PxMcRequest_EvWait` waits until either there is a free object or an event specified in the set events occurs.

8.7 PxMcResolveDefault

NAME

`PxMcResolveDefault()` - resolve a memory class default

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMc_t
```

```
PxMcResolveDefault (PxMc_t mcid);
```

PARAMETERS

`mcid` memory class to resolve

RETURN VALUES

- resolved memory class

ERROR CODES

`PXERR_MC_ILLMC` - `mcid` is not a valid memory class.

SEE ALSO

- [PxMcGetSize\(\)](#)
- [PxMcGetType\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxMcResolveDefault` resolves a memory class default. More precisely, it returns `mcid`, if `mcid` is not `PXMcSystemdefault` or `PXMcTaskdefault`; otherwise it returns the memory class id corresponding to the specified default.

8.8 PxMcReturnAllBlks

NAME

`PxMcReturnAllBlks()` - return all allocated memory blocks of the caller

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxMcReturnAllBlks (void);
```

RETURN VALUES

- all errors from `PxMcReturnBlk`

ERROR CODES

`PXERR_MC_ILLMC` - `mcid` is not a valid memory class

DESCRIPTION

`PxMcReturnAllBlks` returns all blocks which are allocated by the calling task.

8.9 PxMcReturnBlk

NAME

PxMcReturnBlk() - return a block to the memory class

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMcReturnBlk(PxMc_t mcid,
              PxMptr_t blk);
```

PARAMETERS

mcid the memory class handle, the block is returned to

blk the block to be returned

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - the requested memory class is not on the same core

PXERR_MC_DAMAGED_BLOCK - block is damaged

PXERR_MC_ILLMC - mcid is not a valid memory class

PXERR_MC_NOT_ALLOCATED - block is not allocated

PXERR_MC_NOT_ALLOCATED_FROM - block not taken from the memory class

SEE ALSO

- PxMcGetSize()
- PxMcGetType()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMcReturnBlk returns block blk to mcid. blk is set to zero. blk must have been gotten by a PxMcTakeBlk call from the same memory class as it is returned to.

8.10 PxMcTakeBlk

NAME

PxMcTakeBlk() - take a block from memory class

SYNOPSIS

```
#include <pxdef.h>

PxMptr_t
PxMcTakeBlk(PxMc_t mcid,
            PxSize_t size);
```

PARAMETERS

mcid the memory class handle, the block is taken from

`size` requested size

RETURN VALUES

- start address of block on success
- null pointer on failure

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the memory class

PXERR_GLOBAL_ILLEGAL_CORE - the requested memory class is not on the same core

PXERR_MC_ILLMC - mcid is not a valid memory class

PXERR_MC_NOMEM - not enough memory in the memory class to satisfy the request

PXERR_MC_USE_BUDDY_FOR_MSG_ONLY - buddy memory classes must only be used for messages

SEE ALSO

- [PxMcGetSize\(\)](#)
- [PxMcGetType\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMcTakeBlk takes a block of a size no less than `size` (specified in byte units) from `mcid` and returns its address.

9 Message-related Services

9.1 PxMsgAwaitRel

NAME

PxMsgAwaitRel() - waits until a message is released.

PxMsgAwaitRel_EvWait() - waits until a message is released or one of the given events arrives while waiting for events

PxMsgAwaitRel_NoWait() - checks if a message is released with immediate return

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgAwaitRel (PxMsg_t Msg);

PxMsgEvent_t
PxMsgAwaitRel_EvWait (PxMsg_t Msg,
                      PxEvents_t events);

PxMsg_t
PxMsgAwaitRel_NoWait (PxMsg_t Msg);
```

PARAMETERS

Msg message object to await release

Parameters of PxMsgAwaitRel_EvWait()

events event mask to make the call return

RETURN VALUES

- Msg on success
- invalid message handle on failure

Returnvalues of PxMsgAwaitRel_EvWait()

- event that made the function return

ERROR CODES

PXERR_MSG_ABORTED - service was aborted by an event

PXERR_MSG_ILMSG - the passed message handle is invalid

PXERR_MSG_ILLOWNER - the calling task is not the owner of the message

PXERR_MSG_NOT_IMPLEMENTED - the message has not been set to AwaitRelease

Exceptions of PxMsgAwaitRel_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

Exceptions of PxMsgAwaitRel_NoWait()

PXERR_MSG_NOMSG - the message has been released

SEE ALSO

- [PxMsgEnvelop\(\)](#)
- [PxMsgSetToAwaitRel\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

[PxMsgAwaitRel...](#) checks if a message, created by [PxMsgEnvelop](#) or set to [AwaitRelease](#), is released. To use this function, the calling task must be the owner of the message which release is being awaited.

9.2 PxMsgEnvelop

NAME

[PxMsgEnvelop\(\)](#) - envelops a data area in a newly created message object

[PxMsgEnvelop_EvWait\(\)](#) - envelops a data area in a newly created message object while waiting for events

[PxMsgEnvelop_NoWait\(\)](#) - envelops a data area in a newly created message object with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMsg_t
PxMsgEnvelop(PxMsgData_t data_area,
             PxSize_t msgsize,
             PxOpool_t opoolid);

PxMsgEvent_t
PxMsgEnvelop_EvWait(PxMsgData_t data_area,
                   PxSize_t msgsize,
                   PxOpool_t opoolid,
                   PxEvents_t events);

PxMsg_t
PxMsgEnvelop_NoWait(PxMsgData_t data_area,
                   PxSize_t msgsize,
                   PxOpool_t opoolid);
```

PARAMETERS

<code>data_area</code>	the data area to be enveloped
<code>msgsize</code>	the data area's size
<code>opoolid</code>	object pool where to take the message object from

Parameters of [PxMsgEnvelop_EvWait\(\)](#)

<code>events</code>	eventmask that make the call return
---------------------	-------------------------------------

RETURN VALUES

- the received message handle

Returnvalues of PxMsgEnvelop_EvWait()

- the received message handle or the events, that caused the function to return.

Returnvalues of PxMsgEnvelop_NoWait()

- an invalid message handle on failure

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - the passed object pool handle is invalid

PXERR_PROT_PERMISSION - data_area is not accessible for the calling task

Exceptions of PxMsgEnvelop_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

Exceptions of PxMsgEnvelop_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- PxMsgAwaitRel()
- PxMsgGetProtection()
- PxMsgSetProtection()
- PxMsgSetToAwaitRel()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgEnvelop... envelops a data area in a message object taken from the object pool opoolid. The function waits until a free object is available. It is possible to wait for the message release with PxMsgAwaitRel. The calling task becomes the (permanent) owner and (temporary) user of the message created. The task's access rights to the data area is marked in the message object and can be read by calling PxMsgGetProtection. The owner may restrict the access right by calling PxMsgSetProtection.

9.3 PxMsgForceRelease

NAME

PxMsgForceRelease() - release messages if PxRuntask is owner/user

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxMsgForceRelease(PxMsg_t msgId);
```

PARAMETERS

msgId message to release

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MC_ILLMC - memory class for the message is invalid

PXERR_MSG_ILLMSG - message is invalid

PXERR_MSG_ILLOWNER - message has no owner

PXERR_OPOOL_ILLOPOOL - object pool for the message is invalid

SEE ALSO

- [PxMsgRelease\(\)](#)
- [PxMsgReleaseAllMsg\(\)](#)

DESCRIPTION

PxMsgForceRelease will release the message if the caller is the user of the message. If the caller is the owner of the message, PxMsgForceRelease marks this message to release. The flag for PxMsgAwaitRel is cleared and any defined release mailbox will be deleted.

9.4 PxMsgGetBuffersize

NAME

PxMsgGetBuffersize() - return the size of message's corresponding data area.

SYNOPSIS

```
#include <pxdef.h>
```

```
PxSize_t
PxMsgGetBuffersize(PxMsg_t msgid);
```

PARAMETERS

`msgid` the message object

RETURN VALUES

- message data area size

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid message object

PXERR_MSG_ILLUSER - caller is not the user of this message

SEE ALSO

- [PxMsgRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgGetBuffersize returns the messages data area size. If an invalid message is given, 0 is returned.

9.5 PxMsgGetData

NAME

PxMsgGetData() - get data area of a message (task service)

PxMsgGetData_Hnd() - get data area of a message (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxMsgData_t
PxMsgGetData (PxMsg_t msgid);

PxMsgData_t
PxMsgGetData_Hnd (PxMsg_t msgid);
```

PARAMETERS

`msgid` message object

RETURN VALUES

- pointer to message data area

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid message object

PXERR_MSG_ILLUSER - task is not user of this message

PXERR_PROT_NOFREE_ENTRY - no free protection entry

Exceptions of PxMsgGetData_Hnd()

PXERR_MSG_NOT_IMPLEMENTED - function is called by a system interrupt handler

SEE ALSO

- [PxMsgReceive\(\)](#)
- [PxMsgRelDataAccess\(\)](#)
- [PxMsgRequest\(\)](#)
- [PxMsgSend\(\)](#)
- [PxMsgSetData\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgGetData returns a C-pointer to the data area corresponding to message msgid. If an invalid message is given a null pointer is returned. PxMsgGetData_Hnd is the corresponding handler function. It should only be used from handlers running in a task's context.

9.6 PxMsgGetMetadata

NAME

PxMsgGetMetadata() - get the metadata of the message

PxMsgGetMetadata_Hnd() - get the metadata of the message

SYNOPSIS

```
#include <pxdef.h>

PxMsgMetadata_t
```

```
PxMsgGetMetadata(PxMsg_t msgid);

PxMsgMetadata_t
PxMsgGetMetadata_Hnd(PxMsg_t msgid);
```

PARAMETERS

`msgid` the message object

RETURN VALUES

- message metadata

ERROR CODES

PXERR_MSG_ILLMSG - `msgid` is not a valid message object

PXERR_MSG_ILLUSER - calling task is not user of this message

Exceptions of PxMsgGetMetadata_Hnd()

PXERR_MSG_NOT_IMPLEMENTED - function is called by a system interrupt handler

SEE ALSO

- [PxMsgRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxMsgGetMetadata` returns the metadata of the message. The size of the metadata is 8 byte. `PxMsgGetMetadata_Hnd` is the corresponding handler function. It should only be used from handlers running in a task's context.

9.7 PxMsgGetOwner

NAME

`PxMsgGetOwner()` - return the owner taskid of the message

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTask_t
PxMsgGetOwner(PxMsg_t msgid);
```

PARAMETERS

`msgid` the message object

RETURN VALUES

- message owner

ERROR CODES

PXERR_MSG_ILLMSG - `msgid` is not a valid object id

PXERR_MSG_ILLUSER - calling task is not user of this message

SEE ALSO

- [PxMsgRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgGetOwner returns the messages owner.

9.8 PxMsgGetProtection

NAME

PxMsgGetProtection() - get protection mode of a message

SYNOPSIS

```
#include <pxdef.h>

PxProtectType_t
PxMsgGetProtection(PxMsg_t msgid);
```

PARAMETERS

`msgid` message object for which the protection mode is requested

RETURN VALUES

- protection mode (NoAccessProtection, ReadProtection, WriteProtection, WRProtection)

ERROR CODES

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLUSER - the calling task is not the user of the message

SEE ALSO

- [PxMsgEnvelop\(\)](#)

DESCRIPTION

PxMsgGetProtection returns the protection mode for the data area of a given message object msgid. The following values are possible:

- NoAccessProtection the caller has no access to the messages data area
- ReadProtection the caller has read only access to the messages data area
- WriteProtection the caller has write only access to the messages data area
- WRProtection the caller has read and write access to the messages data area

9.9 PxMsgGetSender

NAME

PxMsgGetSender() - return the sender taskid of the message

SYNOPSIS

```
#include <pxdef.h>

PxTask_t
PxMsgGetSender(PxMsg_t msgid);
```

PARAMETERS

`msgid` the message object

RETURN VALUES

- message sender

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid object id

PXERR_MSG_ILLUSER - calling task is not user of this message

SEE ALSO

- [PxMsgReceive\(\)](#)
- [PxMsgRequest\(\)](#)
- [PxMsgSend\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgGetSender returns the messages sender. If the message has been sent by a handler the previous user of the message is returned.

9.10 PxMsgGetSize

NAME

PxMsgGetSize() - return the message size

SYNOPSIS

```
#include <pxdef.h>
```

```
PxSize_t  
PxMsgGetSize(PxMsg_t msgid);
```

PARAMETERS

msgid the message object

RETURN VALUES

- message size

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid object id

PXERR_MSG_ILLUSER - caller is not the user of this message

SEE ALSO

- [PxMsgRequest\(\)](#)
- [PxMsgSetSize\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgGetSize returns the size of message msgid. If an invalid message is given, 0 is returned.

9.11 PxMsgInstallRelmbx

NAME

PxMsgInstallRelmbx() - install a mailbox as a release mailbox

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
PxMsgInstallRelmbx(PxMsg_t msgid,
                  PxMbx_t mbxid);
```

PARAMETERS

msgid message object for which the release mailbox has to be installed

mbxid release mailbox for the message object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MBX_ILLMBX - the passed mailbox handle is invalid

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLUSER - the calling task is not the user of the message

PXERR_MSG_NOT_IMPLEMENTED - the message has not been requested

PXERR_MSG_RELMBX_INSTALLED - a release mailbox is already installed for the message

SEE ALSO

- [PxMsgSend\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgInstallRelmbx installs mbxid as release mailbox for msgid, if mbxid is a valid mailbox object; if mbxid is an illegal object a potential release mailbox for msgid is de-installed. To use this service, the calling task must be the user of msgid.

If a release mailbox mbxid is installed at msgid and msgid is later released, msgid is not destroyed but sent to mbxid.

9.12 PxMsgReceive

NAME

PxMsgReceive() - receive a message from a mailbox

PxMsgReceive_EvWait() - receive a message from a mailbox while waiting for events

PxMsgReceive_NoWait() - receive a message from a mailbox with immediate return

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgReceive(PxMbx_t mbxid);

PxMsgEvent_t
PxMsgReceive_EvWait(PxMbx_t mbxid,
                    PxEvents_t events);

PxMsg_t
PxMsgReceive_NoWait(PxMbx_t mbxid);
```

PARAMETERS

`mbxid` the mailbox handle

Parameters of `PxMsgReceive_EvWait()`

`events` eventmask to wait for

RETURN VALUES

- the received message handle

Returnvalues of `PxMsgReceive_EvWait()`

- the received message handle or/and the received events

ERROR CODES

PXERR_MBX_ILLMBX - the passed mailbox handle is invalid

Exceptions of `PxMsgReceive_EvWait()`

PXERR_EVENT_ZERO - the given event mask is zero

PXERR_MSG_NOMSG - no message available

SEE ALSO

- `PxMbxInstallHnd()`
- `PxMsgGetData()`
- `PxMsgGetSender()`
- `PxMsgRelDataAccess()`
- `PxMsgSend()`
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxMsgReceive...` receives a message object from mailbox `mbxid` and returns the received message handle. The task becomes the user of the received message.

9.13 PxMsgRelDataAccess

NAME

`PxMsgRelDataAccess()` - release the access to the data area of a message (task service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMsgRelDataAccess (PxMsg_t msgid);
```

PARAMETERS

`msgid` message object

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid message object

PXERR_MSG_ILLUSER - calling task is not user of this message

PXERR_PROT_ILL_HANDLE - the associated protection handle is not valid

SEE ALSO

- [PxMsgGetData\(\)](#)
- [PxMsgReceive\(\)](#)
- [PxMsgRequest\(\)](#)
- [PxMsgSend\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgRelDataAccess releases the access to the data area of the message msgid. It is required that the calling task is the current user of msg. After calling PxMsgRelDataAccess no access to the data area of the message msg is possible. The access right may be recovered through PxMsgGetData.

9.14 PxMsgRelease

NAME

PxMsgRelease() - release a message object (task service)

PxMsgRelease_Hnd() - release a message object (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgRelease (PxMsg_t Msg);

PxMsg_t
PxMsgRelease_Hnd (PxMsg_t Msg);
```

PARAMETERS

`Msg` message object to be released

RETURN VALUES

- invalid message handle on success

- Msg on failure

ERROR CODES

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLUSER - the calling task is not the user of the message

Exceptions of PxBMsgRelease_Hnd()

PXERR_MBX_ILLMBX - the passed message's release mailbox handle is invalid

PXERR_MSGREL_NOT_INITED - the message release server is not yet initialized

SEE ALSO

- PxBMsgForceRelease()
- PxBMsgReleaseAllMsg()
- PxBMsgRequest()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxBMsgRelease... release the message specified in Msg. All necessary cleanup operations are performed. For a message created with PxBMsgEnvelop..., the release service checks if a task is waiting for the message being released. If so, the task is readied. Before calling PxBMsgRelease_Hnd, one must initialize the message release service with PxBMsgrelServiceInit. PxBMsgRelease_Hnd sends the message to mailbox instantiated with PxBMsgrelServiceInit and sends the event PXSERVICE_HND_MSGREL to the PXROS service task.

9.15 PxBMsgReleaseAllMsg

NAME

PxBMsgReleaseAllMsg() - release all messages of PxBRuntime

SYNOPSIS

```
#include <pxdef.h>
```

```
PxBError_t
```

```
PxBMsgReleaseAllMsg(void);
```

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MC_ILLMC - memory class for the message is invalid

PXERR_MSG_ILLMSG - message is invalid

PXERR_MSG_ILLOWNER - message has no owner

PXERR_OPOOL_ILLOPOOL - object pool for the message is invalid

SEE ALSO

- PxBMsgForceRelease()

- [PxMsgRelease\(\)](#)

DESCRIPTION

PxMsgReleaseAllMsg will release all messages where the caller is user. If the caller is the owner of the message, PxMsgForceRelease marks this message to release. The flag for PxMsgAwaitRel is cleared and any defined release mailbox will be deleted.

9.16 PxMsgRequest

NAME

PxMsgRequest() - create a message object together with a data area

PxMsgRequest_EvWait() - create a message object together with data area while waiting for events

PxMsgRequest_NoWait() - create a message object together with data area with immediate return

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgRequest(PxSize_t msgsize,
             PxMc_t mcId,
             PxOpool_t opoolId);

PxMsgEvent_t
PxMsgRequest_EvWait(PxSize_t msgsize,
                   PxMc_t mcId,
                   PxOpool_t opoolId,
                   PxEvents_t events);

PxMsg_t
PxMsgRequest_NoWait(PxSize_t msgsize,
                   PxMc_t mcId,
                   PxOpool_t opoolId);
```

PARAMETERS

<code>msgsize</code>	size of the message
<code>mcId</code>	memory class where to take the buffer from
<code>opoolId</code>	object pool where to take the message object from

Parameters of PxMsgRequest_EvWait()

<code>events</code>	event mask to make the call return
---------------------	------------------------------------

RETURN VALUES

- the requested message handle

Returnvalues of PxMsgRequest_EvWait()

- the requested message handle or the events, that caused the return.

Returnvalues of PxMsgRequest_NoWait()

- an invalid message handle on failure

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool or the memory class

PXERR_GLOBAL_ILLEGAL_CORE - the passed memory class is not on the same core

PXERR_MC_ILLMC - the passed memory class handle is invalid

PXERR_MC_NOMEM - not enough memory

PXERR_OPOOL_ILLOPOOL - the passed object pool handle is invalid

Exceptions of PxMsgRequest_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- [PxMsgGetBuffersize\(\)](#)
- [PxMsgGetData\(\)](#)
- [PxMsgGetMetadata\(\)](#)
- [PxMsgGetOwner\(\)](#)
- [PxMsgGetSender\(\)](#)
- [PxMsgGetSize\(\)](#)
- [PxMsgRelDataAccess\(\)](#)
- [PxMsgRelease\(\)](#)
- [PxMsgSetData\(\)](#)
- [PxMsgSetMetadata\(\)](#)
- [PxMsgSetSize\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgRequest family functions create messages with a data area of size msgsize (specified in byte units). The data area is taken from memory class mc, and the message structure is taken from object pool opool. The message object handle is returned. The calling task becomes the (permanent) owner and the (temporary) user of the message created. The call also fails, if the memory class does not contain sufficient memory to meet the request. A fatal PXROS error occurs, if mc is fixsize and its block size is smaller than msgsize. If msgsize is zero, no data buffer is requested. Only the metadata of the message can be used.

9.17 PxMsgSend

NAME

PxMsgSend() - send normal message (task service)

PxMsgSend_Hnd() - send normal message (handler service)

PxMsgSend_Prio() - send prioritized message (task service)

PxMsgSend_PrioHnd() - send prioritized message (handler service)

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMsg_t
PxMsgSend(PxMsg_t msg,
          PxMbx_t mbx);
```

```
PxMsg_t
PxMsgSend_Hnd(PxMsg_t msg,
              PxMbx_t mbx);
```

```
PxMsg_t
PxMsgSend_Prio(PxMsg_t msg,
              PxMbx_t mbx);
```

```
PxMsg_t
PxMsgSend_PrioHnd(PxMsg_t msg,
                  PxMbx_t mbx);
```

PARAMETERS

`msg` the message handle to send

`mbx` the mailbox handle

RETURN VALUES

- invalid message handle on success
- msg including PXROS error code on failure

ERROR CODES

PXERR_MBX_ILLMBX - the mailbox handle is invalid

PXERR_MSG_ILLMSG - the message handle is invalid

PXERR_MSG_ILLUSER - the sending task is not user of the message

SEE ALSO

- [PxMbxInstallHnd\(\)](#)
- [PxMsgGetData\(\)](#)
- [PxMsgGetSender\(\)](#)
- [PxMsgInstallRelmbx\(\)](#)
- [PxMsgReceive\(\)](#)

- [PxMsgRelDataAccess\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgSend.. functions send the message Msg into the mailbox mbxid. The calling task must be the user of Msg. The handler service requires that Msg has a user. After the message is sent, it does not have a user (temporarily).

9.18 PxMsgSetData

NAME

PxMsgSetData() - set message data pointer

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMsgSetData(PxMsg_t msgid,
             PxMsgData_t new_data);
```

PARAMETERS

`msgid` message object for which the new data pointer is set

`new_data` new message data

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLOWNER - the calling task is not the owner of the message

PXERR_MSG_ILLUSER - the calling task is not the user of the message

PXERR_MSG_ILL_NEW_DATA - new data pointer not within the corresponding data area.

SEE ALSO

- [PxMsgGetData\(\)](#)
- [PxMsgRequest\(\)](#)
- [PxMsgSetSize\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgSetData sets the data pointer of message msgid to new_data. The calling task must be the user of msgid. new_data must be part of the messages data area. if new_data is NULL, the message's data pointer is reset to the start of the data area. In this case the calling task must be the owner of msgid.

9.19 PxMsgSetMetadata

NAME

PxMsgSetMetadata() - set the metadata for the message

PxMsgSetMetadata_Hnd() - set the metadata for the message

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxMsgSetMetadata(PxMsg_t msgid,  
                 PxMsgMetadata_t metadata);
```

```
PxError_t
```

```
PxMsgSetMetadata_Hnd(PxMsg_t msgid,  
                     PxMsgMetadata_t metadata);
```

PARAMETERS

`msgid` the message object

`metadata` the metadata

ERROR CODES

PXERR_MSG_ILLMSG - msgid is not a valid message object

PXERR_MSG_ILLUSER - calling task is not user of this message

Exceptions of PxMsgSetMetadata_Hnd()

PXERR_MSG_NOT_IMPLEMENTED - function is called by a system interrupt handler

SEE ALSO

- PxMsgRequest()
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgSetMetadata sets the metadata for the message. The size of the metadata is 8 byte. PxMsgSetMetadata_Hnd is the corresponding handler function. It should only be used from handlers running in a task's context.

9.20 PxMsgSetProtection

NAME

PxMsgSetProtection() - set protection mode of a message

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxMsgSetProtection(PxMsg_t msgid,  
                   PxProtectType_t protection);
```

PARAMETERS

`msgid` message object for which the protection mode is changed

`protection` new protection mode for the message data.

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLUSER - the calling task is not the user of the message

PXERR_PROT_PERMISSION - the calling task has no permission to change the protection.

SEE ALSO

- `PxMsgEnvelop()`

DESCRIPTION

`PxMsgSetProtection` sets the protection mode for the data area of a given message object `msgid`. The Task has to be the owner (creator) of the message `msgid`! The following values are possible:

- `NoAccessProtection` the caller has no access to the messages data area
- `ReadProtection` the caller has read only access to the messages data area
- `WriteProtection` the caller has write only access to the messages data area
- `WRProtection` the caller has read and write access to the messages data area

The protection mode can only be changed if the caller has the appropriate permission. For example the caller cannot set `WriteProtection` if it has no write access to the data area.

9.21 PxMsgSetSize

NAME

`PxMsgSetSize()` - set message size

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
PxMsgSetSize(PxMsg_t msgid,
             PxSize_t size);
```

PARAMETERS

`msgid` message object for which the size is set

`size` new message size

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MSG_ILLMSG - the passed message handle is invalid

PXERR_MSG_ILLSIZE - size exceeds the size of the corresponding data area.

PXERR_MSG_ILLUSER - the calling task is not the user of the message

SEE ALSO

- [PxMsgGetSize\(\)](#)
- [PxMsgRequest\(\)](#)
- [PxMsgSetData\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgSetSize sets the size of message msgid to size (specified in byte units). The calling task must be the user of msgid.

9.22 PxMsgSetToAwaitRel

NAME

PxMsgSetToAwaitRel() - sets the message to AwaitRelease.

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMsgSetToAwaitRel(PxMsg_t Msg);
```

PARAMETERS

Msg the message object to be marked

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MSG_ILLMSG - Msg is not a valid message object

PXERR_MSG_ILLOWNER - calling task is not the message's owner

SEE ALSO

- [PxMsgAwaitRel\(\)](#)
- [PxMsgEnvelop\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxMsgSetToAwaitRel sets the message to AwaitRelease. Further calls to PxMsgRelease will only mark this message as released but don't really release them. The owner of this message has to wait for the release and is also responsible for releasing the message. This call is only valid for the message owner.

10 Object-related Services

10.1 PxDelayIdError

NAME

PxDelayIdError() - return the last error of the object handle Delay

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxDelayIdError(PxDelay_t Delay);
```

PARAMETERS

Delay a Delay handle

RETURN VALUES

- the last error of the object handle Delay

DESCRIPTION

PxDelayIdError returns the last error of the object handle Delay

10.2 PxDelayIdGet

NAME

PxDelayIdGet() - return the object ID of the object handle Delay

SYNOPSIS

```
#include <pxdef.h>
```

```
PxObjId_t  
PxDelayIdGet(PxDelay_t Delay);
```

PARAMETERS

Delay a Delay handle

RETURN VALUES

- object ID of the object handle Delay

DESCRIPTION

PxDelayIdGet returns the object ID of the object handle Delay

10.3 PxDelayIdInvalidate

NAME

PxDelayIdInvalidate() - return an invalid Delay handle for initialization

SYNOPSIS

```
#include <pxdef.h>
```

```
PxDelay_t
PxDelayIdInvalidate(void);
```

RETURN VALUES

- an invalid Delay handle

DESCRIPTION

PxDelayIdInvalidate returns an invalid Delay handle for initialization

10.4 PxDelayIdsValid

NAME

PxDelayIdsValid() - return TRUE if Delay holds a valid Delay handle

SYNOPSIS

```
#include <pxdef.h>

int
PxDelayIdIsValid(PxDelay_t Delay);
```

PARAMETERS

Delay a Delay handle

RETURN VALUES

- TRUE if Delay is a valid Delay handle

DESCRIPTION

PxDelayIdsValid returns TRUE if obj holds a valid Delay handle.

10.5 PxDelayIdResetError

NAME

PxDelayIdResetError() - reset the last error of the object handle Delay

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxDelayIdResetError(PxDelay_t *Delay);
```

PARAMETERS

Delay pointer to the Delay handle

RETURN VALUES

- the last error of the object handle Delay

DESCRIPTION

PxDelayIdResetError resets the last error of the object handle Delay

10.6 PxDelayIdSet

NAME

PxDelayIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxDelay_t
PxDelayIdSet (PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Delay handle

DESCRIPTION

PxDelayIdSet returns an object handle which holds id as the object ID

10.7 PxGetObjectsize

NAME

PxGetObjectsize() - return the size of an object

SYNOPSIS

```
#include <pxdef.h>

PxSize_t
PxGetObjectsize (void);
```

RETURN VALUES

- size of a generic PXROS object

DESCRIPTION

PxGetObjectsize returns the size (in bytes) for an object representation in the current PXROS version.

10.8 PxInterruptIdError

NAME

PxInterruptIdError () - return the last error of the object handle Interrupt

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxInterruptIdError (PxInterrupt_t Interrupt);
```

PARAMETERS

Interrupt a Interrupt handle

RETURN VALUES

- the last error of the object handle Interrupt

DESCRIPTION

PxInterruptIdError returns the last error of the object handle Interrupt

10.9 PxInterruptIdGet

NAME

PxInterruptIdGet () - return the object ID of the object handle Interrupt

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxInterruptIdGet (PxInterrupt_t Interrupt);
```

PARAMETERS

Interrupt a Interrupt handle

RETURN VALUES

- object ID of the object handle Interrupt

DESCRIPTION

PxInterruptIdGet returns the object ID of the object handle Interrupt

10.10 PxInterruptIdInvalidate

NAME

PxInterruptIdInvalidate () - return an invalid Interrupt handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxInterrupt_t
PxInterruptIdInvalidate (void);
```

RETURN VALUES

- an invalid Interrupt handle

DESCRIPTION

PxInterruptIdInvalidate returns an invalid Interrupt handle for initialization

10.11 PxInterruptIdsValid

NAME

PxInterruptIdsValid () - return TRUE if Interrupt holds a valid Interrupt handle

SYNOPSIS

```
#include <pxdef.h>

int
PxInterruptIdsValid (PxInterrupt_t Interrupt);
```

PARAMETERS

Interrupt a Interrupt handle

RETURN VALUES

- TRUE if Interrupt is a valid Interrupt handle

DESCRIPTION

PxInterruptIdIsValid returns TRUE if obj holds a valid Interrupt handle.

10.12 PxInterruptIdResetError

NAME

PxInterruptIdResetError () - reset the last error of the object handle Interrupt

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxInterruptIdResetError (PxInterrupt_t *Interrupt);
```

PARAMETERS

Interrupt pointer to the Interrupt handle

RETURN VALUES

- the last error of the object handle Interrupt

DESCRIPTION

PxInterruptIdResetError resets the last error of the object handle Interrupt

10.13 PxInterruptIdSet

NAME

PxInterruptIdSet () - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>
```

```
PxInterrupt_t
```

```
PxInterruptIdSet (PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Interrupt handle

DESCRIPTION

PxInterruptIdSet returns an object handle which holds id as the object ID

10.14 PxMbxIdError

NAME

PxMbxIdError() - return the last error of the object handle Mbx

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxMbxIdError (PxMbx_t Mbx);
```

PARAMETERS

Mbx a Mbx handle

RETURN VALUES

- the last error of the object handle Mbx

DESCRIPTION

PxMbxIdError returns the last error of the object handle Mbx

10.15 PxMbxIdGet

NAME

PxMbxIdGet() - return the object ID of the object handle Mbx

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxMbxIdGet (PxMbx_t Mbx);
```

PARAMETERS

Mbx a Mbx handle

RETURN VALUES

- object ID of the object handle Mbx

DESCRIPTION

PxMbxIdGet returns the object ID of the object handle Mbx

10.16 PxMbxIdInvalidate

NAME

PxMbxIdInvalidate() - return an invalid Mbx handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxMbx_t
PxMbxIdInvalidate (void);
```

RETURN VALUES

- an invalid Mbx handle

DESCRIPTION

PxMbxIdInvalidate returns an invalid Mbx handle for initialization

10.17 PxMbxIdIsValid

NAME

PxMbxIdIsValid() - return TRUE if Mbx holds a valid Mbx handle

SYNOPSIS

```
#include <pxdef.h>

int
PxMbxIdIsValid (PxMbx_t Mbx);
```


PARAMETERS

Mbx a Mbx handle

RETURN VALUES

- TRUE if Mbx is a valid Mbx handle

DESCRIPTION

PxMbxIdsValid returns TRUE if obj holds a valid Mbx handle.

10.18 PxMbxIdResetError

NAME

PxMbxIdResetError() - reset the last error of the object handle Mbx

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMbxIdResetError (PxMbx_t *Mbx);
```

PARAMETERS

Mbx pointer to the Mbx handle

RETURN VALUES

- the last error of the object handle Mbx

DESCRIPTION

PxMbxIdResetError resets the last error of the object handle Mbx

10.19 PxMbxIdSet

NAME

PxMbxIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxMbx_t
PxMbxIdSet (PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Mbx handle

DESCRIPTION

PxMbxIdSet returns an object handle which holds id as the object ID

10.20 PxMcIdError

NAME

PxMcIdError() - return the last error of the object handle Mc

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxMcIdError (PxMc_t Mc);
```

PARAMETERS

Mc a Mc handle

RETURN VALUES

- the last error of the object handle Mc

DESCRIPTION

PxMcIdError returns the last error of the object handle Mc

10.21 PxMcIdGet

NAME

PxMcIdGet() - return the object ID of the object handle Mc

SYNOPSIS

```
#include <pxdef.h>
```

```
PxObjId_t  
PxMcIdGet (PxMc_t Mc);
```

PARAMETERS

Mc a Mc handle

RETURN VALUES

- object ID of the object handle Mc

DESCRIPTION

PxMcIdGet returns the object ID of the object handle Mc

10.22 PxMcIdInvalidate

NAME

PxMcIdInvalidate() - return an invalid Mc handle for initialization

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMc_t  
PxMcIdInvalidate (void);
```

RETURN VALUES

- an invalid Mc handle

DESCRIPTION

PxMcIdInvalidate returns an invalid Mc handle for initialization

10.23 PxMcIdIsValid

NAME

PxMcIdIsValid() - return TRUE if Mc holds a valid Mc handle

SYNOPSIS

```
#include <pxdef.h>

int
PxMcIdIsValid(PxMc_t Mc);
```

PARAMETERS

Mc a Mc handle

RETURN VALUES

- TRUE if Mc is a valid Mc handle

DESCRIPTION

PxMcIdIsValid returns TRUE if obj holds a valid Mc handle.

10.24 PxMcIdResetError

NAME

PxMcIdResetError() - reset the last error of the object handle Mc

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMcIdResetError(PxMc_t *Mc);
```

PARAMETERS

Mc pointer to the Mc handle

RETURN VALUES

- the last error of the object handle Mc

DESCRIPTION

PxMcIdResetError resets the last error of the object handle Mc

10.25 PxMcIdSet

NAME

PxMcIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxMc_t
PxMcIdSet(PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Mc handle

DESCRIPTION

PxMclSet returns an object handle which holds id as the object ID

10.26 PxMsgIdError

NAME

PxMsgIdError() - return the last error of the object handle Msg

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMsgIdError (PxMsg_t Msg);
```

PARAMETERS

Msg a Msg handle

RETURN VALUES

- the last error of the object handle Msg

DESCRIPTION

PxMsgIdError returns the last error of the object handle Msg

10.27 PxMsgIdGet

NAME

PxMsgIdGet() - return the object ID of the object handle Msg

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxMsgIdGet (PxMsg_t Msg);
```

PARAMETERS

Msg a Msg handle

RETURN VALUES

- object ID of the object handle Msg

DESCRIPTION

PxMsgIdGet returns the object ID of the object handle Msg

10.28 PxMsgIdInvalidate

NAME

PxMsgIdInvalidate() - return an invalid Msg handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgIdInvalidate (void);
```

RETURN VALUES

- an invalid Msg handle

DESCRIPTION

PxMsgIdInvalidate returns an invalid Msg handle for initialization

10.29 PxMsgIdIsValid

NAME

PxMsgIdIsValid() - return TRUE if Msg holds a valid Msg handle

SYNOPSIS

```
#include <pxdef.h>

int
PxMsgIdIsValid(PxMsg_t Msg);
```

PARAMETERS

Msg a Msg handle

RETURN VALUES

- TRUE if Msg is a valid Msg handle

DESCRIPTION

PxMsgIdIsValid returns TRUE if obj holds a valid Msg handle.

10.30 PxMsgIdResetError

NAME

PxMsgIdResetError() - reset the last error of the object handle Msg

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMsgIdResetError(PxMsg_t *Msg);
```

PARAMETERS

Msg pointer to the Msg handle

RETURN VALUES

- the last error of the object handle Msg

DESCRIPTION

PxMsgIdResetError resets the last error of the object handle Msg

10.31 PxMsgIdSet

NAME

PxMsgIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxMsg_t
PxMsgIdSet (PxObjId_t id);
```

PARAMETERS

`id` the object id

RETURN VALUES

- Msg handle

DESCRIPTION

`PxMsgIdSet` returns an object handle which holds `id` as the object ID

10.32 PxObjGetName

NAME

`PxObjGetName()` - return the name of an object

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxObjGetName (PxObjId_t objid,
              PxChar_t *buffer,
              PxUInt_t bufsize);
```

PARAMETERS

`objid` the object's id

`buffer` data area to copy the object's name to

`bufsize` size of data area

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_NAME_BUFOVERFLOW - data area is too short to store the objects name

PXERR_OBJ_ILLOBJ - `objid` is not a valid object id

PXERR_PROT_PERMISSION - data area is not writeable for the calling task

SEE ALSO

- [PxObjSetName\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxObjGetName` copies the name of `objid` into the buffer `buffer` of length `bufsize`. The result is zero-terminated.

10.33 PxObjSetName

NAME

PxObjSetName() - assign a name to an object.

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxObjSetName(PxObjId_t objid,
             const PxChar_t *name,
             PxUInt_t namelen);
```

PARAMETERS

<code>objid</code>	the object's id
<code>name</code>	name to assign to the object
<code>namelen</code>	size of given name

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - the requested object is not on the same core

PXERR_NAME_BUFOVERFLOW - the name has been truncated

PXERR_OBJ_ILLOBJ - the passed object handle is not valid

PXERR_PROT_PERMISSION - name is not readable for the calling task

SEE ALSO

- [PxObjGetName\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxObjSetName assigns the name to the object with handle `objid`.

10.34 PxOpoolIdError

NAME

PxOpoolIdError() - return the last error of the object handle `Opool`

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxOpoolIdError(PxOpool_t Opool);
```

PARAMETERS

<code>Opool</code>	a Opool handle
--------------------	----------------

RETURN VALUES

- the last error of the object handle `Opool`

DESCRIPTION

PxOpoolIdError returns the last error of the object handle Opool

10.35 PxOpoolIdGet

NAME

PxOpoolIdGet() - return the object ID of the object handle Opool

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxOpoolIdGet (PxOpool_t Opool);
```

PARAMETERS

Opool a Opool handle

RETURN VALUES

- object ID of the object handle Opool

DESCRIPTION

PxOpoolIdGet returns the object ID of the object handle Opool

10.36 PxOpoolIdInvalidate

NAME

PxOpoolIdInvalidate() - return an invalid Opool handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxOpool_t
PxOpoolIdInvalidate (void);
```

RETURN VALUES

- an invalid Opool handle

DESCRIPTION

PxOpoolIdInvalidate returns an invalid Opool handle for initialization

10.37 PxOpoolIdIsValid

NAME

PxOpoolIdIsValid() - return TRUE if Opool holds a valid Opool handle

SYNOPSIS

```
#include <pxdef.h>

int
PxOpoolIdIsValid (PxOpool_t Opool);
```

PARAMETERS

Opool a Opool handle

RETURN VALUES

- TRUE if Opool is a valid Opool handle

DESCRIPTION

PxOpoolIdIsValid returns TRUE if obj holds a valid Opool handle.

10.38 PxOpoolIdResetError

NAME

PxOpoolIdResetError() - reset the last error of the object handle Opool

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxOpoolIdResetError(PxOpool_t *Opool);
```

PARAMETERS

Opool pointer to the Opool handle

RETURN VALUES

- the last error of the object handle Opool

DESCRIPTION

PxOpoolIdResetError resets the last error of the object handle Opool

10.39 PxOpoolIdSet

NAME

PxOpoolIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxOpool_t
PxOpoolIdSet(PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Opool handle

DESCRIPTION

PxOpoolIdSet returns an object handle which holds id as the object ID

10.40 PxPeldError

NAME

PxPeldError() - return the last error of the object handle Pe

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
PxPeIdError (PxPe_t Pe);
```

PARAMETERS

Pe a Pe handle

RETURN VALUES

- the last error of the object handle **Pe**

DESCRIPTION

PxPeldError returns the last error of the object handle **Pe**

10.41 PxPeldGet

NAME

PxPeldGet() - return the object ID of the object handle **Pe**

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxPeIdGet (PxPe_t Pe);
```

PARAMETERS

Pe a Pe handle

RETURN VALUES

- object ID of the object handle **Pe**

DESCRIPTION

PxPeldGet returns the object ID of the object handle **Pe**

10.42 PxPeldInvalidate

NAME

PxPeldInvalidate() - return an invalid Pe handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxPe_t
PxPeIdInvalidate (void);
```

RETURN VALUES

- an invalid Pe handle

DESCRIPTION

PxPeldInvalidate returns an invalid Pe handle for initialization

10.43 PxPeldIsValid

NAME

PxPeldIsValid() - return TRUE if **Pe** holds a valid Pe handle

SYNOPSIS

```
#include <pxdef.h>

int
PxPeldIsValid(PxPe_t Pe);
```

PARAMETERS

Pe a Pe handle

RETURN VALUES

- TRUE if Pe is a valid Pe handle

DESCRIPTION

PxPeldIsValid returns TRUE if obj holds a valid Pe handle.

10.44 PxPeldResetError

NAME

PxPeldResetError() - reset the last error of the object handle Pe

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxPeldResetError(PxPe_t *Pe);
```

PARAMETERS

Pe pointer to the Pe handle

RETURN VALUES

- the last error of the object handle Pe

DESCRIPTION

PxPeldResetError resets the last error of the object handle Pe

10.45 PxPeldSet

NAME

PxPeldSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxPe_t
PxPeldSet(PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- Pe handle

DESCRIPTION

PxPeldSet returns an object handle which holds id as the object ID

10.46 PxSysObjReleaseAllObjects

NAME

PxSysObjReleaseAllObjects() - release all objects requested by the caller

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxSysObjReleaseAllObjects(void);
```

RETURN VALUES

- release error on failure
- PXERR_NOERROR on success

ERROR CODES

all - errors available from Px...Release

DESCRIPTION

PxSysObjReleaseAllObjects releases all objects which are requested by the calling task. Right now only PxDelay, PxTo, PxPe, PxMbx and PxTask are handled.

10.47 PxTaskIdError

NAME

PxTaskIdError() - return the last error of the object handle Task

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTaskIdError(PxTask_t Task);
```

PARAMETERS

Task a Task handle

RETURN VALUES

- the last error of the object handle Task

DESCRIPTION

PxTaskIdError returns the last error of the object handle Task

10.48 PxTaskIdGet

NAME

PxTaskIdGet() - return the object ID of the object handle Task

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxTaskIdGet(PxTask_t Task);
```

PARAMETERS

Task a Task handle

RETURN VALUES

- object ID of the object handle Task

DESCRIPTION

PxTaskIdGet returns the object ID of the object handle Task

10.49 PxTaskIdInvalidate

NAME

PxTaskIdInvalidate () - return an invalid Task handle for initialization

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTask_t
PxTaskIdInvalidate(void);
```

RETURN VALUES

- an invalid Task handle

DESCRIPTION

PxTaskIdInvalidate returns an invalid Task handle for initialization

10.50 PxTaskIdIsValid

NAME

PxTaskIdIsValid () - return TRUE if Task holds a valid Task handle

SYNOPSIS

```
#include <pxdef.h>
```

```
int
PxTaskIdIsValid(PxTask_t Task);
```

PARAMETERS

Task a Task handle

RETURN VALUES

- TRUE if Task is a valid Task handle

DESCRIPTION

PxTaskIdIsValid returns TRUE if obj holds a valid Task handle.

10.51 PxTaskIdResetError

NAME

PxTaskIdResetError() - reset the last error of the object handle Task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
PxTaskIdResetError(PxTask_t *Task);
```

PARAMETERS

`Task` pointer to the Task handle

RETURN VALUES

- the last error of the object handle Task

DESCRIPTION

PxTaskIdResetError resets the last error of the object handle Task

10.52 PxTaskIdSet

NAME

PxTaskIdSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxTask_t
PxTaskIdSet (PxObjId_t id);
```

PARAMETERS

`id` the object id

RETURN VALUES

- Task handle

DESCRIPTION

PxTaskIdSet returns an object handle which holds id as the object ID

10.53 PxToldError

NAME

PxToldError() - return the last error of the object handle To

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxToIdError (PxTo_t To);
```

PARAMETERS

`To` a To handle

RETURN VALUES

- the last error of the object handle To

DESCRIPTION

PxToldError returns the last error of the object handle To

10.54 PxToldGet

NAME

PxToldGet() - return the object ID of the object handle To

SYNOPSIS

```
#include <pxdef.h>

PxObjId_t
PxToIdGet (PxTo_t To);
```

PARAMETERS

To a To handle

RETURN VALUES

- object ID of the object handle To

DESCRIPTION

PxToldGet returns the object ID of the object handle To

10.55 PxToldInvalidate

NAME

PxToldInvalidate() - return an invalid To handle for initialization

SYNOPSIS

```
#include <pxdef.h>

PxTo_t
PxToIdInvalidate(void);
```

RETURN VALUES

- an invalid To handle

DESCRIPTION

PxToldInvalidate returns an invalid To handle for initialization

10.56 PxToldIsValid

NAME

PxToldIsValid() - return TRUE if To holds a valid To handle

SYNOPSIS

```
#include <pxdef.h>

int
PxToIdIsValid(PxTo_t To);
```

PARAMETERS

To a To handle

RETURN VALUES

- TRUE if To is a valid To handle

DESCRIPTION

PxToldIsValid returns TRUE if obj holds a valid To handle.

10.57 PxToldResetError

NAME

PxToldResetError() - reset the last error of the object handle To

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxToIdResetError(PxTo_t *To);
```

PARAMETERS

To pointer to the To handle

RETURN VALUES

- the last error of the object handle To

DESCRIPTION

PxToldResetError resets the last error of the object handle To

10.58 PxToldSet

NAME

PxToldSet() - return an object handle which holds id as the object ID

SYNOPSIS

```
#include <pxdef.h>

PxTo_t
PxToIdSet(PxObjId_t id);
```

PARAMETERS

id the object id

RETURN VALUES

- To handle

DESCRIPTION

PxToldSet returns an object handle which holds id as the object ID

11 Object Pool

11.1 PxOpoolGetCurrentCapacity

NAME

PxOpoolGetCurrentCapacity() - return the current capacity of an object pool

SYNOPSIS

```
#include <pxdef.h>
```

```
PxUInt_t
```

```
PxOpoolGetCurrentCapacity(PxOpool_t opoolid);
```

PARAMETERS

`opoolid` an object pool id

RETURN VALUES

- the current capacity of the given object pool

ERROR CODES

PXERR_OPOOL_ILLOPOOL - `opoolid` is not a valid object pool id

SEE ALSO

- [PxOpoolRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxOpoolGetCurrentCapacity returns the current capacity in `opool`. This is the actual number of available objects only for real object pools. For virtual objects pools, it describes the number of objects that can be obtained from the object pool, provided that its source has sufficient capacity. If PxOpoolGetCurrentCapacity returns 0, either the object pool is empty or an error has occurred. Call PxGetError to see, if an error occurred.

11.2 PxOpoolGetType

NAME

PxOpoolGetType() - return the object pool type.

SYNOPSIS

```
#include <pxdef.h>
```

```
PxOpoolType_t
```

```
PxOpoolGetType(PxOpool_t opoolid);
```

PARAMETERS

`opoolid` an object pool id

RETURN VALUES

- the object pool type

ERROR CODES

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool id

SEE ALSO

- [PxOpoolRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxOpoolGetType returns the type of the given object pool. The type can be either PXOpoolVirtual or PXOpoolReal or PXOpoolIllegalType if an invalid object pool is given.

11.3 PxOpoolRelease

NAME

PxOpoolRelease() - release an object pool object

SYNOPSIS

```
#include <pxdef.h>

PxOpool_t
PxOpoolRelease(PxOpool_t Opool);
```

PARAMETERS

Opool object pool object to be released

RETURN VALUES

- invalid object pool handle on success
- object pool on failure

ERROR CODES

PXERR_OPOOL_ILLOPOOL - Opool is not a valid object pool object

SEE ALSO

- [PxOpoolRequest\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxOpoolRelease releases the object pool handle Opool by converting it into a generic object and releasing this object.

11.4 PxOpoolRequest

NAME

PxOpoolRequest() - request a object pool

PxOpoolRequest_NoWait() - request a object pool with immediate return

SYNOPSIS

```
#include <pxdef.h>

PxOpool_t
PxOpoolRequest(PxOpoolType_t opooltype,
```

```

        PxUInt_t capacity,
        PxOpool_t src,
        PxOpool_t opoolid);

PxOpool_t
PxOpoolRequest_NoWait (PxOpoolType_t opooltype,
                      PxUInt_t capacity,
                      PxOpool_t src,
                      PxOpool_t opoolid);

```

PARAMETERS

<code>opooltype</code>	what kind of object pool is requested (real or virtual)
<code>capacity</code>	number of objects in the created object pool
<code>src</code>	the object pool, where the objects for the object pool object are requested from.
<code>opoolid</code>	the object pool, where the object pool object is requested from.

RETURN VALUES

- invalid object pool handle on failure
- object pool on success

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - opoolid or src is not a valid object pool

Exceptions of PxOpoolRequest_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- [PxOpoolGetCurrentCapacity\(\)](#)
- [PxOpoolGetType\(\)](#)
- [PxOpoolRelease\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxOpoolRequest... functions create an object pool handle by converting a generic object from object pool opoolid. The handle's identifier is returned

The functions act differently if there is no object available. In such a case PxOpoolRequest_NoWait fails and PxOpoolRequest waits until a free object is available.

11.5 PxOpoolResolveDefault

NAME

PxOpoolResolveDefault() - resolve an object pool default

SYNOPSIS

```
#include <pxdef.h>
```

```
PxOpool_t
```

```
PxOpoolResolveDefault (PxOpool_t opoolid);
```

PARAMETERS

`opoolid` an object pool id

RETURN VALUES

- object pool id

ERROR CODES

PXERR_OPOOL_ILLOPOOL - opoolid is not a valid object pool id

SEE ALSO

- [PxInit\(\)](#)
- [PxTaskCreate\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxOpoolResolveDefault resolves an object pool default; more precisely, it returns opoolid, if opoolid is not PXOpoolSystemdefault or PXOpoolTaskdefault; otherwise it returns the object pool id corresponding to the specified default.

12 TriCore™ Implementation To Access Peripheral Registers

12.1 PxRegisterRead

NAME

PxRegisterRead() - read a value from a SFR register

PxRegisterRead_Hnd() - read a value from a SFR register

SYNOPSIS

```
#include <pxdef.h>
```

```
PxULong_t  
PxRegisterRead(volatile PxULong_t *addr);
```

```
PxULong_t  
PxRegisterRead_Hnd(volatile PxULong_t *addr);
```

PARAMETERS

addr address of the desired register to read

RETURN VALUES

- value of the desired register
- 0 on error. If 0 is returned, the caller has to check the task error code with 'PxGetError()'

ERROR CODES

PXERR_ACCESS_RIGHT - task has not the right to access peripheral registers

PXERR_PROT_ILL_REGION - addr is not within additional protection region

DESCRIPTION

PxRegisterRead returns the content of the peripheral register **addr** if the calling task has read access right to this special function register.

The peripheral register **addr** must be covered in the additional protection region table passed in the element **ts_protect_region** of **PxTaskSpec_t** during **PxTaskCreate**.

PxRegisterRead_Hnd is the corresponding handler function. It should only be used from handlers running in a task's context.

12.2 PxRegisterSetMask

NAME

PxRegisterSetMask() - write all bits of value to a SFR register according to the mask

PxRegisterSetMask_Hnd() - write all bits of value to a SFR register according to the mask

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxRegisterSetMask(volatile PxULong_t *addr,
                  PxULong_t mask,
                  PxULong_t val);

PxError_t
PxRegisterSetMask_Hnd(volatile PxULong_t *addr,
                      PxULong_t mask,
                      PxULong_t val);
```

PARAMETERS

addr address of the desired register to modify

mask mask of bits to modify

val value to write to the register

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - task has not the right to access peripheral registers

PXERR_PROT_ILL_REGION - addr is not within additional protection region

DESCRIPTION

PxRegisterSetMask sets the bits described in mask of the peripheral register addr to val if the calling task has write access right to this special function register.

The peripheral register addr must be covered in the additional protection region table passed in the element ts_protect_region of PxTaskSpec_t during PxTaskCreate.

PxRegisterSetMask_Hnd is the corresponding handler function. It should only be used from handlers running in a task's context.

12.3 PxRegisterWrite

NAME

PxRegisterWrite() - write a value to a SFR register

PxRegisterWrite_Hnd() - write a value to a SFR register

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxRegisterWrite(volatile PxULong_t *addr,
                PxULong_t val);

PxError_t
PxRegisterWrite_Hnd(volatile PxULong_t *addr,
                    PxULong_t val);
```

PARAMETERS

<code>addr</code>	address of the desired register to write
<code>val</code>	value to write to the register

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - task has not the right to access peripheral registers

PXERR_PROT_ILL_REGION - `addr` is not within additional protection region

DESCRIPTION

PxRegisterWrite sets the content of the peripheral register `addr` to `val` if the calling task has write access right to this special function register.

The peripheral register `addr` must be covered in the additional protection region table passed in the element `ts_protect_region` of `PxTaskSpec_t` during `PxTaskCreate`.

PxRegisterWrite_Hnd is the corresponding handler function. It should only be used from handlers running in a task's context.

13 PxInit

13.1 PxInit

NAME

PxInit() - PXROS initialization

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxInit(PxInitSpecsArray_t _initspecs,
       PxUInt_t noOfCores);
```

PARAMETERS

`_initspecs` the array of initialization specifications

`noOfCores` number of cores to initialize

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - number of cores not supported

PXERR_GLOBAL_OBJLIST_INCONSISTENCY - inconsistency between global and local init

PXERR_ILLEGAL_ACCESS - incorrect access permission for _initspecs elements

PXERR_ILL_NULLPOINTER_PARAMETER - invalid system stack specification

PXERR_INIT_ILLALIGN - invalid memory block or size alignment in initialization

PXERR_INIT_ILLMCTYPE - type for PXMcSystemdefault is different from PXMcVar-sized, PXMcVarsizedAdjusted and PXMcVarsizedAligned

PXERR_INIT_NOMEM - not enough memory for initialization

PXERR_INIT_SCHEDULE_FAILED - the scheduling of the init task failed

PXERR_INIT_SEGBOUNDARY - block crosses segment boundary

PXERR_MC_ILLALIGN - incorrectly aligned memory for PXMcSystemdefault

PXERR_MC_ILLSIZE - size for PXMcSystemdefault is too small

PXERR_OBJECT_SHORTAGE - not enough objects given in initstruct

PXERR_PROT_ILL_REGION - illegal protection region definition

PXERR_PROT_PERMISSION - memory protection unit cannot be activated

SEE ALSO

- [PxOpoolResolveDefault\(\)](#)
- [PxSysInfoGetNumberOfObjects\(\)](#)
- [PxTaskCreate\(\)](#)
- [PxTaskSetPrio\(\)](#)
- [_PxInit_Start_Cores\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxlInit initializes a PXROS application system according to the specification array `_initspecs` for given number of cores (`noOfCores`). PxlInit creates `PXMcSystemdefault` and `PXOpoolSystemdefault`, creates generic PXROS objects, puts them into the `PXOpoolSystemdefault` object pool, and creates and activates the initialization task.

One element of the specification array is of type `PxlInitSpec_T`.

This data structure specifies the initial properties of a PXROS application system. Current specification items include:

- a description of the system memory class and first initial block
- a description of the memory for the generic PXROS objects
- the number of PXROS objects and the maximum size of their names
- the number of objects which should be defined for intercore communication
- a specification of the initialization tasks properties

The four initial parameters specify the system memory class properties and the initial block. This block must be large enough to allocate all task control blocks. The size of a task control block is defined by `PXTASK_SIZE`.

The memory block for the PXROS objects must be large enough to hold all generic PXROS objects and their names. The size of an object is defined by `PXOBJ_SIZE`.

The parameter `is_obj_number` specifies the total number of generic PXROS objects (tasks, messages, mailboxes, delay jobs etc.) existing at any given moment. The parameter `is_obj_namlength` determines an additional memory area (following immediately the object memory) which can be used for naming individual objects. PXROS requires a certain amount of objects for internal purposes like e.g.:

- the default system memory class `PXMcSystemdefault`
- the default object pool `PXOpoolSystemdefault`.

During initialization, all PXROS objects are created and stored in the object pool `PXOpoolSystemdefault`. From this object pool, these objects may be requested and placed into other object pools or transformed into special PXROS objects.

The parameter `is_global_obj_number` specifies the amount of available objects for the communication across cores. These objects are stored in the global object pool `PXOpoolGlobalSystemdefault`. If `is_global_obj_number` is 0, all object are treated as global objects.

TriCore™ Memory Protection initialization:

The TriCore™ Memory Protection Unit provides a certain amount (depending on architecture version) of data protection register pairs and code protection register pairs which are defined through a lower and upper bound address and the protection rights. These protection register pairs are divided between system and application.

```
/* the code protection register definition for the PXROS system */
const PxCodeProtectSet_T *is_sys_code;
/* the data protection register definition for the PXROS system */
const PxDataProtectSetInit_T *is_sys_data;
/* the code protection register definition for the application */
const PxCodeProtectSet_T *is_task_code;
```

13.2 PxInitializeBeforePxInit

NAME

`PxInitializeBeforePxInit ()` - call all functions from the `__PxInitializeTable`

SYNOPSIS

```
#include <pxdef.h>

void
PxInitializeBeforePxInit (void);
```

SEE ALSO

- `_PxInitcall ()`

DESCRIPTION

`PxInitializeBeforePxInit` calls all functions entered into the `__PxInitializeTable`. These functions should initialize hardware and special function registers, where supervisor mode is needed. Endinit protection has to be removed if necessary. This function should be called before `PxInit` is called.

13.3 _PxInit_Start_Cores

NAME

`_PxInit_Start_Cores()` - start core according to initialization structure

SYNOPSIS

```
#include <pxdef.h>

PxError_t
_PxInit_Start_Cores (PxUChar_t,
                    PxInitSpec_t);
```

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - unsupported coreId

SEE ALSO

- [PxInit\(\)](#)

DESCRIPTION

`_PxInit_Start_Cores` implements a default function for starting other cores in a multi core environment. This function is called inside of `PxInit`. It can be overridden by the user to customize the application.

13.4 _PxInitcall

NAME

`_PxInitcall()` - define a function which can be called before `PxInit`

SYNOPSIS

```
#include <pxdef.h>

_PxInitcall
(void func,
 parms...);
```

PARAMETERS

<code>func</code>	function to call
<code>parms...</code>	parameters to function

SEE ALSO

- [PxInitializeBeforePxInit\(\)](#)

DESCRIPTION

`_PxInitcall` defines a wrapper function `func___` to call the function `func(...)` with the proper arguments. The wrapper function `func___` will get an entry in `__PxInitializeTable`

14 Special PXROS Services

14.1 PxServiceTaskInit

NAME

PxServiceTaskInit () - instantiate the calling task as PXROS service task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxServiceTaskInit (void);
```

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to act as a service task

PXERR_TASK_DIESRV_INITIALIZED - the dieserver is (seems to be) already initialized

SEE ALSO

- [PxDie\(\)](#)
- [PxDieService\(\)](#)
- [PxMsgrelService](#)
- [PxTerminate\(\)](#)

DESCRIPTION

PxServiceTaskInit instantiates the calling task as the PXROS service task. The PXROS service task is responsible to call PxDieService and PxMsgrelService on request. There are two special events which announce the PXROS service task to call these services. The event PXSERVICE_TASK_DIED is signaled from the syscall PxDie after the calling task has been suspended and is no longer scheduled and ready for deletion. In receiving this event the PXROS service task has to call PxDieService to remove the terminated task from the system. The event PXSERVICE_HND_MSGREL is signaled by the syscall PxMsgRelease_Hnd if a handler releases a message. After receiving this event the PXROS service task has to call PxMsgrelService to release the handler message.

An example for a PXROS service task could be:

```
PxError_t ServiceTaskCode (void)  
{  
    PxMbx_t relmbx;  
    PxEvents_t ev;  
    PxError_t err;  
    relmbx = PxMbxRequest (PXOpoolTaskdefault);  
    /* or
```

```

relmbx = PxTaskGetMbx(PxGetId());
*/
if (PxMbxIdError(relmbx) != PXERR_NOERROR)
return PxMbxIdError(relmbx);

err = PxServiceTaskInit();
if (err != PXERR_NOERROR)
return err;

err = PxMsgrelServiceInit(relmbx);
if (err != PXERR_NOERROR)
return err;

while(1)
{
ev = PxAwaitEvents(PXSERVICE_TASK_DIED | PXSERVICE_HND_MSGREL);
if (ev & PXSERVICE_TASK_DIED)
{
err = PxDieService();
if (err != PXERR_NOERROR)
{
/* do error handling */
}
}
if (ev & PXSERVICE_HND_MSGREL)
{
msg = PxMsgrelService();
if (PxMsgIdError(msg) != PXERR_NOERROR)
{
/* do error handling */
}
}
}
}
}
}

```

15 System Information Functions

15.1 PxDelayInfoGetDelayInfo

NAME

PxDelayInfoGetDelayInfo() - Function to get delay info

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>
```

```
PxError_t
PxDelayInfoGetDelayInfo(PxDelayInfo_t *DelayInfo,
                        PxDelay_t delayId); \details PxDelayInfoGetDelayInfo stores the
```

PARAMETERS

DelayInfo pointer to data area to store delay information to

delayId Id of delay object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_DELAY_ILLDelay - given object is no delay object

PXERR_PROT_PERMISSION - task has no write permission on the DelayInfo object

SEE ALSO

- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

PxDelayInfoGetDelayInfo stores the contents of the delay structure delayId into the info structure DelayInfo. The structure DelayInfo has the following format:

```
typedef struct
{
    PxDelayInfoType_t PxDelayInfo_Type; /* delay type */
    void (*PxDelay_Handler)(PxArg_t); /* delay handler function */
    PxULong_t PxDelay_Param; /* delay handler function's arguments */
    PxULong_t PxDelay_Ticks; /* delay ticks */
    PxULong_t PxDelay_RestTicks; /* delay rest ticks */
    PxTask_t PxDelay_RequestingTask; /* task which requested the delay object */
} PxDelayInfo_t;
```

The delay object delayId may have one of the following types:

```
typedef enum {
    DelayType_InUse,
    DelayType_NotUsed
} PxDelayInfoType_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
PxInfoMC_t      McInfo; /* memory class information struct */
PxInfoOpool_t   OpoolInfo; /* Opool information struct */
PxInfoMsg_t     MsgInfo; /* message information struct */
PxInfoMbx_t     MbxInfo; /* mailbox information struct */
PxInfoDelay_t   DelayInfo; /* delay object information struct */
PxInfoPe_t      PeInfo; /* periodic event information struct */
PxInfoTo_t      ToInfo; /* timeout object information struct */
PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;
```

15.2 PxDynSysInfoGetInterruptInfo

NAME

PxDynSysInfoGetInterruptInfo () - Function to get interrupt info

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxDynSysInfoGetInterruptInfo (PxInfoInterrupt_t *InterruptInfo,
                             PxInterrupt_t interruptId);
```

PARAMETERS

***InterruptInfo** pointer to data area to store interrupt information to

interruptId Id of interrupt object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_INTERRUPT_ILLINTERRUPT - given object is no interrupt object

PXERR_PROT_PERMISSION - task has no write permission on the InterruptInfo object

SEE ALSO

- Interrupt and Trap Services, see [chapter 6](#) on page 19

DESCRIPTION

PxDynSysInfoGetInterruptInfo stores the contents of the interrupt structure interruptId into the info structure InterruptInfo. The structure InterruptInfo has the following format:

```
typedef struct
{
PxUInt_t      PxInfoInterrupt_Number; /* interrupt number */
void          (*PxInfoInterrupt_Handler) (PxArg_t); /* interrupt handler function */
PxULong_t     PxInfoInterrupt_Param; /* interrupt handler function's arguments */
PxTask_t      PxInfoInterrupt_RequestingTask; /* task which requested the interrupt */
} PxInfoInterrupt_t;
```

There is a union available containing all sysinfo types:


```

typedef union {
PxInfoMC_t      McInfo; /* memory class information struct */
PxInfoOpool_t   OpoolInfo; /* Opool information struct */
PxInfoMsg_t     MsgInfo; /* message information struct */
PxInfoMbx_t     MbxInfo; /* mailbox information struct */
PxInfoDelay_t   DelayInfo; /* delay object information struct */
PxInfoPe_t      PeInfo; /* periodic event information struct */
PxInfoTo_t      ToInfo; /* timeout object information struct */
PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;

```

15.3 PxDynInfoGetMCInfo

NAME

PxDynInfoGetMCInfo() - Function to get memory class info

SYNOPSIS

```

#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxDynInfoGetMCInfo (PxInfoMC_t *MCInfo,
                    PxMc_t mcId);

```

PARAMETERS

***MCInfo** pointer to data area to store memory class information to

mcId Id of memory class object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MC_ILLMC - given object is not a memory class object

PXERR_PROT_PERMISSION - task has no write permission on the MCInfo object

PXERR_SERVICE_NOT_CONFIGURED - MONITOR_OBJECTS is not configured

SEE ALSO

- Memory Management, see [chapter 8](#) on page 31

DESCRIPTION

PxDynInfoGetMCInfo stores the memory class type (fix/var), the free memory and the lowest capacity of the memory class mcId into the info structure MCInfo. In fixsized memory classes PxInfoMC_FreeMem and PxInfoMC_MinCapacity represent the number of blocks, in varsized memory classes the number of bytes.

The structure MCInfo has the following format:

```

typedef struct
{
PxInfoMCType_t      PxInfoMC_Type; /* memory class type */
PxUChar_t           *PxInfoMC_FirstBlock; /* first memory block in memory class */
PxULong_t           PxInfoMC_FreeMem; /* free memory in memory class */
}

```

```

PxUlong_t    PxInfoMC_MinCapacity; /* minimal capacity of memory class */
PxTask_t     PxInfoMC_RequestedTask; /* task, which requested the memory class */
} PxInfoMC_t;

```

The memory class mclId may have one of the following types:

```

typedef enum {
MCType_FixSized,
MCType_VarSized
} PxInfoMCType_t;

```

There is a union available containing all sysinfo types:

```

typedef union {
PxInfoMC_t      McInfo; /* memory class information struct */
PxInfoOpool_t   OpoolInfo; /* Opool information struct */
PxInfoMsg_t     MsgInfo; /* message information struct */
PxInfoMbx_t     MbxInfo; /* mailbox information struct */
PxInfoDelay_t   DelayInfo; /* delay object information struct */
PxInfoPe_t      PeInfo; /* periodic event information struct */
PxInfoTo_t      ToInfo; /* timeout object information struct */
PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;

```

15.4 PxSysInfoGetMbxInfo

NAME

PxSysInfoGetMbxInfo() - Function to get mailbox info

SYNOPSIS

```

#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxSysInfoGetMbxInfo (PxInfoMbx_t *MbxInfo,
                    PxMbx_t mbxId);

```

PARAMETERS

***MbxInfo** pointer to data area to store mailbox information to

mbxId Id of mailbox object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MBX_ILLMBX - given object is not a mailbox object

PXERR_PROT_PERMISSION - task has no write permission on the MbxInfo object

SEE ALSO

- Mailbox Services, see [chapter 7](#) on page 27

DESCRIPTION

PxSysInfoGetMbxInfo stores the number of prio/normal messages in the mailbox mbxId and the number of tasks waiting at this mailbox into the info structure MbxInfo.

The structure MbxFInfo has the following format:

```
typedef struct
{
    PxUlong_t PxInfoMbx_NormalMsgs; /* number of normal messages in mailbox */
    PxMsg_t    PxInfoMbx_FirstNormalMsg; /* first normal message in mailbox */
    PxUlong_t PxInfoMbx_PrioMsgs; /* number of prioritized messages in mailbox */
    PxMsg_t    PxInfoMbx_FirstPrioMsg; /* first prioritized message in mailbox */
    PxUlong_t PxInfoMbx_WaitingTasks; /* number of tasks waiting at this mailbox */
    PxTask_t   PxInfoMbx_FirstWaitingTask; /* first waiting task */
    PxTask_t   PxInfoMbx_RequestingTask; /* task, which requested the mailbox */
} PxInfoMbx_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
    PxInfoMC_t    McInfo; /* memory class information struct */
    PxInfoOpool_t OpoolInfo; /* Opool information struct */
    PxInfoMsg_t    MsgInfo; /* message information struct */
    PxInfoMbx_t    MbxFInfo; /* mailbox information struct */
    PxInfoDelay_t   DelayInfo; /* delay object information struct */
    PxInfoPe_t      PeInfo; /* periodic event information struct */
    PxInfoTo_t      ToInfo; /* timeout object information struct */
    PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;
```

15.5 PxBusInfoGetMsgInfo

NAME

PxBusInfoGetMsgInfo() - Function to get info about a PXROS message

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxBusInfoGetMsgInfo(PxInfoMsg_t *MsgInfo,
                    PxMsg_t msgId);
```

PARAMETERS

***MsgInfo** pointer to data area to store message information to

msgId Id of message object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MSG_ILLMSG - given object is not a message object

PXERR_PROT_PERMISSION - task has no write permission on the MsgInfo object

SEE ALSO

- Message-related Services, see [chapter 9](#) on page 39

DESCRIPTION

PxSysInfoGetMsgInfo stores the owner, user, pointer to message data, message size, message buffer size and message type of the message object msgId into the info structure MsgInfo.

The structure MsgInfo has the following format:

```
typedef struct
{
    PxInfoMsgType_t    PxInfoMsg_Type; /* message type */
    PxTask_t          PxInfoMsg_Owner; /* message owner */
    PxTask_t          PxInfoMsg_User; /* message user */
    PxMbx_t           PxInfoMsg_Mbx; /* mailbox, where message lies */
    PxUChar_t         *PxInfoMsg_Data; /* message data area */
    PxULong_t         PxInfoMsg_Size; /* actual size of data area */
    PxULong_t         PxInfoMsg_BufSize; /* size of requested data area */
    PxTask_t          PxInfoMsg_RequestingTask; /* task, which has requested the message */
    PxMbx_t           PxInfoMsg_RelMbx; /* messages release mailbox */
} PxInfoMsg_t;
```

The message msgId may have one of the following types:

```
/*
 * Message type
 */
typedef enum {
    MsgType_Unknown, /* unknown type */
    MsgType_Enveloped, /* requested via PxMsgEnvelop */
    MsgType_Requested /* requested via PxMsgRequest */
} PxInfoMsgType_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
    PxInfoMC_t      McInfo; /* memory class information struct */
    PxInfoOpool_t   OpoolInfo; /* Opool information struct */
    PxInfoMsg_t     MsgInfo; /* message information struct */
    PxInfoMbx_t     MbxInfo; /* mailbox information struct */
    PxInfoDelay_t    DelayInfo; /* delay object information struct */
    PxInfoPe_t      PeInfo; /* periodic event information struct */
    PxInfoTo_t      ToInfo; /* timeout object information struct */
    PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;
```

15.6 PxSysInfoGetMsgsInMbx

NAME

2009

PxSysInfoGetMsgsInMbx() - get the ids of the messages stored in a mailbox

SYNOPSIS

```
#include <pxdef.h>

int
PxSysInfoGetMsgsInMbx (PxMbx_t mbxId,
                      PxMsgType_t Type,
                      PxObjId_t *MsgArray,
                      int Max);
```

PARAMETERS

<code>mbxId</code>	mailbox which messages are counted
<code>Type</code>	type of messages to be counted
<code>*MsgArray</code>	array to store the messages id
<code>Max</code>	maximum number of messages to be counted

RETURN VALUES

- Number of message ids stored in the array
- -1 if `mbxId` is not a valid mailbox or if an inconsistency is detected

SEE ALSO

- Mailbox Services, see [chapter 7](#) on page 27
- Message-related Services, see [chapter 9](#) on page 39

DESCRIPTION

`PxBusInfoGetMsgsInMbx` counts the messages of type `Type` available in the mailbox `mbxId`. The message id's are stored in `MsgArray` until the maximum number of messages, which can be stored in this array, is reached. `Max` represents this number. The function returns the number of messages in the mailbox or -1 if `mbxId` is not a valid mailbox object or if an inconsistency is detected. Such an inconsistency may occur, if messages are received from this mailbox when counting these messages.

15.7 PxBusInfoGetNumberOfObjects

NAME

`PxBusInfoGetNumberOfObjects()` - request the number of PXROS objects

2009

SYNOPSIS

```
#include <pxdef.h>

PxUInt_t
PxBusInfoGetNumberOfObjects(void);
```

RETURN VALUES

- The number of objects in the PXROS system.

SEE ALSO

- `PxInit()`

DESCRIPTION

`PxBusInfoGetNumberOfObjects` returns the number of objects in the PXROS system. This value represents the value given in the system description used in the `PxInit` function.

15.8 PxBusInfoGetObjType

NAME

`PxBusInfoGetObjType()` - request the type of an object

2009

SYNOPSIS

```
#include <pxdef.h>

_PxObjType_t
PxSysInfoGetObjType(PxObj_t oId);
```

PARAMETERS

`oId` object id to be checked

RETURN VALUES

- The type of the object
- `_PXObjIllObject` if old is not a valid PXROS object

DESCRIPTION

`PxSysInfoGetObjType` returns the object type of the given object `old`. If `old` is not a valid PXROS object, `_PxObjIllObject` is returned.

15.9 PxSysInfoGetOpoolInfo

NAME

`PxSysInfoGetOpoolInfo()` - Function to get object pool info

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxSysInfoGetOpoolInfo(PxInfoOpool_t *OpoolInfo,
                      PxOpool_t opoolId);
```

PARAMETERS

`*OpoolInfo` pointer to data area to store object pool information to

`opoolId` Id of object pool object

RETURN VALUES

- PXROS error code

ERROR CODES

`PXERR_OPOOL_ILLOPOOL` - given object is not a mailbox object

`PXERR_PROT_PERMISSION` - task has no write permission on the `OpoolInfo` object

SEE ALSO

- Object Pool, see [chapter 11](#) on page 81

DESCRIPTION

`PxSysInfoGetOpoolInfo` stores the object pool `opoolId`'s type (real/virtual), its capacity, its minimal capacity and - if virtual - the superior object pool into the info structure `OpoolInfo`.

The structure `OpoolInfo` has the following format:

```
/*
 * Object pool info
```

```

*/
typedef struct
{
    PxDynInfoOpoolType_t PxDynInfoOpool_Type; /* opool's type */
    PxDynInfoOpool_t     PxDynInfoOpool_Superior; /* real opool, where virtual opools get their obj
    PxDynInfoULong_t     PxDynInfoOpool_Capacity; /* opool's capacity */
    PxDynInfoULong_t     PxDynInfoOpool_MinCapacity; /* opool's lowest capacity */
    PxDynInfoTask_t      PxDynInfoOpool_RequestedTask; /* task, which requested this opool */
} PxDynInfoOpool_t;

```

The object pool PxDynInfoOpoolId may be one of the following types:

```

typedef enum {
    PxDynInfoOpoolType_Real,
    PxDynInfoOpoolType_Virtual
} PxDynInfoOpoolType_t;

```

There is a union available containing all sysinfo types:

```

typedef union {
    PxDynInfoMC_t      McInfo; /* memory class information struct */
    PxDynInfoOpool_t   OpoolInfo; /* Opool information struct */
    PxDynInfoMsg_t     MsgInfo; /* message information struct */
    PxDynInfoMbx_t     MbxInfo; /* mailbox information struct */
    PxDynInfoDelay_t   DelayInfo; /* delay object information struct */
    PxDynInfoPe_t      PeInfo; /* periodic event information struct */
    PxDynInfoTo_t      ToInfo; /* timeout object information struct */
    PxDynInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxDynInfoObjInfo_T;

```

15.10 PxDynInfoGetPeInfo

NAME

PxDynInfoGetPeInfo() - Function to get Pe info

SYNOPSIS

```

#include <pxdef.h>
#include <file pxinfo.h>

PxDynInfoError_t
PxDynInfoGetPeInfo(PxDynInfoPe_t *PeInfo,
                  PxDynInfoPe_t peId);

```

PARAMETERS

PeInfo pointer to data area to store periodic event object information to

peId Id of periodic event object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PE_ILLPE - given object is no periodic event object

PXERR_PROT_PERMISSION - task has no write permission on the PeInfo object

SEE ALSO

- Time Management, see [chapter 19](#) on page 131

DESCRIPTION

PxSysInfoGetPeInfo stores the contents of the periodic event handler peld into the info structure PeInfo.

The structure PeInfo has the following format:

```
typedef struct
{
    PxTask_t      PxInfoPe_Task; /* task to receive periodic event */
    PxEvents_t    PxInfoPe_Event; /* periodic event */
    PxULong_t     PxInfoPe_Period; /* period */
    PxULong_t     PxInfoPe_RestTicks; /* rest ticks */
    PxTask_t      PxInfoPe_RequestedTask; /* task which requested the pe object */
} PxInfoPe_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
    PxInfoMC_t      McInfo; /* memory class information struct */
    PxInfoOpool_t   OpoolInfo; /* Opool information struct */
    PxInfoMsg_t     MsgInfo; /* message information struct */
    PxInfoMbx_t     MbxInfo; /* mailbox information struct */
    PxInfoDelay_t   DelayInfo; /* delay object information struct */
    PxInfoPe_t      PeInfo; /* periodic event information struct */
    PxInfoTo_t      ToInfo; /* timeout object information struct */
    PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;
```

15.11 PxSysInfoGetTaskInfo

NAME

PxSysInfoGetTaskInfo() - Function to get task info

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>

PxError_t
PxSysInfoGetTaskInfo(PxInfoTask_t *TaskInfo,
                      PxTask_t taskId);
```

PARAMETERS

***TaskInfo** pointer to data area to store task information to

taskId Id of task object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PROT_PERMISSION - task has no write permission on the TaskInfo object

PXERR_TASK_ILLTASK - given object is not a task object

SEE ALSO

- Task Manipulation Services, see [chapter 17](#) on page 111

DESCRIPTION

PxDynInfoGetTaskInfo stores the contents of the task control block task into the info structure TaskInfo. The structure TaskInfo has the following format:

```
typedef
struct PxDynInfoTask_t {
    char        *PxDynInfoTask_Name; /* task's name */
    PxDynUChar_t PxDynInfoTask_Priority; /* task's priority */
    PxDynError_t PxDynInfoTask_Error; /* task's last error code */
    PxDynTmode_t PxDynInfoTask_Mode; /* Bitmask of task's mode */
    PxDynInfoTaskState_t PxDynInfoTask_State; /* task's state */
    PxDynEvents_t PxDynInfoTask_SavedEvents; /* task's saved events */
    PxDynEvents_t PxDynInfoTask_EventMask; /* events, the task is waiting for */
    PxDynMc_t PxDynInfoTask_Mc; /* task's default memory class */
    PxDynOpool_t PxDynInfoTask_Opool; /* task's default object pool */
    PxDynMbx_t PxDynInfoTask_PrivateMailbox; /* task's private mailbox */
    PxDynInt_t PxDynInfoTask_ExtremeStackSize; /* task's extreme stack level */
    PxDynInt_t PxDynInfoTask_CurrentStackSize; /* task's current stack level */
    PxDynInt_t PxDynInfoTask_TotalStackSize; /* task's total stack size */
    PxDynInt_t PxDynInfoTask_AbortStackSize; /* task's abort stack size */
    PxDynTask_t PxDynInfoTask_Creator; /* task's creator task */
    PxDynUInt_t PxDynInfoTask_AccessRights; /* task's access rights */
} PxDynInfoTask_t;
```

The task may be in one of the following states:

```
typedef enum {
    TaskState_Unknown, /* unknown state */
    TaskState_Ready, /* task is ready */
    TaskState_Waiting, /* task is waiting */
    TaskState_Waiting_PxDynAwaitEvents, /* task is waiting for events */
    TaskState_Waiting_PxDynMsgRcv, /* task is waiting for a message */
    TaskState_Waiting_PxDynObjReq, /* task is requesting an object */
    TaskState_Waiting_PxDynMcTakeBlk, /* task is requesting a memory block */
    TaskState_Suspended, /* task is suspended */
    TaskState_Suspended_PxDynAwaitEvents, /* task is suspended while waiting for events */
    TaskState_Suspended_PxDynMsgRcv, /* task is suspended while waiting for a message */
    TaskState_Suspended_PxDynObjReq, /* task is suspended while requesting an object */
    TaskState_Suspended_PxDynMcTakeBlk /* task is suspended while requesting a memory block */
} PxDynInfoTaskState_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
    PxDynInfoMC_t McInfo; /* memory class information struct */
    PxDynInfoOpool_t OpoolInfo; /* Opool information struct */
    PxDynInfoMsg_t MsgInfo; /* message information struct */
    PxDynInfoMbx_t MbxCInfo; /* mailbox information struct */
    PxDynInfoDelay_t DelayInfo; /* delay object information struct */
    PxDynInfoPe_t PeInfo; /* periodic event information struct */
    PxDynInfoTo_t ToInfo; /* timeout object information struct */
    PxDynInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxDynObjInfo_T;
```

15.12 PxDynInfoGetToInfo

NAME

PxDynInfoGetToInfo() - Function to get To info

SYNOPSIS

```
#include <pxdef.h>
#include <file pxinfo.h>
```

```
PxError_t
PxDynInfoGetToInfo (PxInfoTo_t *ToInfo,
                    PxTo_t toId);
```

PARAMETERS

***ToInfo** pointer to data area to store timeout object information to

toId Id of timeout object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PROT_PERMISSION - task has no write permission on the ToInfo object

PXERR_TO_ILLTO - given object is no timeout object

SEE ALSO

- Time Management, see [chapter 19](#) on page [131](#)

DESCRIPTION

PxDynInfoGetToInfo stores the contents of the timeout handler told into the info structure ToInfo.

The structure ToInfo has the following format:

```
typedef struct
{
    PxTask_t      PxInfoTo_Task; /* task to receive timeout event      */
    PxEvents_t     PxInfoTo_Event; /* timeout event                  */
    PxULong_t     PxInfoTo_Timeout; /* timeout period                */
    PxULong_t     PxInfoTo_RestTicks; /* rest ticks                    */
    PxTask_t      PxInfoTo_RequestingTask; /* task which requested the to object */
} PxInfoTo_t;
```

There is a union available containing all sysinfo types:

```
typedef union {
    PxInfoMC_t      McInfo; /* memory class information struct */
    PxInfoOpool_t   OpoolInfo; /* Opool information struct */
    PxInfoMsg_t     MsgInfo; /* message information struct */
    PxInfoMbx_t     MbxInfo; /* mailbox information struct */
    PxInfoDelay_t   DelayInfo; /* delay object information struct */
    PxInfoPe_t      PeInfo; /* periodic event information struct */
    PxInfoTo_t      ToInfo; /* timeout object information struct */
    PxInfoInterrupt_t InterruptInfo; /* interrupt object information struct */
} PxObjInfo_T;
```

16 PXROS Internal System Functions

16.1 PxGetCoreId

NAME

PxGetCoreId() - get the Id of the running core

SYNOPSIS

```
#include <pxdef.h>
```

```
PxCoreId_t  
PxGetCoreId(void);
```

RETURN VALUES

- core Id

DESCRIPTION

PxGetCoreId returns the Id of the running core

16.2 PxGetGlobalServerMbx

NAME

PxGetGlobalServerMbx() - get the requested application-specific server mailbox from an other core

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxGetGlobalServerMbx(PxCoreId_t ServerCore,  
PxMbxReq_t mbxreqid);
```

PARAMETERS

ServerCore the coreId to get the mailbox from

mbxreqid the request service mailbox

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - the requested object pool is not on the same core

PXERR_NAME_ILL_REQUEST - the mbxreqid is not known

PXERR_OPOOL_ILLOPOOL - the passed object pool handle is not valid

PXERR_REQUEST_FAILED - the requested object is not valid

SEE ALSO

- PxMbxRegisterMbx()
- PxMbxRequestMbx()

DESCRIPTION

PxGetGlobalServerMbx requests a system specific mailbox id for a defined service (mbxreqid) from another core (ServerCore).

16.3 PxMbxRegisterMbx

NAME

PxMbxRegisterMbx() - register an application specific server mailbox

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxMbxRegisterMbx (PxMbxReq_t mbxreqid,
                  PxMbx_t mbxid);
```

PARAMETERS

mbxreqid the request id of the server mailbox to register

mbxid of the mailbox to register

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MBX_ILLMBX - the given mailbox is an illegal mailbox object

PXERR_NAME_ILL_REQUEST - the mbxreqid is not known or the task is not allowed to do this call

SEE ALSO

- PxGetGlobalServerMbx()

DESCRIPTION

PxMbxRegisterMbx registers an application-specific mailbox id (mbxid) for a defined system service (mbxreqid).

16.4 PxMbxRequestMbx

NAME

PxMbxRequestMbx() - request an application-specific server mailbox

SYNOPSIS

```
#include <pxdef.h>

PxMbx_t
PxMbxRequestMbx (PxMbxReq_t mbxreqid);
```

PARAMETERS

mbxreqid the request id of the requested server mailbox

RETURN VALUES

- mailbox id of the requested mailbox if available
- invalid mailbox id if mailbox is not available

SEE ALSO

- [PxGetGlobalServerMbx\(\)](#)

DESCRIPTION

PxMbxRequestMbx requests an application-specific mailbox id that has been assigned before for a defined system service (mbxreqid) by PxMbxRegisterMbx.

16.5 PxVersion

NAME

PxVersion() - return PXROS version string

SYNOPSIS

```
#include <pxdef.h>

const
PxChar_t * PxVersion(void);
```

RETURN VALUES

- PXROS version string

DESCRIPTION

PxVersion returns the PXROS version string

16.6 _PxHndcall

NAME

_PxHndcall() - PXROS handler call

SYNOPSIS

```
#include <pxdef.h>

PxArg_t
_PxHndcall(handler,
           task,
           varsize,
           parms...);
```

PARAMETERS

<code>handler</code>	function to be called in supervisor mode
<code>task</code>	calling task (not used on tricore)
<code>varsize</code>	total size of the arguments in bytes (not used on tricore)
<code>parms...</code>	arguments for handler function

DESCRIPTION

_PxHndcall calls the passed function handler with its arguments parms... in supervisor mode. Since the function is executed in the context of the calling task using the task stack, the function can only access data of the task context.

17 Task Manipulation Services

17.1 PxDie

NAME

PxDie() - terminate the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxDie(void);
```

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_MBX_TASKWAITS - there are waiting tasks at the private mailbox

PXERR_TASK_DIESRV_NOT_INITED - dieserver not initialized

SEE ALSO

- [PxServiceTaskInit\(\)](#)
- [PxTaskCreate\(\)](#)
- [PxTerminate\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxDie terminates the calling task, queues it for removing by PxDieService and sends the event PXSERVICE_TASK_DIED to the PXROS service task. The PXROS service task must be initialized before calling PxDie. If no error occurs, the call does not return.

17.2 PxDieService

NAME

PxDieService() - the PXROS die service

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxDieService(void);
```

RETURN VALUES

- PXROS error code

ERROR CODES

all - errors available from Px...Release

SEE ALSO

- [PxServiceTaskInit\(\)](#)
- [PxTerminate\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxDieService removes a task from the system which has called PxDie before. This function has to be called by the PXROS service task after receiving the event PXSERVICE_TASK_DIED.

17.3 PxGetId

NAME

PxGetId() - return the calling task's identifier

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTask_t  
PxGetId(void);
```

RETURN VALUES

- the calling task's id

SEE ALSO

- Application Information Services, see [chapter 2](#) on page 3
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxGetId returns the PXROS object identifier of the calling task.

17.4 PxGetPrivileges

NAME

PxGetPrivileges() - get current privileges of the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxArg_t  
PxGetPrivileges(void);
```

RETURN VALUES

- privileges of the calling task

DESCRIPTION

PxGetPrivileges returns the current privileges of the calling task

17.5 PxGetTimeslices

NAME

PxGetTimeslices() - return timeslices

SYNOPSIS

```
#include <pxdef.h>

PxTicks_t
PxGetTimeslices(void);
```

RETURN VALUES

- reload value for the calling task's timeslice mechanism

SEE ALSO

- PxTaskCreate()
- Application Information Services, see [chapter 2](#) on page 3
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxGetTimeslices returns the reload value for the calling task's timeslice mechanism.

17.6 PxRemoveAccessRights

NAME

PxRemoveAccessRights() - remove access rights

SYNOPSIS

```
#include <pxdef.h>

PxUInt_t
PxRemoveAccessRights(PxUInt_t accessrights);
```

PARAMETERS

`accessrights` the accessrights no longer needed

RETURN VALUES

- the new accessrights

SEE ALSO

- PxTaskCreate()

DESCRIPTION

PxRemoveAccessRights removes the given access rights from the calling task's task control block. The task loses these access rights.

17.7 PxRestoreAccessRights

NAME

PxRestoreAccessRights() - restore access rights

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t
```

```
PxRestoreAccessRights(PxUInt_t accessrights);
```

PARAMETERS

`accessrights` the accessrights to restore

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task does not have the right to restore its access rights

SEE ALSO

- PxTaskCreate()

DESCRIPTION

PxRestoreAccessRights restores the given access rights the task has abandoned earlier. The task must have the access right PXACCESS_TASK_RESTORE_ACCESS_RIGHTS to use this service.

17.8 PxSetPrivileges

NAME

PxSetPrivileges() - set privileges of the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxArg_t
```

```
PxSetPrivileges(PxArg_t privs);
```

PARAMETERS

`privs` new privileges of the calling task.

RETURN VALUES

- privileges of the calling task before the change

DESCRIPTION

PxSetPrivileges gets the privileges specified by `privs`. This service returns the privileges acquired before the change.

17.9 PxSetTimeslices

NAME

PxSetTimeslices() - set timeslices

SYNOPSIS

```
#include <pxdef.h>
```

```
void
```

```
PxSetTimeslices(PxTicks_t timeslices);
```

PARAMETERS

`timeslices` reload value for calling task's timeslice mechanism

SEE ALSO

- [PxTaskCreate\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

`PxSetTimeslices` installs `timeslices` as reload value for the calling task's timeslice mechanism. If `timeslices == 0`, timeslicing is automatically disabled for the calling task.

17.10 PxTaskCheck

NAME

`PxTaskCheck()` - check the validity of a task

SYNOPSIS

```
#include <pxdef.h>

PxBool_t
PxTaskCheck(PxTask_t taskid);
```

PARAMETERS

`taskid` the task object

RETURN VALUES

- true if `taskid` is a task object
- false if `taskid` is not a task object

SEE ALSO

- Application Information Services, see [chapter 2](#) on page 3
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

`PxTaskCheck` checks the validity of a task object. The function returns true if the parameter is a valid task object, else false.

17.11 PxTaskCreate

NAME

`PxTaskCreate()` - create a task object

SYNOPSIS

```
#include <pxdef.h>

PxTask_t
PxTaskCreate(PxOpool_t opool,
             PxTaskSpec_t taskspec,
             PxPrio_t prio,
             PxEvents_t actevents);
```

PARAMETERS

<code>opool</code>	the object pool where to request the task object
<code>taskspec</code>	the task specification
<code>prio</code>	the task's priority
<code>actevents</code>	the event, that activates the task

RETURN VALUES

- task handle on success
- invalid task handle on failure

ERROR CODES

`PXERR_ACCESS_RIGHT` - the new tasks gets more access rights than the calling task or the task does not have the right to create a new task

`PXERR_GLOBAL_ILLEGAL_CORE` - the requested object pool is not on the same core

`PXERR_ILLEGAL_ACCESS` - `taskspec` is not readable for the calling task

`PXERR_MC_ILLMC` - task default memory class is invalid

`PXERR_MC_SEGBOUNDARY` - stack crosses segment boundary

`PXERR_OBJ_ILLOBJ` - unable to get an object for the task's private mailbox

`PXERR_OPOOL_ILLOPOOL` - the passed object pool handle is not valid

`PXERR_TASK_ILLPRIO` - The new task gets a higher priority than the creator's priority without the appropriate access right

`PXERR_TASK_ILLREGION` - The new task has protection regions that are not within its parents protection regions

`PXERR_TASK_ILLSTACKSPECTYPE` - invalid stack type

`PXERR_TASK_SCHEDEXT_NOT_CONFIGURED` - task extensions not configured in this PXROS version

`PXERR_TASK_STACKUNKNOWN` - stack begin could not be determined: specify stack size

`PXERR_TASK_STKMEM` - insufficient memory to allocate the task stack

`PXERR_TASK_TCBMEM` - insufficient memory to allocate the task control block

SEE ALSO

- [PxDie\(\)](#)
- [PxExpectAbort\(\)](#)
- [PxGetAbortFrameSize\(\)](#)
- [PxGetTimeslices\(\)](#)

- [PxInit\(\)](#)
- [PxIntInstallHandler\(\)](#)
- [PxOpoolResolveDefault\(\)](#)
- [PxRemoveAccessRights\(\)](#)
- [PxRestoreAccessRights\(\)](#)
- [PxSetTimeslices\(\)](#)
- [PxTaskGetAccessRights\(\)](#)
- [PxTaskGetMbx\(\)](#)
- [PxTaskGetPrio\(\)](#)
- [PxTaskGetSize\(\)](#)
- [PxTaskSetPrio\(\)](#)
- [PxTerminate\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Event Handling Services, see [chapter 5](#) on page 13
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskCreate creates a PXROS task object according to the specification taskspec and returns its identifier. This new task, identified by its task handle, is scheduled by PXROS until termination.

If not done by the task linker description file, taskspec may specify (among other things) the new task stack and the abort stack. The task stack is used for all local data, procedure calls and calls to PXROS services. The task stack is required at all times. The abort stack is only used with calls to PxExpectAbort. Each nested PxExpectAbort call requires an abort frame.

After task creation all bits in the task mode are cleared. See [chapter 18](#) on page 127 for details.

The task starts when PxTaskSignalEvents... sends one of the events specified in the ts_actevents structure. If 0 was specified, the task starts immediately.

The structure type PxTaskSpec_t:

```
typedef struct {
/* task name —— for debugging */
const PxUChar_t *ts_name;
/* task main function*/
void (*ts_fun) (PxTask_t task, PxMbx_t mbx, PxEvents_t events);
/* the task's PXMctaskdefault */
```

```

PxMc_t   ts_mc;
/* the task's PXOpoolTaskdefault */
PxOpool_t ts_opool;
/* task stack specification */
PxStackSpec_T ts_taskstack;
/* task interrupt stack specification */
PxStackSpec_T ts_inttaskstack;
/* the tasks initial priority */
PxPrio_t   ts_prio;
/* events activating the task, if 0 the task is activated immediately,
otherwise it waits for one of these events */
PxEvents_t ts_actevents;
/* reload value for timeslice account if 0 the default is used
(currently 0,i.e. not participating in the timeslicing mechanism) */
PxTicks_t  ts_timeslices;
/* The abort stack size (given in abort frames).
[Refer to the abortion mechanism description PxExpectAbort (B)]*/
PxSize_t   ts_abortstacksize;
/* privileges for the task; its interpretation is processor dependent; */
PxArg_t     ts_privileges;
/* access rights of the task.
The access rights are described by a mask of allowed access classes */
PxUInt_t    ts_accessrights;
/* In PXROS-HR description of the task address space */
PxTaskContext_ct ts_context;
/* In PXROS-HR a table of memory regions accessible by the task */
PxProtectRegion_ct ts_protect_region;
} PxTaskSpec_T;

```

Description of the main function

```
void (*ts_fun) (PxTask_t task, PxMbx_t mbx, PxEvents_t events)
```

ts_fun must be a C routine which never returns to its caller. The task may only terminate by calling PxDie or PxTerminate, but many application tasks loop forever. ts_fun and all procedures called by ts_fun (including all of their local data) comprise the new task.

The function parameters specify the new task's task handle (task), the mailbox handle mbx of its private mailbox, and the events (events) by which the task gets activated.

There are three different ways to specify a task stack by using the structure type PxStackSpec_t:

1. Using the stack type PXStackAlloc

This type is used for automatic task stack creation. The specification includes the stack size in PxInt_t units (stk_size) and a specification of the memory class from which the stack memory must be taken (stk_src). Stack memory is requested as one block and released automatically as the task terminates.

2. Using the stack type PXStackGrow

This type specifies the stacks lowest address (stk_src) and its size (stk_size).

3. Using the stack type PXStackFall

This type specifies the address above the stack (`stk_src`) and its size in `PxInt_t` units (`stk_size`).

In the last two cases, stack size information may be given as `PXStackDontCheck`. This method does not check for stack overflow. The address provided is taken directly for the stack pointer. If `PXStackDontCheck` is used for size information, the stack type must be compatible with the stack type used by the processor.

```
/* stack specification */
typedef struct
{
    /* specification type */
    PxStackSpecType_t stk_type;
    /* stacksize in "PxInt_t" units */
    PxSize_t stk_size;
    /* stack source: "PxMc_t" for PXStackAlloc, "PxMemAligned_t *" otherwise */
    union
    {
        PxStackAligned_t *stk;
        PxMc_t mc;
    } stk_src;
} PxStackSpec_T;
```

If the task wants to install interrupt handlers, delay handlers, etc, it is necessary that the task has an own interrupt stack. This interrupt stack has to be defined on task creation. This stack is used by each interrupt function, so only one interrupt stack is required. This stack must not be allocated, but has to be defined as `PXStackFall` or `PXStackGrow`, depending on the processor architecture.

The member `ts_privileges` specifies the task's privileges. In PXROS-HR on the TriCore™ processor family there are two allowed privileges for tasks:

- `PXUser0Privilege` - the task will be executed in the User-0 mode.
- `PXUser1Privilege` - the task will be executed in the User-1 mode with access to the periphery.

A task may only create a new task with less or equal privileges. I.e. a task of privilege User-0 cannot create a task in privilege mode User-1

The member `ts_accessrights` specifies the access rights of the task. There exist different access classes. The access right is defined as a bit mask of the allowed access classes.

- `PXACCESS_HANDLERS` - the right to execute `PxHndcalls` and install interrupt handlers with system privileges
- `PXACCESS_INSTALL_HANDLERS` - the right to install interrupt handlers which are executed as PXROS handlers like delay jobs and normal interrupts
- `PXACCESS_INSTALL_SERVICES` - the right to install PXROS services as handlers
- `PXACCESS_REGISTERS` - the right to execute system functions with access to special core registers. These functions are normally processor dependent.

- `PXACCESS_SYSTEMDEFAULT` - the right to allocate from the system default resources `PXMcSystemdefault` and `PXOpoolSystemdefault`
- `PXACCESS_RESOURCES` - the right to access resources which are not owned by the task itself i.e. not `Taskdefault` and not created by the task itself
- `PXACCESS_NEW_RESOURCES` - the right to create new resources, i.e. new objectpools and memory classes
- `PXACCESS_SYSTEM_CONTROL` - the right to execute special system functions which can influence the system behavior (e.g. `PxSetMessagefun`, `PxTaskSuspend`)
- `PXACCESS_MODEBITS` - the right for a task to set its modebits. A task may always clear its modebits.
- `PXACCESS_OVERRIDE_ABORT_EVENTS` - the right to override the aborting events from `PxExpectAbort`; a task can use aborting events itself inside a supervised function.
- `PXACCESS_TASK_CREATE` - the right to create a task
- `PXACCESS_TASK_CREATE_HIGHER_PRIO` - the right to create a task with a higher priority
- `PXACCESS_TASK_SET_HIGHER_PRIO` - the right for a task to set its priority to a higher priority than the one it has been created with
- `PXACCESS_CHANGE_PRIO` - the right for a task to change its priority to a lower priority than the one it has been created with
- `PXACCESS_TASK_RESTORE_ACCESS_RIGHTS` - the right for a task to restore its access rights to those it has been created with
- `PXACCESS_TASK_CREATE_HIGHER_ACCESS` - the right to create a task without respecting memory inheritance rule
- `PXACCESS_TRACECTRL` - the right for a task to use `PxTraceCtrl`
- `PXACCESS_GLOBAL_OBJECTS` - the right to allocate from system default resource `PXOPoolGlobalSystemdefault`

On task creation PXROS will compare the access rights of the created task against the access rights of the creator and report an error if the created task would have more rights than the creator.

17.12 PxTaskForceTermination

NAME

`PxTaskForceTermination()` - terminate a task

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTaskForceTermination(PxTask_t taskId);
```


PARAMETERS

`taskId` Id of the task to terminate

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_TASK_ILLPRIV - the calling task is not the creator

PXERR_TASK_ILLTASK - the taskId is illegal

SEE ALSO

- [PxTerminate\(\)](#)

DESCRIPTION

If the caller is the creator of the task to terminate, the task will be forced to call `PxTerminate` for self destruction. If the task is waiting elsewhere it will be activated to terminate immediately.

17.13 PxTaskGetAccessRights

NAME

`PxTaskGetAccessRights()` - return the access rights of a task.

SYNOPSIS

```
#include <pxdef.h>
```

```
PxUInt_t
PxTaskGetAccessRights(PxTask_t taskId);
```

PARAMETERS

`taskId` task, which access rights are requested

RETURN VALUES

- access rights of given task

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - the requested task is not on the same core

PXERR_TASK_ILLTASK - taskId is not a valid task object

SEE ALSO

- [PxTaskCreate\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxTaskGetAccessRights` returns the access rights of the task `taskId`.

17.14 PxTaskGetMbx

NAME

`PxTaskGetMbx()` - get task's mailbox

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMbx_t
```

```
PxTaskGetMbx(PxTask_t taskid);
```

PARAMETERS

taskid task, which mailbox is asked for

RETURN VALUES

- invalid mailbox handle on failure
- mailbox on success

ERROR CODES

PXERR_TASK_ILLTASK - task is not a valid task object

SEE ALSO

- [PxTaskCreate\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskGetMbx returns the private mailbox of the task **taskid**.

17.15 PxTaskGetPrio

NAME

PxTaskGetPrio() - get task priority

SYNOPSIS

```
#include <pxdef.h>
```

```
PxPrio_t
```

```
PxTaskGetPrio(PxTask_t taskid);
```

PARAMETERS

taskid task, which priority is asked for

RETURN VALUES

- priority of the given task

ERROR CODES

PXERR_TASK_ILLTASK - taskid is not a valid task object

SEE ALSO

- [PxTaskCreate\(\)](#)
- [PxTaskSetPrio\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9

- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskGetPrio returns the current priority of the task taskid.

17.16 PxTaskGetSize

NAME

PxTaskGetSize() - return the size of a task control block

SYNOPSIS

```
#include <pxdef.h>

PxSize_t
PxTaskGetSize(void);
```

RETURN VALUES

- size of a task control block

SEE ALSO

- [PxTaskCreate\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskGetSize returns the size of a task control block.

17.17 PxTaskResume

NAME

PxTaskResume() - remove the scheduling inhibition

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTaskResume(PxTask_t taskid);
```

PARAMETERS

taskid task which scheduling inhibition should be removed

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to resume other tasks

PXERR_GLOBAL_ILLEGAL_CORE - the requested task is not on the same core

PXERR_TASK_ILLRDYFUN - invalid ready function detected

PXERR_TASK_ILLTASK - taskid is not a valid task object

SEE ALSO

- [PxTaskSuspend\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskResume removes the scheduling inhibition for the task.

17.18 PxTaskSetPrio

NAME

PxTaskSetPrio() - set task priority

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTaskSetPrio(PxTask_t task,
              PxPrio_t prio);
```

PARAMETERS

<code>task</code>	task, which priority is changed
<code>prio</code>	the new priority of the task

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - calling task does not have the right to change the prio of the task

PXERR_GLOBAL_ILLEGAL_CORE - the requested task is not on the same core

PXERR_TASK_ILLPRIOR - prio is not a valid PXROS priority

PXERR_TASK_ILLTASK - task is not a valid task object

SEE ALSO

- [PxInit\(\)](#)
- [PxTaskCreate\(\)](#)
- [PxTaskGetPrio\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

This service gives task the priority prio. If task is the calling tasks identifier and prio its current priority, the calling task is placed at the end of the appropriate ready list (i.e. scheduling is enforced).

17.19 PxTaskSuspend

NAME

PxTaskSuspend() - prevent a task from being scheduled

PxTaskSuspend_Hnd() - prevent a task from being scheduled

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTaskSuspend(PxTask_t taskid);

PxError_t
PxTaskSuspend_Hnd(PxTask_t taskid);
```

PARAMETERS

taskid task to be suspended

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to suspend other tasks

PXERR_GLOBAL_ILLEGAL_CORE - the requested task is not on the same core

PXERR_TASK_ILLRDYFUN - invalid ready function detected

PXERR_TASK_ILLTASK - taskid is not a valid task object

SEE ALSO

- [PxTaskResume\(\)](#)
- Application Information Services, see [chapter 2](#) on page 3
- Error Handling Services, see [chapter 4](#) on page 9
- Task Mode Manipulation Services, see [chapter 18](#) on page 127

DESCRIPTION

PxTaskSuspend suspends task, i.e. prevents task from being scheduled. If the task is waiting for a resource, the resource may eventually be delivered to task but task remains in a waiting state until resumed by PxTaskResume.

17.20 PxTerminate

NAME

PxTerminate() - terminate the calling task and release all its allocated objects

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTerminate(PxBool_t release);
```

PARAMETERS

release if TRUE release all allocated objects of this task

RETURN VALUES

- error on failure
- no return on success

ERROR CODES

all - errors available from Px...Release

SEE ALSO

- [PxDie\(\)](#)
- [PxDieService\(\)](#)
- [PxServiceTaskInit\(\)](#)
- [PxTaskCreate\(\)](#)
- [PxTaskForceTermination\(\)](#)
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxTerminate terminates the calling task and releases all its allocated objects by calling PxMsgReleaseAllMsg and PxSysObjReleaseAllObjects. If the release was successful PxTerminate calls PxDie to schedule the task for deletion. If no error occurs the call does not return.

18 Task Mode Manipulation Services

18.1 PxClearModebits

NAME

PxClearModebits() - clear modebits of the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTmode_t  
PxClearModebits(PxTmode_t modebits);
```

PARAMETERS

modebits the new modebits

RETURN VALUES

- the old value of modebits

SEE ALSO

- PxAwaitEvents()
- PxExpectAbort()
- PxGetAbortingEvents()
- PxGetSavedEvents()
- PxResetEvents()
- PxTaskSignalEvents()
- Event Handling Services, see [chapter 5](#) on page 13

DESCRIPTION

PxClearModebits clears the bits modebits in the calling task's task control block. This reenables the corresponding mechanism.

The task mode specifies, how a task reacts on external events. More specifically, the task mode contains the following modebits:

- PXTmodeDisableAborts to control the abort mechanism
- PXTmodeDisableTimeslicing to control the timeslice mechanism

18.2 PxSetModebits

NAME

PxSetModebits() - set modebits of the calling task

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTmode_t
PxSetModebits(PxTmode_t modebits);
```

PARAMETERS

`modebits` the new modebits

RETURN VALUES

- the old value of modebits

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task does not have the right to manipulate its modebits

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxExpectAbort\(\)](#)
- [PxGetAbortingEvents\(\)](#)
- [PxGetSavedEvents\(\)](#)
- [PxResetEvents\(\)](#)
- [PxTaskSignalEvents\(\)](#)
- Event Handling Services, see [chapter 5](#) on page 13

DESCRIPTION

PxSetModebits sets the bits modebits in the calling task's task control block. This disables the corresponding mechanism. The task mode specifies, how a task reacts on external events. More specifically, the task mode contains the following modebits:

- PXTmodeDisableAborts to control the abort mechanism
- PXTmodeDisableTimeslicing to control the timeslice mechanism

18.3 PxTaskGetModebits

NAME

PxTaskGetModebits() - return the modebits of a task.

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTmode_t
PxTaskGetModebits(PxTask_t taskid);
```

PARAMETERS

`taskid` task, which modebits are requested

RETURN VALUES

- modebits of given task

ERROR CODES

PXERR_GLOBAL_ILLEGAL_CORE - the requested task is not on the same core

PXERR_TASK_ILLTASK - taskid is not a valid task object

SEE ALSO

- [PxExpectAbort\(\)](#)
- [PxTaskClearModeBits](#)
- [PxTaskSetModeBits](#)
- Event Handling Services, see [chapter 5](#) on page 13

DESCRIPTION

PxTaskGetModebits returns the modebits from the task's task control block. The task mode specifies, how a task reacts on external events. More specifically, the task mode contains the following modebits:

- PXTmodeDisableAborts to control the abort mechanism
- PXTmodeDisableTimeslicing to control the timeslice mechanism

19 Time Management

19.1 PxPeChange

NAME

PxPeChange() - change the period and events associated with Pe

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxPeChange(PxPe_t Pe,
           PxTicks_t period,
           PxEvents_t events);
```

PARAMETERS

Pe	periodic event object
period	new period
events	new events

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PE_ILLPE - Pe is not a valid periodic event handler object

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the Pe object

SEE ALSO

- PxAwaitEvents()
- PxResetEvents()
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxPeChange changes the period and events associated with Pe. It implicitly stops Pe.

19.2 PxPeRelease

NAME

PxPeRelease() - release the periodic event object

SYNOPSIS

```
#include <pxdef.h>

PxPe_t
PxPeRelease(PxPe_t Pe);
```

PARAMETERS

`Pe` periodic event handler object

RETURN VALUES

- invalid periodic event object on success
- periodic event object on failure

ERROR CODES

`PXERR_PE_ILLPE` - `Pe` is not a valid periodic event handler object

SEE ALSO

- `PxAwaitEvents()`
- `PxPeRequest()`
- `PxResetEvents()`
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxPeRelease` stops the periodic event object `Pe` (if necessary) and releases the associated resources (allocated by `PxPeRequest`). After this call, `Pe` may not be used.

19.3 PxPeRequest

NAME

`PxPeRequest()` - request a periodic event object

`PxPeRequest_EvWait()` - request a periodic event object while waiting for events

`PxPeRequest_NoWait()` - request a periodic event object with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxPe_t
```

```
PxPeRequest(PxOpool_t opoolid,  
            PxTicks_t period,  
            PxEvents_t events);
```

```
PxPe_t
```

```
PxPeRequest_EvWait(PxOpool_t opoolid,  
                   PxTicks_t period,  
                   PxEvents_t events,  
                   PxEvents_t abortevents);
```

```
PxPe_t
```

```
PxPeRequest_NoWait(PxOpool_t opoolid,  
                   PxTicks_t period,  
                   PxEvents_t events);
```

PARAMETERS

`opoolid` the object pool to request the periodic event object from

`period` number of ticks after which the event is signalled

`events` events to be signalled

Parameters of `PxPeRequest_EvWait()`

`abortevents` event mask with events making the call return

RETURN VALUES

- periodic event handler object id

Returnvalues of `PxPeRequest_EvWait()`

- invalid periodic event object on failure
- periodic event object on success
- events, if request aborted by an event

ERROR CODES

`PXERR_ACCESS_RIGHT` - the calling task has not the right to access the object pool

`PXERR_OPOOL_ILLOPOOL` - the passed object pool handle is not valid

Exceptions of `PxPeRequest_EvWait()`

`PXERR_EVENT_ZERO` - the given event mask is zero

`PXERR_OBJ_ABORTED` - request aborted by an event

Exceptions of `PxPeRequest_NoWait()`

`PXERR_OBJ_NOOBJ` - no free object available

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxPeRelease\(\)](#)
- [PxResetEvents\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

`PxPeRequest...` creates and initializes a periodic event object, which is returned. It is associated with `period` `PXROS` ticks and events which are signalled periodically. Signalling is active after the periodic event is started with `PxPeStart`. It is necessary to finish up with `PxPeRelease`, when the periodic event is no longer used. The functions act differently if there is no object available. In such a case `PxPeRequest_NoWait` fails, `PxPeRequest` waits until a free object is available, and `PxPeRequest_EvWait` waits until either there is a free object or an event specified in `abortevents` occurs.

19.4 PxPeStart

NAME

`PxPeStart()` - start periodic events (task service)

`PxPeStart_Hnd()` - start periodic events (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxPeStart (PxPe_t PeId);

PxError_t
PxPeStart_Hnd (PxPe_t PeId);
```

PARAMETERS

PeId periodic event object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PE_ILLPE - Peld is not a valid periodic event object

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the Peld object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxPeStart starts signalling periodic events associated with Pe.

19.5 PxPeStop

NAME

PxPeStop() - stop periodic events (task service)

PxPeStop_Hnd() - stop periodic events (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxPeStop (PxPe_t PeId);

PxError_t
PxPeStop_Hnd (PxPe_t PeId);
```

PARAMETERS

PeId periodic event object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_PE_ILLPE - Peld is not a valid periodic event object

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the Peld object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxPeStop stops signalling periodic events associated with Pe.

19.6 PxTickDefine_Hnd

NAME

PxTickDefine_Hnd() - define a PXROS tick

SYNOPSIS

```
#include <pxdef.h>

void
PxTickDefine_Hnd(void);
```

ERROR CODES

PXERR_TIMESLOTLIST_OVERFLOW - The system could not execute delay jobs due to exceptional interrupt load


SEE ALSO

- [PxTickSetTicksPerSecond\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

Each time, PxTickDefine_Hnd is called, PXROS internal clock is incremented. After interrupt and pending handler processing, PXROS checks whether any delay job is associated with the current time and activates the corresponding delay handlers.

PxTickDefine_Hnd is a handler service. There is no corresponding task service.

 Usually, a real time application is equipped with system clock hardware to provide a clock interrupt at periodic intervals. If so, PxTicksDefine_Hnd is usually called in the corresponding interrupt service routine.

19.7 PxTickGetCount

NAME

PxTickGetCount() - get PXROS tick counter

SYNOPSIS

```
#include <pxdef.h>

PxTicks_t
PxTickGetCount(void);
```

RETURN VALUES

- PXROS tick counter

SEE ALSO

- [PxTickSetTicksPerSecond\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

PXROS maintains a tick counter, incremented by any call to `PxTickDefine_Hnd`. The value of this counter is returned by `PxTickGetCount`.

19.8 PxTickGetTicksFromMilliseconds

NAME

`PxTickGetTicksFromMilliseconds()` - convert time from milliseconds to PXROS ticks

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTicks_t
```

```
PxTickGetTicksFromMilliseconds(PxULong_t millis);
```

PARAMETERS

`millis` milliseconds to convert

RETURN VALUES

- PXROS ticks

SEE ALSO

- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

`PxTickGetTicksFromMilliseconds` converts the time given in milliseconds to the number of PXROS ticks.

19.9 PxTickGetTimeInMilliseconds

NAME

`PxTickGetTimeInMilliseconds()` - return the actual time since the first call to `PxTickDefine_Hnd` in milliseconds

SYNOPSIS

```
#include <pxdef.h>
```

```
PxULong_t
```

```
PxTickGetTimeInMilliseconds(void);
```

RETURN VALUES

- actual time since the first call to `PxTickDefine_Hnd` in milliseconds
- 0, if the ticks per second are not (yet) set

SEE ALSO

- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

PxTickGetTimeInMilliseconds returns the actual time since the first call to PxTickDefine_Hnd in milliseconds.

19.10 PxTickSetTicksPerSecond

NAME

PxTickSetTicksPerSecond() - set the frequency of the internal PXROS tick

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxTickSetTicksPerSecond(PxUInt_t ticksperssecond);
```

PARAMETERS

`ticksperssecond` number of calls to PxTickDefine_Hnd in one second

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to set the frequency

SEE ALSO

- PxDelaySched()
- PxTickDefine_Hnd()
- PxTickGetCount()
- Delay Job Services, see [chapter 3](#) on page 5

DESCRIPTION

PxTickSetTicksPerSecond sets the frequency of the internal PXROS tick. `ticksperssecond` defines the number of calls to PxTickDefine_Hnd in one second.

19.11 PxToChange

NAME

PxToChange() - change the timeout and events associated with To

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxToChange(PxTo_t To,
           PxTicks_t timeout,
           PxEvents_t events);
```

PARAMETERS

<code>To</code>	timeout object
<code>timeout</code>	new timeout

`events` new events

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the To object

PXERR_TO_ILLTO - To is not a valid timeout object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxToChange changes the timeout and events associated with To. It implicitly stops To.

19.12 PxToRelease

NAME

PxToRelease() - release the timeout object

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTo_t
PxToRelease(PxTo_t To);
```

PARAMETERS

To timeout object

RETURN VALUES

- invalid timeout object on success
- timeout object on failure

ERROR CODES

PXERR_TO_ILLTO - To is not a valid timeout object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- [PxToRequest\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxToRelease stops the timeout object To (if necessary) and releases the associated resources (allocated by PxToRequest). After this call, To may not be used.

19.13 PxToRequest

NAME

PxToRequest() - request a timeout object

PxToRequest_EvWait() - request a timeout object while waiting for events

PxToRequest_NoWait() - request a timeout object with immediate return

SYNOPSIS

```
#include <pxdef.h>
```

```
PxTo_t
```

```
PxToRequest (PxOpool_t opoolid,  
             PxTicks_t timeout,  
             PxEvents_t events);
```

```
PxTo_t
```

```
PxToRequest_EvWait (PxOpool_t opoolid,  
                    PxTicks_t timeout,  
                    PxEvents_t events,  
                    PxEvents_t abortevents);
```

```
PxTo_t
```

```
PxToRequest_NoWait (PxOpool_t opoolid,  
                    PxTicks_t timeout,  
                    PxEvents_t events);
```

PARAMETERS

<code>opoolid</code>	the object pool to request the timeout object from
<code>timeout</code>	number of ticks after which the event is signalled
<code>events</code>	events to be signalled after period ticks

Parameters of PxToRequest_EvWait()

<code>abortevents</code>	event mask with events making the call return
--------------------------	---

RETURN VALUES

- timeout object

Returnvalues of PxToRequest_EvWait()

- invalid timeout object on failure
- timeout object on success
- events, if request aborted by an event

ERROR CODES

PXERR_ACCESS_RIGHT - the calling task has not the right to access the object pool

PXERR_OPOOL_ILLOPOOL - the passed object pool handle is not valid

Exceptions of PxToRequest_EvWait()

PXERR_EVENT_ZERO - the given event mask is zero

PXERR_OBJ_ABORTED - request aborted by an event

Exceptions of PxToRequest_NoWait()

PXERR_OBJ_NOOBJ - no free object available

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- [PxToRelease\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxToRequest... creates and initializes a timeout event handler object, which is returned. It is associated with period and events. System resources are allocated, as needed, to signal events to the calling task after a period of ticks has passed since the timeout is started (PxToStart). It is necessary to finish up with PxToRelease, when the timeout event is no longer used. The functions act differently if there is no object available. In such a case PxToRequest_NoWait fails, PxToRequest waits until a free object is available, and PxToRequest_EvWait waits until either there is a free object or an event specified in abortevents occurs.

19.14 PxToStart

NAME

PxToStart() - start timeout (task service)

PxToStart_Hnd() - start timeout (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxToStart (PxTo_t ToId) ;

PxError_t
PxToStart_Hnd (PxTo_t ToId) ;
```

PARAMETERS

ToId timeout object

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the Told object

PXERR_TO_ILLTO - Told is not a valid timeout object

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)

- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxToStart starts the timeout associated with Told.

19.15 PxToStop

NAME

PxToStop() - stop timeout (task service)

PxToStop_Hnd() - stop timeout (handler service)

SYNOPSIS

```
#include <pxdef.h>

PxError_t
PxToStop(PxPe_t ToId);

PxError_t
PxToStop_Hnd(PxPe_t ToId);
```

PARAMETERS

ToId timeout handler object id

RETURN VALUES

- PXROS error code

ERROR CODES

PXERR_REQUEST_ILLEGAL - The caller is not the requester of the Told object

PXERR_TO_ILLTO - Told is not a valid timeout handler object id

SEE ALSO

- [PxAwaitEvents\(\)](#)
- [PxResetEvents\(\)](#)
- Delay Job Services, see [chapter 3](#) on page 5
- Error Handling Services, see [chapter 4](#) on page 9

DESCRIPTION

PxToStop stops the timeout associated with Told.

20 Trace Services

20.1 PxSetTraceFunc

NAME

PxSetTraceFunc() - set a new trace function

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxSetTraceFunc(PxTraceFunc_t PxTraceFunc);
```

PARAMETERS

PxTraceFunc the new PXROS trace function

ERROR CODES

PXERR_ACCESS_RIGHT - calling task does not have the right to set the trace function

DESCRIPTION

PxSetTraceFunc sets the PXROS trace function to PxTraceFunc.

20.2 PxTraceAssignBuffer

NAME

PxTraceAssignBuffer() - set a new trace buffer

SYNOPSIS

```
#include <pxdef.h>
```

```
PxError_t  
PxTraceAssignBuffer(PxAligned_t *trcbuffer,  
                   PxULong_t capacity,  
                   PxULong_t size);
```

PARAMETERS

trcbuffer the new trace buffer

capacity number of trace buffer entries

size the size of a trace buffer entry in Byte

ERROR CODES

PXERR_ACCESS_RIGHT - calling task does not have the right to set the trace buffer

PXERR_PROT_PERMISSION - calling task or kernel has no access to the passed buffer

DESCRIPTION

PxTraceAssignBuffer assigns a circular buffer to the PXROS trace mechanism. The buffer size must be a multiple of the size of a PxTrace entry (actual 24 bytes).

20.3 PxTraceCtrl

NAME

PxTraceCtrl() - set or get trace relevant data

SYNOPSIS

```
#include <pxdef.h>
```

```
PxULong_t
PxTraceCtrl(PxTraceCtrl_t cmd,
            PxArg_t arg);
```

PARAMETERS

cmd command to be executed

arg argument for the command

ERROR CODES

PXERR_ACCESS_RIGHT - calling task does not have the right to control the trace interface

PXERR_TASK_ILLTASK - invalid taskid passed

PXERR_TRACE_ILLCtrl - cmd is illegal or unknown

DESCRIPTION

PxTraceCtrl controls the PXROS trace mechanism. There are commands to set or get trace relevant data. The following commands are available:

- PxTraceSetTraceFunction set the PXROS trace function
- PxTraceStart start PXROS tracing, the original trace state is returned
- PxTraceStop stop PXROS tracing, the original trace state is returned
- PxTraceGetState get the actual PXROS trace state
- PxTraceSetGroupMask set the complete PXROS trace group mask
- PxTraceGetGroupMask get the complete PXROS trace group mask
- PxTraceEnableGroup enables the specified group of services
- PxTraceDisableGroup disables the specified group of services
- PxTraceEnableTask enables the PXROS tracing for the specified task
- PxTraceDisableTask disables the PXROS tracing for the specified task
- PxTraceGetTaskState get the actual trace state of the task (disabled or enabled)

20.4 PxTraceGetBuffer

NAME

PxTraceGetBuffer() - return a message object containing the trace buffer

SYNOPSIS

```
#include <pxdef.h>
```

```
PxMsg_t
```

```
PxTraceGetBuffer(PxOpool_t opoolId);
```

PARAMETERS

`opoolId` objectpool to get the message object from

ERROR CODES

PXERR_OPOOL_ILLOPOOL - illegal opool passed

DESCRIPTION

PxTraceGetBuffer returns a message containing the circular buffer of the PXROS trace mechanism.

21 Access Rights

21.1 Access Rights

NAME

PXACCESS_HANDLERS

DESCRIPTION

The right to execute PxHndcalls and install interrupt handlers with system privileges

FUNCTIONS

PxIntInstallFastHandler, PxTrapInstallHandler

NAME

PXACCESS_INSTALL_HANDLERS

DESCRIPTION

The right to install interrupt handlers which are executed as PXROS handlers like delay jobs and normal interrupts

FUNCTIONS

PxMbxInstallHnd, PxIntInstallFastContextHandler, PxIntInstallHandler

NAME

INSTALL_SERVICES_HANDLERS

DESCRIPTION

The right to install PXROS services as handlers

FUNCTIONS

PxIntInstallService

NAME

PXACCESS_REGISTERS

DESCRIPTION

The right to execute system functions with access to special function registers. These functions are normally processor dependent

FUNCTIONS

PxRegisterRead, PxRegisterSetMask, PxRegisterWrite

NAME

PXACCESS_SYSTEMDEFAULT

DESCRIPTION

The right to allocate from the system default resources PXMcSystemdefault and PXOpoolSystemdefault. Tasks which have the Taskdefaults set to Systemdefaults must have also set this access right

FUNCTIONS

PxTaskCreate, All functions requesting memory from PXMcSystemdefault. All functions requesting objects from PXOpoolSystemdefault.

NAME

PXACCESS_RESOURCES

DESCRIPTION

The right to access resources which are not owned by the task itself i.e. not Taskdefault and not created by the task

FUNCTIONS

All functions requesting memory from a memory class created by another task. All functions requesting objects from an object pool created by another task.

NAME

PXACCESS_NEW_RESOURCES

DESCRIPTION

The right to create new resources, i.e. new objectpools and memory classes

FUNCTIONS

PxMcRequest, PxOpoolRequest

NAME

PXACCESS_SYSTEM_CONTROL

DESCRIPTION

The right to execute special system function which can influence the system behaviour like PxTaskSuspend

FUNCTIONS

PxServiceTaskInit, PxSetMessageFun, PxSetTraceFunc, PxTaskForceTermination, PxTaskResume, PxTaskSetPrio for other tasks, PxTaskSuspend, PxTickSetTicksPerSecond

NAME

PXACCESS_MODEBITS

DESCRIPTION

The right for a task to set its modebits

FUNCTIONS

PxSetModebits

NAME

PXACCESS_OVERRIDE_ABORT_EVENTS

DESCRIPTION

The right to override the aborting events from PxExpectAbort; a task can use aborting events itself inside a supervised function

FUNCTIONS

PxAwaitEvents PxMsgAwaitRel EvWait, All functions awaiting events

NOTE

This access right is only tested if the functions are called in a function called from a PxExpectAbort frame.

NAME

PXACCESS_TASK_CREATE

DESCRIPTION

The right to create a task

FUNCTIONS

PxTaskCreate

NAME

PXACCESS_TASK_CREATE_HIGHER_PRIO

DESCRIPTION

The right to create a task with a higher priority

FUNCTIONS

PxTaskCreate

NAME

PXACCESS_TASK_SET_HIGHER_PRIO

DESCRIPTION

The right for a task to set its priority to a higher priority than the one it has been created with

FUNCTIONS

PxTaskSetPrio

NAME

PXACCESS_TASK_RESTORE_ACCESS_RIGHTS

DESCRIPTION

The right for a task to set its access rights to those it has been created with

FUNCTIONS

PxRestoreAccessRights

NAME

PXACCESS_TASK_CREATE_HIGHER_ACCESS

DESCRIPTION

The right for a task to create a task which has access to memory areas outside of the creators memory areas

FUNCTIONS

PxTaskCreate

NAME

PXACCESS_TRACECTRL

DESCRIPTION

The right for a task to use the function PxTraceCtrl to control and manipulate the trace interface

FUNCTIONS

PxTraceCtrl

NAME

PXACCESS_GLOBAL_OBJECTS

DESCRIPTION

The right to allocate objects from the system default resource PXOpoolGlobalSystemdefault. Tasks which have the Taskdefaults set to Systemdefaults must have also set this

access right

FUNCTIONS

PxTaskSignalEvents

21.2 Functions needing Access Rights

Function	Access Right
PxAwaitEvents	PXACCESS_OVERRIDE_ABORT_EVENTS
PxMsgAwaitRel_EvWait	PXACCESS_OVERRIDE_ABORT_EVENTS
PxMbxInstallHnd	PXACCESS_INSTALL_HANDLERS
PxMcRequest	PXACCESS_NEW_RESOURCES
PxMcRequest_EvWait	PXACCESS_NEW_RESOURCES
PxMcRequest_NoWait	PXACCESS_NEW_RESOURCES
PxOpoolRequest	PXACCESS_NEW_RESOURCES
PxOpoolRequest_NoWait	PXACCESS_NEW_RESOURCES
PxIntlInstallFastContextHandler	PXACCESS_INSTALL_HANDLERS
PxIntlInstallFastHandler	PXACCESS_HANDLERS
PxIntlInstallHandler	PXACCESS_INSTALL_HANDLERS
PxIntlInstallService	PXACCESS_INSTALL_SERVICES
PxRegisterRead	PXACCESS_REGISTERS
PxRegisterSetMask	PXACCESS_REGISTERS
PxRegisterWrite	PXACCESS_REGISTERS
PxRestoreAccessRights	PXACCESS_TASK_RESTORE_ACCESS_RIGHTS
PxServiceTaskInit	PXACCESS_SYSTEM_CONTROL
PxSetMessageFun	PXACCESS_SYSTEM_CONTROL
PxSetModebits	PXACCESS_MODEBITS
PxSetTraceFunc	PXACCESS_SYSTEM_CONTROL
PxTaskCreate	PXACCESS_TASK_CREATE
	PXACCESS_TASK_CREATE_HIGHER_PRIO
	PXACCESS_SYSTEMDEFAULT
	PXACCESS_TASK_CREATE_HIGHER_ACCESS
PxTaskForceTermination	PXACCESS_SYSTEM_CONTROL
PxTaskResume	PXACCESS_SYSTEM_CONTROL
PxTaskSetPrio	PXACCESS_SYSTEM_CONTROL
	PXACCESS_TASK_SET_HIGHER_PRIO
PxTaskSuspend	PXACCESS_SYSTEM_CONTROL
PxTickSetTicksPerSecond	PXACCESS_SYSTEM_CONTROL
PxTrapInstallHandler	PXACCESS_HANDLERS
PxTaskSignalEvents across cores	PXACCESS_GLOBAL_OBJECTS
PxTraceCtrl	PXACCESS_TRACECTRL

PxTraceAssignBuffer	PXACCESS_TRACECTRL
All functions requesting memory	PXACCESS_SYSTEMDEFAULT
	PXACCESS_RESOURCES
All functions requesting an object	PXACCESS_SYSTEMDEFAULT
	PXACCESS_RESOURCES
All functions awaiting events	PXACCESS_OVERRIDE_ABORT_EVENTS

Index

Symbols

_PxHndcall	109
_PxInit_Start_Cores	91
_PxInitcall	92

A

Access Rights	147
---------------------	-----

F

Functions needing Access Rights	150
---------------------------------------	-----

P

PxAbort	9
PxAwaitEvents	13
PxClearModebits	127
PxDelayIdError	59
PxDelayIdGet	59
PxDelayIdInvalidate	59
PxDelayIdIsValid	60
PxDelayIdResetError	60
PxDelayIdSet	60
PxDelayRelease	5
PxDelayRequest	5
PxDelayRequest_EvWait	5
PxDelayRequest_NoWait	5
PxDelaySched	6
PxDelaySched_Hnd	6
PxDie	111
PxDieService	111
PxExpectAbort	14
PxGetAbortFrameSize	1
PxGetAbortingEvents	15
PxGetAppinfo	3
PxGetCoreId	107
PxGetError	9
PxGetGlobalServerMbx	107
PxGetId	112
PxGetObjsize	61
PxGetPrivileges	112
PxGetSavedEvents	16
PxGetTimeslices	113
PxInit	89
PxInitializeBeforePxInit	91
PxInterruptIdError	61
PxInterruptIdGet	62
PxInterruptIdInvalidate	62

PxInterruptIdIsValid	62
PxInterruptIdResetError	63
PxInterruptIdSet	63
PxInterruptRelease	22
PxInterruptRequest	23
PxInterruptRequest_EvWait	23
PxInterruptRequest_NoWait	23
PxIntInitVectab	19
PxIntInstallFastContextHandler	19
PxIntInstallFastHandler	20
PxIntInstallHandler	20
PxIntInstallService	21
PxMbxCheck	27
PxMbxIdError	63
PxMbxIdGet	64
PxMbxIdInvalidate	64
PxMbxIdIsValid	64
PxMbxIdResetError	65
PxMbxIdSet	65
PxMbxInstallHnd	27
PxMbxRegisterMbx	108
PxMbxRelease	29
PxMbxRequest	29
PxMbxRequest_EvWait	29
PxMbxRequest_NoWait	29
PxMbxRequestMbx	108
PxMcGetSize	31
PxMcGetType	31
PxMcGetVarsizedOverhead	32
PxMcIdError	65
PxMcIdGet	66
PxMcIdInvalidate	66
PxMcIdIsValid	67
PxMcIdResetError	67
PxMcIdSet	67
PxMcInsertBlk	33
PxMcRelease	34
PxMcRequest	34
PxMcRequest_EvWait	34
PxMcRequest_NoWait	34
PxMcResolveDefault	36
PxMcReturnAllBlks	36
PxMcReturnBlk	37
PxMcTakeBlk	37
PxMessage	9

PxMessageFunDefault	10	PxOpoolIdInvalidate	72
PxMsgAwaitRel	39	PxOpoolIdIsValid	72
PxMsgAwaitRel_EvWait	39	PxOpoolIdResetError	73
PxMsgAwaitRel_NoWait	39	PxOpoolIdSet	73
PxMsgEnvelop	40	PxOpoolRelease	82
PxMsgEnvelop_EvWait	40	PxOpoolRequest	82
PxMsgEnvelop_NoWait	40	PxOpoolRequest_NoWait	82
PxMsgForceRelease	41	PxOpoolResolveDefault	83
PxMsgGetBufferSize	42	PxPanic	11
PxMsgGetData	42	PxPeChange	131
PxMsgGetData_Hnd	42	PxPeldError	73
PxMsgGetMetadata	43	PxPeldGet	74
PxMsgGetMetadata_Hnd	43	PxPeldInvalidate	74
PxMsgGetOwner	44	PxPeldIsValid	74
PxMsgGetProtection	45	PxPeldResetError	75
PxMsgGetSender	45	PxPeldSet	75
PxMsgGetSize	46	PxPeRelease	131
PxMsgIdError	68	PxPeRequest	132
PxMsgIdGet	68	PxPeRequest_EvWait	132
PxMsgIdInvalidate	68	PxPeRequest_NoWait	132
PxMsgIdIsValid	69	PxPeStart	133
PxMsgIdResetError	69	PxPeStart_Hnd	133
PxMsgIdSet	69	PxPeStop	134
PxMsgInstallRelmbx	47	PxPeStop_Hnd	134
PxMsgReceive	47	PxRegisterRead	85
PxMsgReceive_EvWait	47	PxRegisterRead_Hnd	85
PxMsgReceive_NoWait	47	PxRegisterSetMask	85
PxMsgRelDataAccess	48	PxRegisterSetMask_Hnd	85
PxMsgRelease	49	PxRegisterWrite	86
PxMsgRelease_Hnd	49	PxRegisterWrite_Hnd	86
PxMsgReleaseAllMsg	50	PxRemoveAccessRights	113
PxMsgRequest	51	PxResetEvents	16
PxMsgRequest_EvWait	51	PxRestoreAccessRights	113
PxMsgRequest_NoWait	51	PxServiceTaskInit	93
PxMsgSend	53	PxSetAppinfo	3
PxMsgSend_Hnd	53	PxSetError	11
PxMsgSend_Prio	53	PxSetMessageFun	11
PxMsgSend_PrioHnd	53	PxSetModebits	127
PxMsgSetData	54	PxSetPrivileges	114
PxMsgSetMetadata	55	PxSetTimeslices	114
PxMsgSetMetadata_Hnd	55	PxSetTraceFunc	143
PxMsgSetProtection	55	PxSysInfoGetDelayInfo	95
PxMsgSetSize	56	PxSysInfoGetInterruptInfo	96
PxMsgSetToAwaitRel	57	PxSysInfoGetMbxInfo	98
PxObjGetName	70	PxSysInfoGetMCInfo	97
PxObjSetName	71	PxSysInfoGetMsgInfo	99
PxOpoolGetCurrentCapacity	81	PxSysInfoGetMsgsInMbx	100
PxOpoolGetType	81	PxSysInfoGetNumberOfObjects	101
PxOpoolIdError	71	PxSysInfoGetObjType	101
PxOpoolIdGet	72	PxSysInfoGetOpoolInfo	102

PxSysInfoGetPeInfo	103	PxTrapInstallHandler	25
PxSysInfoGetTaskInfo	104	PxVersion	109
PxSysInfoGetTolInfo	106		
PxSysObjReleaseAllObjects	76		
PxTaskCheck	115		
PxTaskCreate	115		
PxTaskForceTermination	120		
PxTaskGetAccessRights	121		
PxTaskGetMbx	121		
PxTaskGetModebits	128		
PxTaskGetPrio	122		
PxTaskGetSize	123		
PxTaskIdError	76		
PxTaskIdGet	76		
PxTaskIdInvalidate	77		
PxTaskIdIsValid	77		
PxTaskIdResetError	77		
PxTaskIdSet	78		
PxTaskResume	123		
PxTaskSetPrio	124		
PxTaskSignalEvents	17		
PxTaskSignalEvents_Hnd	17		
PxTaskSuspend	125		
PxTaskSuspend_Hnd	125		
PxTerminate	125		
PxTickDefine_Hnd	135		
PxTickGetCount	135		
PxTickGetTicksFromMilliseconds	136		
PxTickGetTimeInMilliseconds	136		
PxTickSetTicksPerSecond	137		
PxToChange	137		
PxToldError	78		
PxToldGet	78		
PxToldInvalidate	79		
PxToldIsValid	79		
PxToldResetError	80		
PxToldSet	80		
PxToRelease	138		
PxToRequest	139		
PxToRequest_EvWait	139		
PxToRequest_NoWait	139		
PxToStart	140		
PxToStart_Hnd	140		
PxToStop	141		
PxToStop_Hnd	141		
PxTraceAssignBuffer	143		
PxTraceCtrl	144		
PxTraceGetBuffer	144		
PxTrapAbort	24		
PxTrapInitVectab	24		