PXROS-HR Kernel v8.2.0

Code of Practice

Table of Contents

1. Introduction

This document describes the correct usage of PXROS-HR services in safety-related applications.

Before calling and after return of a PXROS-HR function several tests have to be done in order to check for a successful call and ensure a correct program continuation. These tests comprise the following:

- Validation of the program code by visual check (V)
- Runtime test by comparison of the return value (C)
- Runtime test by function call (F)

2. Initialization Functions

2.1. PxInitializeBeforeInit

Function

`void PxInitializeBeforePxInit(void)`

Before call

This function has to be called before `PxInit()`. (V)

After call

No checks necessary.

Code of practice

No restrictions

2.2. _PxInitcall

Function

`_PxInitcall (void func, parms…)`

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions

2.3. PxInit

Function

`PxError_t PxInit(PxInitSpecsArray_t _initspecs, PxUInt_t noOfCores)`

Before call

`_initspecs` has to be a valid PXROS-HR initialization structure. (V)

`noOfCores` must not exceed the number of available cores. (V)

After call

On success `PxInit` never returns. If the call fails, the reason is given as return value of type `PxError_t`. (C)

Code of practice

`PxInit` initializes PXROS-HR and must be called once only.

3. Application Information Functions

3.1. PxSetAppinfo

Function

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

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

3.2. PxGetAppinfo

Function

```
PxArg_t PxGetAppinfo(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

4. Delay Job Functions

4.1. PxDelayRequest

Function

`PxDelay_t PxDelayRequest(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxDelay_t`. This id may be checked with one of the following macros:

- `PxDelayIdIsValid()` must be true.
- `PxDelayIdGet()` must not be `_PXIllegalObjId`.
- `PxDelayIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxDelayRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxDelayRequest_NoWait()` should be used instead.

`PxDelayRequest` should only be called during initialization to ensure the availability of the delay object.

4.2. PxDelayRequest_NoWait

Function

`PxDelay_t PxDelayRequest_NoWait(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxDelay_t`. This id may be checked with one of the following macros:

- `PxDelayIdIsValid()` must be true.
- `PxDelayIdGet()` must not be `_PXIllegalObjId`.
- `PxDelayIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxDelayRequest_NoWait` should only be called during initialization to ensure the availability of the delay object.

4.3. PxDelayRequest_EvWait

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxDelayEvent_t`. The received events are stored in the `events` part, the delay id is given in the `delay` part of the structure. This id may be checked with one of the following macros:

- `PxDelayIdIsValid()` must be true.
- `PxDelayIdGet()` must not be `_PXIllegalObjId`.
- `PxDelayIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxDelayRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxDelayRequest_NoWait()` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxDelayRequest_EvWait` should only be called during initialization to ensure the availability of

the delay object.

4.4. PxDelayRelease

Function

`PxDelay_t PxDelayRelease(PxDelay_t Delay)`

Before call

`Delay` must be a valid PXROS-HR delay object created with a `PxDelayRequest` call (V). The validity of `Delay` may also be checked by the `PxDelayIsValid` macro (F). The delay object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxDelayCoreId` and the own core id with `PxGetCoreId` (C).

After call

`PxDelayRelease` returns the delay object to the object pool it has been taken from. On success `PxDelayRelease` returns the invalidated delay object. This may be checked with one of the following macros:

- `PxDelayIdIsValid()` must be false.
- `PxDelayIdGet()` must be `_PXIllegalObjId`.
- `PxDelayIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxDelayRelease`, the given delay object `Delay` is no longer valid and may never be used as delay object!

4.5. PxDelaySched

Function

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

Before call

`delayId` must be a valid PXROS-HR delay object created with a `PxDelayRequest` call (V). The validity of `delayId` may also be checked by the `PxDelayIsValid` macro (F). The delay object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxDelayCoreId` and the own core id with `PxGetCoreId` (C).

If `ticks` is zero, the delay job identified by `delayId` is stopped. (C)

The parameter `handler` must be a pointer to a valid function (V).

After call

The function returns `PXERR_NOERROR` if the delay job could be scheduled. Any other return value

describes an error, which has to be interpreted (C).

Code of practice

This function may be called from tasks only. (V)

4.6. PxDelaySched_Hnd

Function

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

Before call

`delayId` must be a valid PXROS-HR delay object created with a `PxDelayRequest` call (V). The validity of `delayId` may also be checked by the `PxDelayIsValid` macro (F). The delay object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxDelayCoreId` and the own core id with `PxGetCoreId` (C).

If `ticks` is zero, the delay job identified by `delayId` is stopped. (C)

The parameter `handler` must be a pointer to a valid function (V).

After call

The function returns `PXERR_NOERROR` if the delay job could be scheduled. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

5. Error Functions

5.1. PxSetError

Function

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

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

5.2. PxGetError

Function

```
PxError_t PxGetError(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

5.3. PxSetMessageFun

Function

```
PxError_t PxSetMessageFun(PxMessageFun_t messagefun)
```

Before call

The parameter must be a pointer to a valid errorfunction (V).

After call

The function returns `PXERR_NOERROR` if the error reporting function could be set. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

No restrictions.

5.4. PxMessage

Function

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

Before call

The PXROS-HR error class given in `cls` must be of type `PxMessageClass_t` and may have one of the following values (V):

- `PXWarning`
- `PXError`
- `PXLogError`
- `PXFatal`

The PXROS-HR error number `err` must be greater or equal `PXERR_NOERROR` and lower than `PXERR_LAST_ERRNO` (V)

After call

No checks necessary.

Code of practice

If no application specific error function is installed, the default error function `PxMessageFunDefault` will call `PxPanic()`, if `cls` is greater than `PXLogError`.

5.5. PxMessageFunDefault

Function

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

Before call

The PXROS-HR error class given in `cls` must be of type `PxMessageClass_t` and may have one of the following values (V):

- `PXWarning`
- `PXError`
- `PXLogError`
- `PXFatal`

The PXROS-HR error number `err` must be greater or equal `PXERR_NOERROR` and lower than `PXERR_LAST_ERRNO` (V)

After call

No checks necessary.

Code of practice

No restrictions.

5.6. PxPanic

Function

```
void PxPanic(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

5.7. PxAbort

Function

```
void PxAbort(PxError_t error)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

6. Event Handling Functions

6.1. PxAwaitEvents

Function

`PxEvents_t PxAwaitEvents(PxEvents_t events)`

Before call

The parameter `events` contains a bitmask of events awaited and should not be zero, as this will force `PxAwaitEvents` to wait forever (V). Typically the event mask is a constant (V).

After call

All events returned should be evaluated.

Code of practice

No restrictions.

6.2. PxResetEvents

Function

`PxEvents_t PxResetEvents(PxEvents_t events)`

Before call

No checks necessary.

After call

All events returned should be evaluated.

Code of practice

No restrictions.

6.3. PxTaskSignalEvents

Function

`PxError_t PxTaskSignalEvents(PxTask_t taskid, PxEvents_t events)`

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns `PXERR_NOERROR` if the event could be signaled to the task. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

This function may be called from tasks only. (V)

6.4. PxTaskSignalEvents_Hnd

Function

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

Before call

The parameter `taskid` must be a valid task object.

After call

The function returns `PXERR_NOERROR` if the event could be signaled to the task. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

This function should be called from handlers only. (V)

6.5. PxGetSavedEvents

Function

```
PxEvents_t PxGetSavedEvents(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

6.6. PxGetAbortingEvents

Function

```
PxEvents_t PxGetAbortingEvents(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

6.7. PxExpectAbort

Function

`PxEvents_t PxExpectAbort(PxEvents_t ev, void func, parms…)`

Before call

The parameter `ev` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

The parameter `func` must be a pointer to a valid function. (V)

After call

No checks necessary.

Code of practice

No restrictions.

6.8. PxGetAbortFrameSize

Function

`PxSize_t PxGetAbortFrameSize(void)`

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

7. Mailbox Functions

7.1. PxMbxRequest

Function

`PxMbx_t PxMbxRequest(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMbx_t`. This id may be checked with one of the following macros:

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMbxRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxMbxRequest_NoWait` should be used instead.

`PxMbxRequest` should only be called during initialization to ensure the availability of the mailbox object.

7.2. PxMbxRequest_NoWait

Function

`PxMbx_t PxMbxRequest_NoWait(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMbx_t`. This id may be checked with one of the following macros:

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMbxRequest_NoWait` should only be called during initialization to ensure the availability of the mailbox object.

7.3. PxMbxRequest_EvWait

Function

```
PxMbxEvent_t PxMbxRequest_EvWait(PxOpool_t opoolid, PxEvents_t events)
```

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxMbxEvent_t`. The received events are stored in the `events` part, the mailbox id is given in the `mbx` part of the structure. This id may be checked with one of the following macros:

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMbxRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxMbxRequest_NoWait` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxMbxRequest_EvWait` should only be called during initialization to ensure the availability of the

mailbox object.

7.4. PxMbxRelease

Function

`PxMbx_t PxMbxRelease(PxMbx_t Mbx)`

Before call

`Mbx` must be a valid PXROS-HR mailbox object created with a `PxMbxRequest` call (V). The validity of `Mbx` may also be checked by the `PxMbxIsValid` macro (F).

After call

`PxMbxRelease` returns the mailbox object to the object pool it has been taken from. On success `PxMbxRelease` returns the invalidated mailbox object. This may be checked with one of the following macros:

- `PxMbxIsValid()` must be false.
- `PxMbxGet()` must be `_PXIllegalObjId`.
- `PxMbxError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxMbxRelease`, the given mailbox object `Mbx` is no longer valid and may never be used as mailbox object!

7.5. PxMbxInstallHnd

Function

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

Before call

`Mbx` must be a valid PXROS-HR mailbox object created with a `PxMbxRequest` call or the task's private mailbox (V). The validity of `Mbx` may also be checked by the `PxMbxIsValid` macro (F).

`hnd` has to be a pointer to a valid functione (V).

The PXROS-HR message type given in `mode` must be of type `PxMsgType_t` and may have one of the following values (V):

- `PXMsgAnyMsg`
- `PXMsgNormalMsg`
- `PXMsgPrioMsg`

After call

The function returns `PXERR_NOERROR` if the mailbox handler has been installed. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

`PxMbxInstallHnd` should only be called during initialization to ensure the availability of the mailbox handler.

7.6. PxMbxCheck

Function

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

Before call

`mbxid` must be a valid PXROS-HR mailbox object created with a `PxMbxRequest` call or the task's private mailbox (V). The validity of `mbxid` may also be checked by the `PxMbxIsValid` macro (F).

After call

No checks necessary.

Code of practice

No restrictions.

8. Memory Class Functions

8.1. PxMcRequest

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The memory class type given in `mctype` must be of type `PxMcType_t` and may have one of the following values (V):

- `PXMcFixsized`
- `PXMcVarsized`
- `PXMcVarsizedAligned`
- `PXMcVarsizedAdjusted`

The `size` parameter must have a plausible value (V).
The calling task must have the access right to create new ressources (V).

After call

The returned value is the id of type `PxMc_t`. This id may be checked with one of the following macros:

- `PxMcIdIsValid()` must be true.
- `PxMcIdGet()` must not be `_PXIllegalObjId`.
- `PxMcIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMcRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxMcRequest_NoWait` should be used instead.

`PxMcRequest` should only be called during initialization to ensure the availability of the memory class.

8.2. PxMcRequest_NoWait

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The memory class type given in `mctype` must be of type `PxMcType_t` and may have one of the following values (V):

- `PXMcFixsized`
- `PXMcVarsized`
- `PXMcVarsizedAligned`
- `PXMcVarsizedAdjusted`

The `size` parameter must have a plausible value (V).
The calling task must have the access right to create new ressources (V).

After call

The returned value is the id of type `PxMc_t`. This id may be checked with one of the following macros:

- `PxMcIdIsValid()` must be true.
- `PxMcIdGet()` must not be `_PXIllegalObjId`.
- `PxMcIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMcRequest_NoWait` should only be called during initialization to ensure the availability of the memory class.

8.3. PxMcRequest_EvWait

Function

`PxMcEvent_t PxMcRequest_EvWait(PxMcType_t mctype, PxSize_t size, PxOpool_t opoolid, PxEvents_t events)`

Before call

The memory class type given in `mctype` must be of type `PxMcType_t` and may have one of the

following values (V):

- `PXMcFixsized`
- `PXMcVarsized`
- `PXMcVarsizedAligned`
- `PXMcVarsizedAdjusted`

The `size` parameter must have a plausible value (V).
`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.
The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).
The calling task must have the access right to create new ressources (V).

After call

The returned value is a structure of type `PxMcEvent_t`. The received events are stored in the `events` part, the memory class id is given in the `mc` part of the structure. This id may be checked with one of the following macros:

- `PxMcIdIsValid()` must be true.
- `PxMcIdGet()` must not be `_PXIllegalObjId`.
- `PxMcIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMcRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxMcRequest_NoWait` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxMcRequest_EvWait` should only be called during initialization to ensure the availability of the memory class.

8.4. PxMcRelease

Function

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

Before call

`Mc` must be a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `Mc` may also be checked by the `PxMcIsValid` macro (F). The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and

the own core id with `PxGetCoreId` (C).

After call

`PxMcRelease` returns the memory class object to the object pool it has been taken from. On success `PxMcRelease` returns the invalidated delay object. This may be checked with one of the following macros:

- `PxMcIdIsValid()` must be false.
- `PxMcIdGet()` must be `_PXIllegalObjId`.
- `PxMcIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxMcRelease`, the given memory class object `Mc` is no longer valid and may never be used as memory class object!

8.5. PxMcGetSize

Function

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

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

8.6. PxMcGetType

Function

```
PxMcType_t PxMcGetType(PxMc_t mcid)
```

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may

also be checked by the `PxMcIsValid` macro (F).

- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

After call

The function returns `PXMcTypeLast` if the given memory class is invalid.

Code of practice

No restrictions.

8.7. PxMcGetVarsizedOverhead

Function

```
PxSize_t PxMcGetVarsizedOverhead(PxMc_t mcid)
```

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

After call

The function returns 0xFFFFFFFF (-1U) if the given memory class is invalid. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions.

8.8. PxMcResolveDefault

Function

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

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

After call

The returned value is the id of type `PxMc_t`. This id may be checked with one of the following macros:

- `PxMcIdIsValid()` must be true.
- `PxMcIdGet()` must not be `_PXIllegalObjId`.
- `PxMcIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

8.9. PxMcTakeBlk

Function

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

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specfying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). `size` must be a plausible value given as a constant (V) or a variable (C).

After call

`PxMcTakeBlk` returns a null pointer on failure (C).

Code of practice

`PxMcTakeBlk` should only be called during initialization to ensure the availability of the memory block.

8.10. PxMcReturnBlk

Function

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

Before call

`mcid` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcid` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). `blk` must be the start of a memory block taken by `PxMcTakeBlk` from the memory class `mcid`.

After call

The function returns `PXERR_NOERROR` if the memory block has been returned to the memory class. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

After `PxMcReturnBlk`, the given memory block `blk` is no longer valid and may never be used as a memory block!

8.11. PxMcInsertBlk

Function

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

Before call

`mcId` must be

- a valid PXROS-HR memory class created with a `PxMcRequest` call (V). The validity of `mcId` may also be checked by the `PxMcIsValid` macro (F).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). `blk` must be a valid memory pointer. The memory must be an area, which has never been part of a memory class or it has to be removed from a memory class with `PxMcRemoveBlk`.
`size` must be a plausible value given as a constant (V) or as a result of a `PxMcRemoveBlk` call (parameter `*Size` (V).

After call

The function returns `PXERR_NOERROR` if the memory block has been inserted into the memory class. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

`PxMcInsertBlk` should only be called during initialization to ensure the availability of the memory in a memory class.

9. Message Functions

9.1. PxMsgRequest

Function

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

Before call

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`mcId` must be a valid PXROS-HR memory class and the calling task must have the access right to take memory from this memory class (V). The validity of `mcId` may also be checked by the `PxMcIsValid` macro (F). The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default memory class `PXMcTaskdefault` is used for this purpose.

`opoolId` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolId` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMsgRequest` may block, if no PXROS-HR object or memory is available. If blocking calls are prohibited, `PxMsgRequest_NoWait()` should be used instead.

9.2. PxMsgRequest_NoWait

Function

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

Before call

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`mcId` must be a valid PXROS-HR memory class and the calling task must have the access right to take memory from this memory class (V). The validity of `mcId` may also be checked by the `PxMcIsValid` macro (F). The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default memory class `PXMcTaskdefault` is used for this purpose.

`opoolId` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolId` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

9.3. PxMsgRequest_EvWait

Function

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

Before call

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`mcId` must be a valid PXROS-HR memory class and the calling task must have the access right to take memory from this memory class (V). The validity of `mcId` may also be checked by the `PxMcIsValid` macro (F). The memory class must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMcCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default memory class `PXMcTaskdefault` is used for this purpose.

`opoolId` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolId` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs

on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxMsgEvent_t`. The received events are stored in the `events` part, the message id is given in the `msg` part of the structure. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMsgRequest_EvWait` may block, if no PXROS-HR object or memory is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxMsgRequest_NoWait()` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

9.4. PxMsgEnvelop

Function

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

Before call

`data_area` must be a pointer to a valid data area.

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.

- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMsgEnvelop` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxMsgEnvelop_NoWait` should be used instead.

If the message requested with `PxMsgEnvelop` is sent to another task, the `data_area` must not be accessed by the requesting task until the recipient releases the message. `PxMsgAwaitRel` may be used to await the message's release.

9.5. PxMsgEnvelop_NoWait

Function

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

Before call

`data_area` must be a pointer to a valid data area.

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

If the message requested with `PxMsgEnvelop_NoWait` is sent to another task, the `data_area` must not be accessed by the requesting task until the recipient releases the message. `PxMsgAwaitRel` may be used to await the message's release.

9.6. PxMsgEnvelop_EvWait

Function

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

Before call

`data_area` must be a pointer to a valid data area.

`msgsize` must be a plausible value given as a constant (V) or a variable (C).

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxMsgEvent_t`. The received events are stored in the `events` part, the message id is given in the `msg` part of the structure. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

If the message requested with `PxMsgEnvelop_EvWait` is sent to another task, the `data_area` must not be accessed by the requesting task until the recipient releases the message. `PxMsgAwaitRel` may be used to await the message's release. `PxMsgEnvelop_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxMsgEnvelop_NoWait` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

9.7. PxMsgRelease

Function

`PxMsg_t PxMsgRelease(PxMsg_t Msg)`

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received

by a PxMsgReceive... call (V). This id may be checked with one of the following macros:

- PxMsgIdIsValid() must be true.
- PxMsgIdGet() must not be _PXIllegalObjId.
- PxMsgIdError() must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

PxMsgRelease returns the message object to the object pool it has been taken from. On success PxMsgRelease returns the invalidated message object. This may be checked with one of the following macros:

- PxMsgIdIsValid() must be false.
- PxMsgIdGet() must be _PXIllegalObjId.
- PxMsgIdError() must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After PxMsgRelease, the given message object Msg is no longer valid and may never be used as message object!

9.8. PxMsgRelease_Hnd

Function

```
PxMsg_t PxMsgRelease_Hnd(PxMsg_t Msg)
```

Before call

Msg must be a valid message object, requested via PxMsgRequest... or PxMsgEnvelop... or received by a PxMsgReceive... call (V). This id may be checked with one of the following macros:

- PxMsgIdIsValid() must be true.
- PxMsgIdGet() must not be _PXIllegalObjId.
- PxMsgIdError() must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

PxMsgRelease_Hnd returns the message object to the object pool it has been taken from. On success PxMsgRelease_Hnd returns the invalidated message object. This may be checked with one of the following macros:

- PxMsgIdIsValid() must be false.
- PxMsgIdGet() must be _PXIllegalObjId.
- PxMsgIdError() must be PXERR_NOERROR otherwise the returned error code has to be

interpreted (C).

Code of practice

After `PxMsgRelease_Hnd`, the given message object `Msg` is no longer valid and may never be used as message object!

This function should be called from handlers only. (V)

9.9. PxMsgAwaitRel

Function

```
PxMsg_t PxMsgAwaitRel(PxMsg_t Msg)
```

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The calling task must be the message's creator (V), and - if the message is created with `PxMsgRequest...` - the message must be prepared by calling `PxMsgSetToAwaitRel` (V).

After call

The return value of `PxMsgAwaitRel` contains the id of the original message. This may also be checked with the appropriate macros. If the message id is valid, the message may be released by `PxMsgRelease`.

Code of practice

`PxMsgAwaitRel` may block, if the message is never released. If blocking calls are prohibited, `PxMsgAwaitRel_NoWait` should be used instead.

9.10. PxMsgAwaitRel_EvWait

Function

```
PxMsgEvent_t PxMsgAwaitRel_EvWait(PxMsg_t Msg, PxEvents_t events)
```

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.

- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).
The calling task must be the message's creator (V), and - if the message is created with `PxMsgRequest...` - the message must be prepared by calling `PxMsgSetToAwaitRel` (V).

After call

The returned value is a structure of type `PxMsgEvent_t`. The received events are stored in the `events` part, the message id is given in the `msg` part of the structure. This id may also be checked with the appropriate macros. If the message id is valid, the message may be released by `PxMsgRelease`.

Code of practice

`PxMsgAwaitRel_EvWait` may block, if the message is never released and no nstance (task or handler) sends an event. If blocking calls are prohibited, `PxMsgAwaitRel_NoWait` should be used instead.

9.11. PxMsgAwaitRel_NoWait

Function

```
PxMsg_t PxMsgAwaitRel_NoWait(PxMsg_t Msg)
```

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The calling task must be the message's creator (V), and - if the message is created with `PxMsgRequest...` - the message must be prepared by calling `PxMsgSetToAwaitRel` (V).

After call

The return value of `PxMsgAwaitRel` contains the id of the original message, if the message has been released, else the message id is invalid. This may also be checked with the appropriate macros. If the message id is valid, the message may be released by `PxMsgRelease`.

Code of practice

No restrictions.

9.12. PxMsgSend

Function

`PxMsg_t PxMsgSend(PxMsg_t Msg, PxMbx_t mbxid)`

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest`... or `PxMsgEnvelop`... or received by a `PxMsgReceive`... call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The mailbox may be checked with the macros

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).

After call

On success `PxMsgSend` returns the invalidated message object. This may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be false.
- `PxMsgIdGet()` must be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

9.13. PxMsgSend_Prio

Function

`PxMsg_t PxMsgSend_Prio(PxMsg_t Msg, PxMbx_t mbxid)`

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The mailbox may be checked with the macros

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).

After call

On success `PxMsgSend_Prio` returns the invalidated message object. This may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be false.
- `PxMsgIdGet()` must be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

9.14. PxMsgSend_Hnd

Function

`PxMsg_t PxMsgSend_Hnd(PxMsg_t Msg, PxMbx_t mbxid)`

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The mailbox may be checked with the macros

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).

After call

On success `PxMsgSend_Hnd` returns the invalidated message object. This may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be false.
- `PxMsgIdGet()` must be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

As handlers are not allowed to request PXROS-HR objects, the message to be sent has to be requested by a PXROS-HR task and then passed to the handler.

9.15. PxMsgSend_PrioHnd

Function

`PxMsg_t PxMsgSend_PrioHnd(PxMsg_t Msg, PxMbx_t mbxid)`

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest`... or `PxMsgEnvelop`... or received by a `PxMsgReceive`... call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The mailbox may be checked with the macros

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).

After call

On success `PxMsgSend_PrioHnd` returns the invalidated message object. This may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be false.
- `PxMsgIdGet()` must be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

As handlers are not allowed to request PXROS-HR objects, the message to be sent has to be requested by a PXROS-HR task and then passed to the handler.

9.16. PxMsgReceive

Function

`PxMsg_t PxMsgReceive(PxMbx_t mbxid)`

Before call

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

  The mailbox may be checked with the macros

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).
The mailbox mbx must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMbxCoreId` and the own core id with `PxGetCoreId` (C).

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMsgReceive` may block, if no message is available at the mailbox. If blocking calls are prohibited, `PxMsgReceive_NoWait` should be used instead.

9.17. PxMsgReceive_NoWait

Function

`PxMsg_t PxMsgReceive_NoWait(PxMbx_t mbxid)`

Before call

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

  The mailbox may be checked with the macros

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).
The mailbox mbx must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMbxCoreId` and the own core id with `PxGetCoreId` (C).

After call

The returned value is the id of type `PxMsg_t`, if a message is taken from the mailbox. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

9.18. PxMsgReceive_EvWait

Function

```
PxMsgEvent_t PxMsgReceive_EvWait(PxMbx_t mbxid, PxEvents_t events)
```

Before call

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

  The mailbox may be checked with the macros

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.

- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).
The mailbox mbx must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMbxCoreId` and the own core id with `PxGetCoreId` (C).
The parameter events contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxMsgEvent_t`. The received events are stored in the `events` part, the message id is given in the `msg` part of the structure, if a message is taken from the mailbox. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxMsgReceive_EvWait` may block, if no message is available at the mailbox and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxMsgReceive_NoWait()` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

9.19. PxMsgInstallRelmbx

Function

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

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The parameter `mbxid` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)

- part of a message sent by another task (V)

  The mailbox may be checked with the macros

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).
The mailbox mbxid must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxMbxCoreId` and the own core id with `PxGetCoreId` (C).

After call

The function returns `PXERR_NOERROR` if the mailbox is registered as release mailbox for the appropriate message. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function typically is used to create so called "message pools". These message pools should be created during initialization to ensure their availability.

9.20. PxMsgSetSize

Function

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

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

`size` must have a plausible value (V).

After call

The function returns `PXERR_NOERROR` if the message's size is changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.21. PxMsgSetData

Function

`PxError_t PxMsgSetData(PxMsg_t msgid, PxMsgData_t new_data)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

`new_data` must be a pointer to an address within the message's orignial data buffer (V).

After call

The function returns `PXERR_NOERROR` if the message's data area is changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.22. PxMsgSetMetadata

Function

`PxError_t PxMsgSetMetadata(PxMsg_t msgid, PxMsgMetadata_t metadata)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This message object may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the message's metadata is set. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.23. PxMsgSetMetadata_Hnd

Function

`PxError_t PxMsgSetMetadata_Hnd(PxMsg_t msgid, PxMsgMetadata_t metadata)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This message object may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the message's metadata is set. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

9.24. PxMsgSetProtection

Function

`PxError_t PxMsgSetProtection(PxMsg_t msgid, PxProtectType_t protection)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

  `protection` must be of type `PxProtectType_t` and may have one of the following values (V):

- `NoAccessProtection` Neither read nor write access
- `ReadProtection` read access
- `WriteProtection` write access
- `WRProtection` read and write access

After call

The function returns `PXERR_NOERROR` if the message's protection mode is changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.25. PxMsgSetToAwaitRel

Function

```
PxError_t PxMsgSetToAwaitRel(PxMsg_t Msg)
```

Before call

`Msg` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the message is set to `AwaitRelease`. Any other return value describes an error, which has to be interpreted (C).

Code of practice

`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.

9.26. PxMsgGetData

Function

```
PxMsgData_t PxMsgGetData(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

`PxMsgGetData` returns a null pointer on failure (C).

Code of practice

No restrictions.

9.27. PxMsgGetData_Hnd

Function

```
PxMsgData_t PxMsgGetData_Hnd(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest`... or `PxMsgEnvelop`... or received by a `PxMsgReceive`... call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

`PxMsgGetData_Hnd` returns a null pointer on failure (C).

Code of practice

This function should be called from handlers only. (V)

9.28. PxMsgGetMetadata

Function

```
PxMsgMetadata_t PxMsgGetMetadata(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest`... or `PxMsgEnvelop`... or received by a `PxMsgReceive`... call (V). This message object may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

`PxMsgGetMetadata` returns a null value on failure (C). In this case the task must call `PxGetError` to check which error has occurred (F).

Code of practice

No restrictions.

9.29. PxMsgGetMetadata_Hnd

Function

`PxMsgMetadata_t PxMsgGetMetadata_Hnd(PxMsg_t msgid)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This message object may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

`PxMsgGetMetadata_Hnd` returns a null value on failure (C).

Code of practice

This function should be called from handlers only. (V)

9.30. PxMsgGetOwner

Function

`PxTask_t PxMsgGetOwner(PxMsg_t msgid)`

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The returned value is the id of type `PxTask_t`. This id may be checked with one of the following macros:

- `PxTaskIdIsValid()` must be true.
- `PxTaskIdGet()` must not be `_PXIllegalObjId`.

- `PxTaskIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Additionally the task id may be checked with `PxTaskCheck()` (F).

Code of practice

No restrictions.

9.31. PxMsgGetSender

Function

```
PxTask_t PxMsgGetSender(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The returned value is the task object id of type `PxTask_t`. This id may be checked with one of the following macros:

- `PxTaskIdIsValid()` must be true.
- `PxTaskIdGet()` must not be `_PXIllegalObjId`.
- `PxTaskIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Additionally the task object may be checked with `PxTaskCheck()` (F).

Code of practice

No restrictions.

9.32. PxMsgGetSize

Function

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

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns 0 if the given message has no data area (e.g. short message) or is invalid. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions.

9.33. PxMsgGetBuffersize

Function

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

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns 0 if the given message has no data area (e.g. short message) or is invalid. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions.

9.34. PxMsgGetProtection

Function

```
PxProtectType_t PxMsgGetProtection(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.

- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

`PxMsgGetProtection` returns a value of type `PxProtectType_t`, which may have one of the following values (V):

- `NoAccessProtection` Neither read nor write access
- `ReadProtection` read access
- `WriteProtection` write access
- `WRProtection` read and write access

Code of practice

No restrictions.

9.35. PxMsgForceRelease

Function

```
PxError_t PxMsgForceRelease(PxMsg_t msgId)
```

Before call

`msgId` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the message could be released. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.36. PxMsgRelDataAccess

Function

```
PxError_t PxMsgRelDataAccess(PxMsg_t msgid)
```

Before call

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the access to the message data could be released. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

9.37. PxMsgReleaseAllMsg

Function

```
PxError_t PxMsgReleaseAllMsg(void)
```

Before call

No checks necessary.

After call

The function returns `PXERR_NOERROR` if the messages could be released. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

No restrictions.

10. Object Functions

10.1. PxGetObjsize

Function

`PxSize_t PxGetObjsize(void)`

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

10.2. PxObjGetName

Function

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

Before call

`objid` must be a valid PXROS-HR object, which may be checked with one of the following macros:

- `PxObjIdIsValid()` must be true.
- `PxObjIdGet()` must not be `_PXIllegalObjId`.
- `PxObjIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

`buffer` points to a valid memory area of `bufsize` bytes length, where the object's name is stored.

After call

The function returns `PXERR_NOERROR` if the object's name could be copied. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

10.3. PxObjSetName

Function

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

Before call

`objid` must be a valid PXROS-HR object, which may be checked with one of the following macros:

- `PxObjIdIsValid()` must be true.
- `PxObjIdGet()` must not be `_PXIllegalObjId`.
- `PxObjIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

The object objid must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxObjCoreId` and the own core id with `PxGetCoreId` (C). `name` must be a null terminated string.

After call

The function returns `PXERR_NOERROR` if the object name could be set. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

10.4. PxSysObjReleaseAllObjects

Function

```
PxError_t PxSysObjReleaseAllObjects(void)
```

Before call

No checks necessary.

After call

The function returns `PXERR_NOERROR` if the objects could be released. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

No restrictions.

11. Object Pool Functions

11.1. PxOpoolRequest

Function

`PxOpool_t PxOpoolRequest(PxOpoolType_t opooltype, PxUInt_t capacity, PxOpool_t src, PxOpool_t opoolid)`

Before call

`opooltype` must be of type `PxOpoolType_t` and may have one of the following values

- `PXOpoolReal`
- `PXOpoolVirtual`

`capacity` should be a plausible value held in a variable (C) or in a constant (V).
`src` and `opoolid` must be valid PXROS-HR object pools and the calling task must have the access right to take objects from them. The validity of `src` and `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxOpool_t`. This id may be checked with one of the following macros:

- `PxOpoolIdIsValid()` must be true.
- `PxOpoolIdGet()` must not be `_PXIllegalObjId`.
- `PxOpoolIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxOpoolRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxOpoolRequest_NoWait` should be used instead.

`PxOpoolRequest` should only be called during initialization to ensure the availability of the object pool.

11.2. PxOpoolRequest_NoWait

Function

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

Before call

oppoltype must be of type PxOpoolType_t and may have one of the following values

- PXOpoolReal
- PXOpoolVirtual

capacity should be a plausible value held in a variable (C) or in a constant (V).
src and opoolid must be valid PXROS-HR object pools and the calling task must have the access right to take objects from them. The validity of src and opoolid may also be checked by the PxOpoolIsValid macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro PxOpoolCoreId and the own core id with PxGetCoreId (C). Typically the task's default object pool PXOpoolTaskdefault is used for this purpose.

After call

The returned value is the id of type PxOpool_t. This id may be checked with one of the following macros:

- PxOpoolIdIsValid() must be true.
- PxOpoolIdGet() must not be _PXIllegalObjId.
- PxOpoolIdError() must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

PxOpoolRequest_NoWait should only be called during initialization to ensure the availability of the object pool.

11.3. PxOpoolRelease

Function

```
PxOpool_t PxOpoolRelease(PxOpool_t Opool)
```

Before call

Opool must be a valid PXROS-HR object pool created with a PxOpoolRequest call (V). The validity of Opool may also be checked by the PxOpoolIsValid macro (F).

After call

PxOpoolRelease returns the object pool to the object pool it has been taken from. On success PxOpoolRelease returns the invalidated object pool. This may be checked with one of the following macros:

- PxOpoolIdIsValid() must be false.
- PxOpoolIdGet() must be _PXIllegalObjId.
- PxOpoolIdError() must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxOpoolRelease`, the given object pool `Opool` is no longer valid and may never be used as object pool!

11.4. PxOpoolGetCurrentCapacity

Function

```
PxUInt_t PxOpoolGetCurrentCapacity(PxOpool_t opoolid)
```

Before call

`opoolid` must be a valid PXROS-HR object pool. The validity of `opoolid` may be checked by the following macros:

- `PxOpoolIdIsValid()` must be true.
- `PxOpoolIdGet()` must not be `_PXIllegalObjId`.
- `PxOpoolIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns 0 if the given object pool is invalid. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions.

11.5. PxOpoolGetType

Function

```
PxOpoolType_t PxOpoolGetType(PxOpool_t opoolid)
```

Before call

`opoolid` must be a valid PXROS-HR object pool. The validity of `opoolid` may be checked by the following macros:

- `PxOpoolIdIsValid()` must be true.
- `PxOpoolIdGet()` must not be `_PXIllegalObjId`.
- `PxOpoolIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXOpoolIllegalType` if the given object pool is invalid. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions.

11.6. PxOpoolResolveDefault

Function

`PxOpool_t PxOpoolResolveDefault(PxOpool_t opoolid)`

Before call

`opoolid` must be

- a valid PXROS-HR object pool created with a `PxOpoolRequest` call (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F).
- the symbolic value `PXOpoolSystemdefault` specifying the system object pool(V)
- the symbolic value `PXOpoolTaskdefault` specifying the task's object pool(V)

After call

The returned value is the id of type `PxOpool_t`. This id may be checked with one of the following macros:

- `PxOpoolIdIsValid()` must be true.
- `PxOpoolIdGet()` must not be `_PXIllegalObjId`.
- `PxOpoolIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions.

12. TriCore™ Implementation to access Peripheral Register Functions

12.1. PxRegisterRead

Function

`PxULong_t PxRegisterRead(volatile PxULong_t *addr)`

Before call

`addr` has to be a valid pointer to a special function register (V).

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

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`.

12.2. PxRegisterWrite

Function

`PxError_t PxRegisterWrite(volatile PxULong_t *addr, PxInt_t val)`

Before call

`addr` has to be a valid pointer to a special function register (V).

After call

The function returns `PXERR_NOERROR` if the register could be written. Any other return value describes an error, which has to be interpreted (C).

Code of practice

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`.

12.3. PxRegisterSetMask

Function

`PxError_t PxRegisterSetMask(volatile PxULong_t *addr, PxInt_t mask, PxInt_t val)`

Before call

`addr` has to be a valid pointer to a special function register (V).

After call

The function returns `PXERR_NOERROR` if the register could be modified. Any other return value describes an error, which has to be interpreted (C).

Code of practice

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`.

12.4. PxRegisterRead_Hnd

Function

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

Before call

This function should be called from handlers only. (V)

`addr` has to be a valid pointer to a special function register (V).

After call

The function returns 0 if an error occurred. In this case the appropriate task must call `PxGetError` to check if an error has occurred (F).

Code of practice

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` of the task which has installed the handler.

12.5. PxRegisterWrite_Hnd

Function

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

Before call

This function should be called from handlers only. (V)

`addr` has to be a valid pointer to a special function register (V).

After call

The function returns `PXERR_NOERROR` if the register could be written. Any other return value describes an error, which has to be interpreted (C).

Code of practice

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` of the task which has installed the handler.

12.6. PxRegisterSetMask_Hnd

Function

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

Before call

This function should be called from handlers only. (V)

`addr` has to be a valid pointer to a special function register (V).

After call

The function returns `PXERR_NOERROR` if the register could be modified. Any other return value describes an error, which has to be interpreted (C).

Code of practice

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` of the task which has installed the handler.

13. Task Functions

13.1. PxTaskCreate

Function

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

Before call

`opool` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opool` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

`taskspec` has to be a valid task specification structure (V).

`prio` must be a plausible value for the priority of the new task (typically 0 - 31) (V).

The parameter `actevents` contains a bitmask of events awaited and may be zero, if the task should be activated immediately. Typically the event mask is a constant (V).

After call

The returned value is the id of type `PxTask_t`. This id may be checked with one of the following macros:

- `PxTaskIdIsValid()` must be true.
- `PxTaskIdGet()` must not be `_PXIllegalObjId`.
- `PxTaskIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxTaskCheck()` (C).

Code of practice

`PxTaskCreate` should only be called during initialization to ensure the availability of the task.

13.2. PxTaskCheck

Function

```
PxBool_t PxTaskCheck(PxTask_t taskid)
```

Before call

`taskid` must be a valid PXROS-HR task object created with a `PxTaskCreate` call (V). The validity of `taskid` may also be checked by the `PxTaskIsValid` macro (F).

After call

No checks necessary.

Code of practice

No restrictions.

13.3. PxTaskGetMbx

Function

```
PxMbx_t PxTaskGetMbx(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The returned value is the id of type `PxMbx_t`. This id may be checked with one of the following macros:

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.
- `PxMbxIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

or with a call of `PxMbxCheck()` (C).

Code of practice

No restrictions.

13.4. PxTaskSuspend

Function

```
PxError_t PxTaskSuspend(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)

- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C). Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns `PXERR_NOERROR` if the task could be suspended. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.5. PxTaskSuspend_Pxhnd

Function

```
PxError_t PxTaskSuspend_Pxhnd(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C). Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns `PXERR_NOERROR` if the task could be suspended. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

13.6. PxTaskResume

Function

```
PxError_t PxTaskResume(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C). Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns `PXERR_NOERROR` if the task could be resumed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.7. PxTaskSetPrio

Function

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

Before call

The parameter `task` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C). Additionally the task id may be checked with `PxTaskCheck()` (F).
`prio` should be a plausible priority value (typicall between 0 and 31) (V)

After call

The function returns `PXERR_NOERROR` if the task's priority could be changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.8. PxTaskGetPrio

Function

```
PxPrio_t PxTaskGetPrio(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

This function returns a plausible priority value (typical between 0 and 31) (C)

Code of practice

No restrictions.

13.9. PxDieService

Function

```
PxError_t PxDieService(void)
```

Before call

No checks necessary.

After call

The function returns `PXERR_NOERROR` if the die service was successfully executed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.10. PxDie

Function

```
PxError_t PxDie(void)
```

Before call

No checks necessary.

After call

The function does not return on success, else it returns an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.11. PxGetId

Function

```
PxTask_t PxGetId(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

13.12. PxSetPrivileges

Function

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

Before call

No checks necessary.

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

13.13. PxGetPrivileges

Function

```
PxArg_t PxGetPrivileges(void)
```

Before call

No checks necessary.

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

13.14. PxSetModebits

Function

`PxTmode_t PxSetModebits(PxTmode_t modebits)`

Before call

`modebits` may be a combination of the following bits (V):

- `PXTmodeDisableAborts`
- `PXTmodeDisableTimeslicing`

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

13.15. PxClearModebits

Function

`PxTmode_t PxClearModebits(PxTmode_t modebits)`

Before call

`modebits` may be a combination of the following bits (V):

- `PXTmodeDisableAborts`
- `PXTmodeDisableTimeslicing`

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

13.16. PxTaskGetModebits

Function

`PxTmode_t PxTaskGetModebits(PxTask_t taskid)`

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)

- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C). Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

`PxGetError` must be called to check if an error has occurred. (F)

Code of practice

No restrictions.

13.17. PxGetTimeslices

Function

```
PxTicks_t PxGetTimeslices(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

13.18. PxSetTimeslices

Function

```
void PxSetTimeslices(PxTicks_t timeslices)
```

Before call

`timeslices` has to be a plausible value.

After call

No checks necessary.

Code of practice

No restrictions.

13.19. PxRemoveAccessRights

Function

`PxUInt_t PxRemoveAccessRights(PxUInt_t accessrights)`

Before call

`accessrights` may be a combination of valid access rights (V)

After call

No checks necessary.

Code of practice

No restrictions.

13.20. PxRestoreAccessRights

Function

`PxError_t PxRestoreAccessRights(PxUInt_t accessrights)`

Before call

`accessrights` may be a combination of valid access rights (V)

After call

The function returns `PXERR_NOERROR` if the access rights could be restored. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.21. PxTaskForceTermination

Function

`PxError_t PxTaskForceTermination(PxTask_t taskId)`

Before call

The parameter `taskId` must be a valid task object id. This id may be

- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F). The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C).

After call

The function returns `PXERR_NOERROR` if the task could be terminated. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.22. PxTaskGetName

Function

```
PxError_t PxTaskGetName(PxTask_t taskid, PxChar_t *buffer, PxUInt_t bufsize)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).
`buffer` should point to a valid memory area, where the task's name is stored.
`bufsize` should be a constant or a `sizeof()` operator (V).

After call

The function returns `PXERR_NOERROR` if the task's name could be stored. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

13.23. PxTaskGetSize

Function

```
PxSize_t PxTaskGetSize(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

13.24. PxTaskGetAccessRights

Function

`PxUInt_t PxTaskGetAccessRights(PxTask_t taskid)`

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F). The task object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxTaskCoreId` and the own core id with `PxGetCoreId` (C).

After call

`PxTaskGetAccessRights` returns a null value on failure (C). In this case the task must call `PxGetError` to check which error has occurred (F).

Code of practice

No restrictions.

13.25. PxTerminate

Function

`PxError_t PxTerminate(PxBool_t release)`

Before call

`release` must be TRUE or FALSE. This parameter should be a constant (V).

After call

On success the function does not return. Any return value describes an error, which has to be interpreted.

Code of practice

No restrictions.

14. Time Management Functions

14.1. PxTickGetCount

Function

`PxTicks_t PxTickGetCount(void)`

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

14.2. PxTickDefine_Hnd

Function

`void PxTickDefine_Hnd(void)`

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

This function should be called from handlers only. (V)

14.3. PxPeRequest

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `period` parameter must have a plausible value. It should be a constant or the result of a

`PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The returned value is the id of type `PxPe_t`. This id may be checked with one of the following macros:

- `PxPeIdIsValid()` must be true.
- `PxPeIdGet()` must not be `_PXIllegalObjId`.
- `PxPeIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxPeRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxPeRequest_NoWait` should be used instead.

`PxPeRequest` should only be called during initialization to ensure the availability of the periodic event object.

14.4. PxPeRequest_NoWait

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `period` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The returned value is the id of type `PxPe_t`. This id may be checked with one of the following macros:

- `PxPeIdIsValid()` must be true.

- `PxPeIdGet()` must not be `_PXIllegalObjId`.
- `PxPeIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxPeRequest_NoWait` should only be called during initialization to ensure the availability of the periodic event object.

14.5. PxPeRequest_EvWait

Function

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

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `period` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

The parameter `abortevents` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxPeEvent_t`. The received events are stored in the `events` part, the delay id is given in the pe part of the structure. This id may be checked with one of the following macros:

- `PxPeIdIsValid()` must be true.
- `PxPeIdGet()` must not be `_PXIllegalObjId`.
- `PxPeIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxPeRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxPeRequest_NoWait` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxPeRequest_EvWait` should only be called during initialization to ensure the availability of the periodic event object.

14.6. PxPeStart

Function

```
PxError_t PxPeStart(PxPe_t PeId)
```

Before call

`PeId` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `PeId` may also be checked by the `PxPeIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the periodic event could be started. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.7. PxPeStart_Hnd

Function

```
PxError_t PxPeStart_Hnd(PxPe_t PeId)
```

Before call

`PeId` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `PeId` may also be checked by the `PxPeIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the periodic event could be started. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

14.8. PxPeStop

Function

```
PxError_t PxPeStop(PxPe_t PeId)
```

Before call

`PeId` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `PeId` may also be checked by the `PxPeIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the periodic event could be stopped. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.9. PxPeStop_Hnd

Function

```
PxError_t PxPeStop_Hnd(PxPe_t PeId)
```

Before call

`PeId` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `PeId` may also be checked by the `PxPeIsValid()` macro (F).

After call

The function returns `PXERR_NOERROR` if the periodic event could be stopped. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

14.10. PxPeChange

Function

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

Before call

`PeId` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `PeId` may also be checked by the `PxPeIsValid` macro (F).

The `period` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The function returns `PXERR_NOERROR` if the periodic event could be changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.11. PxPeRelease

Function

`PxPe_t PxPeRelease(PxPe_t Pe)`

Before call

`Pe` must be a valid PXROS-HR periodic event object created with a `PxPeRequest` call (V). The validity of `Pe` may also be checked by the `PxPeIsValid` macro (F).

After call

`PxPeRelease` returns the periodic event object to the object pool it has been taken from. On success `PxPeRelease` returns the invalidated periodic event object. This may be checked with one of the following macros:

- `PxPeIdIsValid()` must be false.
- `PxPeIdGet()` must be `_PXIllegalObjId`.
- `PxPeIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxPeRelease`, the given periodic event object `Pe` is no longer valid and may never be used as periodic event object!

14.12. PxToRequest

Function

`PxTo_t PxToRequest(PxOpool_t opoolid, PxTicks_t timeout, PxEvents_t events)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `timeout` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The returned value is the id of type `PxTo_t`. This id may be checked with one of the following macros:

- `PxToIdIsValid()` must be true.
- `PxToIdGet()` must not be `_PXIllegalObjId`.
- `PxToIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxToRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxToRequest_NoWait()` should be used instead.

`PxToRequest` should only be called during initialization to ensure the availability of the timeout object.

14.13. PxToRequest_NoWait

Function

```
PxTo_t PxToRequest_NoWait(PxOpool_t opoolid, PxTicks_t timeout, PxEvents_t events)
```

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `timeout` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The returned value is the id of type `PxTo_t`. This id may be checked with one of the following macros:

- `PxToIdIsValid()` must be true.
- `PxToIdGet()` must not be `_PXIllegalObjId`.
- `PxToIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxToRequest_NoWait` should only be called during initialization to ensure the availability of the timeout object.

14.14. PxToRequest_EvWait

Function

```
PxTo_t PxToRequest_EvWait(PxOpool_t opoolid, PxTicks_t timeout, PxEvents_t events, PxEvents_t abortevents)
```

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The `timeout` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

The parameter `abortevents` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxToEvent_t`. The received events are stored in the `events` part, the delay id is given in the `to` part of the structure. This id may be checked with one of the following macros:

- `PxToIdIsValid()` must be true.
- `PxToIdGet()` must not be `_PXIllegalObjId`.
- `PxToIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxToRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxToRequest_NoWait()` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxToRequest_EvWait` should only be called during initialization to ensure the availability of the timeout object.

14.15. PxToStart

Function

```
PxError_t PxToStart(PxTo_t ToId)
```

Before call

`ToId` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `ToId` may also be checked by the `PxToIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the timeout could be started. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.16. PxToStart_Hnd

Function

```
PxError_t PxToStart_Hnd(PxTo_t ToId)
```

Before call

`ToId` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `ToId` may also be checked by the `PxToIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the timeout could be started. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

14.17. PxToStop

Function

```
PxError_t PxToStop(PxTo_t ToId)
```

Before call

`ToId` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `ToId` may also be checked by the `PxToIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the timout could be stopped. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.18. PxToStop_Hnd

Function

```
PxError_t PxToStop_Hnd(PxTo_t ToId)
```

Before call

`ToId` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `ToId` may also be checked by the `PxToIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the timout could be stopped. Any other return value describes an error, which has to be interpreted (C).

Code of practice

This function should be called from handlers only. (V)

14.19. PxToChange

Function

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

Before call

`ToId` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `ToId` may also be checked by the `PxToIsValid` macro (F).

The `timeout` parameter must have a plausible value. It should be a constant or the result of a `PxTickGetTicksFromMilliSeconds` call (V);

The parameter `events` contains an event bit and should not be zero. Typically the event is a constant (V).

After call

The function returns `PXERR_NOERROR` if the timout could be changed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions.

14.20. PxToRelease

Function

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

Before call

`To` must be a valid PXROS-HR timeout object created with a `PxToRequest` call (V). The validity of `To` may also be checked by the `PxToIsValid` macro (F).

After call

`PxToRelease` returns the timeout object to the object pool it has been taken from. On success `PxToRelease` returns the invalidated timeout object. This may be checked with one of the following macros:

- `PxToIdIsValid()` must be false.
- `PxToIdGet()` must be `_PXIllegalObjId`.
- `PxToIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxToRelease`, the given timeout object `To` is no longer valid and may never be used as timeout object!

14.21. PxTickGetTicksFromMilliSeconds

Function

```
PxTicks_t PxTickGetTicksFromMilliSeconds(PxULong_t millis)
```

Before call

`millis` must be a plausible value given as a constant (V) or a variable (C).

After call

No checks necessary.

Code of practice

No restrictions.

14.22. PxTickGetTimeInMilliSeconds

Function

```
PxULong_t PxTickGetTimeInMilliSeconds(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

14.23. PxTickSetTicksPerSecond

Function

`PxError_t PxTickSetTicksPerSecond(PxUInt_t tickspersecond)`

Before call

`tickspersecond` must be a plausible value given as a constant (V) or a variable (C).

After call

The function returns `PXERR_NOERROR` if the ticks per second could be set. Any other return value describes an error, which has to be interpreted. (C)

Code of practice

No restrictions.

15. Interrupt and Trap Functions

15.1. PxInterruptRequest

Function

`PxInterrupt_t PxInterruptRequest(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxInterrupt_t`. This id may be checked with one of the following macros:

- `PxInterruptIdIsValid()` must be true.
- `PxInterruptIdGet()` must not be `_PXIllegalObjId`.
- `PxInterruptIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxInterruptRequest` may block, if no PXROS-HR object is available. If blocking calls are prohibited, `PxInterruptRequest_NoWait()` should be used instead.

`PxInterruptRequest` should only be called during initialization to ensure the availability of the interrupt object.

15.2. PxInterruptRequest_NoWait

Function

`PxInterrupt_t PxInterruptRequest_NoWait(PxOpool_t opoolid)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxInterrupt_t`. This id may be checked with one of the following macros:

- `PxInterruptIdIsValid()` must be true.
- `PxInterruptIdGet()` must not be `_PXIllegalObjId`.
- `PxInterruptIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxInterruptRequest_NoWait` should only be called during initialization to ensure the availability of the interrupt object.

15.3. PxInterruptRequest_EvWait

Function

`PxInterruptEvent_t PxInterruptRequest_EvWait(PxOpool_t opoolid, PxEvents_t events)`

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

The parameter `events` contains a bitmask of events awaited and should not be zero. Typically the event mask is a constant (V).

After call

The returned value is a structure of type `PxInterruptEvent_t`. The received events are stored in the `events` part, the interrupt object id is given in the `interrupt` part of the structure. This id may be checked with one of the following macros:

- `PxInterruptIdIsValid()` must be true.
- `PxInterruptIdGet()` must not be `_PXIllegalObjId`.
- `PxInterruptIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

`PxInterruptRequest_EvWait` may block, if no PXROS-HR object is available and no instance (task or handler) sends an event. If blocking calls are prohibited, `PxInterruptRequest_NoWait()` should be used instead or the call should be monitored by the PXROS-HR `PxTo` mechanism.

`PxInterruptRequest_EvWait` should only be called during initialization to ensure the availability of the interrupt object.

15.4. PxInterruptRelease

Function

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

Before call

`Interrupt` must be a valid PXROS-HR interrupt object created with a `PxInterruptRequest` call (V). The validity of `Interrupt` may also be checked by the `PxInterruptIsValid` macro (F).

After call

`PxInterruptRelease` returns the interrupt object to the object pool it has been taken from. On success `PxInterruptRelease` returns the invalidated interrupt object. This may be checked with one of the following macros:

- `PxInterruptIdIsValid()` must be false.
- `PxInterruptIdGet()` must be `_PXIllegalObjId`.
- `PxInterruptIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

After `PxInterruptRelease`, the given interrupt object `Interrupt` is no longer valid and may never be used as interrupt object!

15.5. PxIntInitVectab

Function

```
void PxIntInitVectab (void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

This function is automatically executed in `PxInit` and must not be called by the application!

15.6. PxIntInstallFastHandler

Function

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

```
PxArg_t arg)
```

Before call

The parameter `intno` must be a valid interrupt id. (V)

`inthandler` must be a pointer to a interrupt handler function.(V)

After call

The function returns `PXERR_NOERROR` if the interrupt handler could be installed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

The calling task must have the right to install fast interrupt handlers (`PXACCESS_HANDLERS`). (V)

15.7. PxIntInstallFastContextHandler

Function

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

Before call

The parameter `intno` must be a valid interrupt id. (V)

`inthandler` must be a pointer to a interrupt handler function.(V)

After call

The function returns `PXERR_NOERROR` if the interrupt handler could be installed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

The calling task must have the right to install fast context interrupt handlers (`PXACCESS_INSTALL_HANDLERS`). (V)

15.8. PxIntInstallHandler

Function

```
PxError_t PxIntInstallHandler(PxUInt_t intno, PxInterrupt_t intObj, void (*
inthandler)(PxArg_t), PxArg_t arg)
```

Before call

The parameter `intno` must be a valid interrupt id. (V)

`intObj` must be a valid PXROS-HR interrupt object created with a `PxInterruptRequest` call (V). The validity of `intObj` may also be checked by the `PxInterruptIsValid` macro (F).

`inthandler` must be a pointer to an interrupt handler function.(V)

After call

The function returns `PXERR_NOERROR` if the interrupt handler could be installed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

The calling task must have the right to install interrupt handlers (`PXACCESS_INSTALL_HANDLERS`). (V)

15.9. PxIntInstallService

Function

```
PxError_t PxIntInstallService(PxUInt_t intno, PxUInt_t service, PxArg_t arg, PxEvents_t events)
```

Before call

The parameter `intno` must be a valid interrupt id. (V)

The parameter `service` must be a valid PXROS-HR service id. (V)

After call

The function returns `PXERR_NOERROR` if the service could be installed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

The calling task must have the right to install services as interrupt handlers (`PXACCESS_INSTALL_SERVICES`). (V)

15.10. PxTrapInitVectab

Function

```
void PxTrapInitVectab(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

This function is automatically executed in `PxInit` and must not be called by the application!

15.11. PxTrapInstallHandler

Function

```
PxError_t PxTrapInstallHandler(PxUInt_t trapno, PxBool_t (*
```

```
traphandler)(PxUInt_t, PxUInt_t, PxUInt_t, PxUInt_t, PxUInt_t *, TC_CSA_t *),
PxUInt_t arg)
```

Before call

The parameter `trapno` must be a valid trap id. (V)

`traphandler` must be a pointer to a trap handler function.(V)

After call

The function returns `PXERR_NOERROR` if the trap handler could be installed. Any other return value describes an error, which has to be interpreted (C).

Code of practice

The calling task must have the right to install trap handlers (`PXACCESS_HANDLERS`). (V)

15.12. PxTrapGetTaskProtection

Function

```
PxDataProtectSet_T PxTrapGetTaskProtection(PxTask_t taskid)
```

Before call

The parameter `taskid` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns a pointer to the task's protection set or a null pointer if taskid is not a valid task object (C).

Code of practice

The caller must be in supervisor mode.

15.13. PxIsOnHndLvl

Function

```
PxInt_t PxIsOnHndLvl(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions

16. Trace Functions

16.1. PxTraceAssignBuffer

Function

```
PxError_t PxTraceAssignBuffer(PxUChar_t *trcbuffer, PxULong_t capacity)
```

Before call

`trcbuffer` must be a pointer to a valid data area.

`capacity` must be a plausible value given as a constant (V) or a variable (C).

After call

The function returns `PXERR_NOERROR` if the trace buffer could be assigned. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

16.2. PxTraceCtrl

Function

```
PxULong_t PxTraceCtrl(PxTraceCtrl_t cmd, PxArg_t arg)
```

Before call

`cmd` must be a valid trace control command. (V)

`arg` must be a valid parameter to the appropriate trace control command. (V)

After call

The function returns -1 if the trace control command could not be executed. In this case `PxGetError` must be called to check which error has occurred. (F)

Code of practice

No restrictions

16.3. PxTraceGetBuffer

Function

```
PxMsg_t PxTraceGetBuffer(PxOpool_t opoolId)
```

Before call

`opoolid` must be a valid PXROS-HR object pool and the calling task must have the access right to take objects from this object pool (V). The validity of `opoolid` may also be checked by the `PxOpoolIsValid` macro (F). The object pool must be created on the same core as the caller runs

on. The creator core id can be read with the macro `PxOpoolCoreId` and the own core id with `PxGetCoreId` (C). Typically the task's default object pool `PXOpoolTaskdefault` is used for this purpose.

After call

The returned value is the id of type `PxMsg_t`. This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

Code of practice

No restrictions

17. System Information Functions

17.1. PxSysInfoGetDelayInfo

Function

```
PxError_t PxSysInfoGetDelayInfo(PxInfoDelay_t *DelayInfo, PxDelay_t delayId)
```

Before call

`DelayInfo` must be a pointer to a valid data area.

`delayId` must be a valid PXROS-HR delay object created with a `PxDelayRequest` call (V). The validity of `delayId` may also be checked by the `PxDelayIsValid` macro (F). The delay object must be created on the same core as the caller runs on. The creator core id can be read with the macro `PxDelayCoreId` and the own core id with `PxGetCoreId` (C).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.2. PxSysInfoGetInterruptInfo

Function

```
PxError_t PxSysInfoGetInterruptInfo(PxInfoInterrupt_t *InterruptInfo,
PxInterrupt_t interruptId)
```

Before call

`InterruptInfo` must be a pointer to a valid data area.

`interruptId` must be a valid PXROS-HR interrupt object created with a `PxInterruptRequest` call (V). The validity of `interruptId` may also be checked by the `PxInterruptIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.3. PxSysInfoGetMCInfo

Function

```
PxError_t PxSysInfoGetMCInfo(PxInfoMC_t *MCInfo, PxMc_t McId)
```

Before call

`MCInfo` must be a pointer to a valid data area.

`McId` must be

- a valid PXROS-HR memory class object created with a `PxMcRequest` call (V).
- the symbolic value `PXMcSystemdefault` specifying the system memory class (V)
- the symbolic value `PXMcTaskdefault` specifying the task's memory class (V)

The validity of `McId` may also be checked by the `PxMcIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.4. PxSysInfoGetMbxInfo

Function

```
PxError_t PxSysInfoGetMbxInfo(PxInfoMbx_t *MbxInfo, PxMbx_t mbxId)
```

Before call

`MbxInfo` must be a pointer to a valid data area.

`mbxId` must be a valid PXROS-HR mailbox object created with a `PxMbxRequest` call or the task's private mailbox (V). The validity of `mbxId` may also be checked by the `PxMbxIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.5. PxSysInfoGetMsgInfo

Function

```
PxError_t PxSysInfoGetMsgInfo(PxInfoMsg_t *MsgInfo, PxMsg_t msgId)
```

Before call

`MsgInfo` must be a pointer to a valid data area.

`msgid` must be a valid message object, requested via `PxMsgRequest...` or `PxMsgEnvelop...` or received by a `PxMsgReceive...` call (V). This id may be checked with one of the following macros:

- `PxMsgIdIsValid()` must be true.
- `PxMsgIdGet()` must not be `_PXIllegalObjId`.
- `PxMsgIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.6. PxSysInfoGetOpoolInfo

Function

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

Before call

`OpoolInfo` must be a pointer to a valid data area.

`opoolId` must be

- a valid PXROS-HR object pool created with a `PxOpoolRequest` call (V). The validity of `opoolId` may also be checked by the `PxOpoolIsValid` macro (F).
- the symbolic value `PXOpoolSystemdefault` specifying the system object pool(V)
- the symbolic value `PXOpoolTaskdefault` specifying the task's object pool(V)

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.7. PxSysInfoGetPeInfo

Function

```
PxError_t PxSysInfoGetPeInfo(PxInfoPe_t *PeInfo, PxPe_t peId)
```

Before call

`PeInfo` must be a pointer to a valid data area.

`peId` must be a valid PXROS-HR periodic event handler object created with a `PxPeRequest` call (V). The validity of `peId` may also be checked by the `PxPeIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.8. PxSysInfoGetToInfo

Function

```
PxError_t PxSysInfoGetToInfo(PxInfoTo_t *ToInfo, PxTo_t toId)
```

Before call

`ToInfo` must be a pointer to a valid data area.

`toId` must be a valid PXROS-HR timeout handler object created with a `PxToRequest` call (V). The validity of `toId` may also be checked by the `PxToIsValid` macro (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.9. PxSysInfoGetTaskInfo

Function

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

Before call

`TaskInfo` must be a pointer to a valid data area.

The parameter `taskId` must be a valid task object id. This id may be

- the calling task's own id read by calling `PxGetId()` (V)
- the return value of a `PxTaskCreate()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

Additionally the task id may be checked with `PxTaskCheck()` (F).

After call

The function returns `PXERR_NOERROR` if the system information could be delivered. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

17.10. PxSysInfoGetMsgsInMbx

Function

```
PxInt_t PxSysInfoGetMsgsInMbx(PxMbx_t mbxId, PxMsgType_t Type, PxObjId_t
*MsgArray, PxUInt_t Max)
```

Before call

`MsgArray` must be a pointer to a valid data area containing space for `Max` object ids.

The parameter `mbxId` must be a valid mailbox object id. This id may be

- the calling task's own mailbox (V)
- the return value of a `PxTaskGetMbx()` call (V)
- the result of a nameserver query (V)
- part of a message sent by another task (V)

The PXROS-HR message type given in `Type` must be of type `PxMsgType_t` and may have one of the following values (V):

- `PXMsgAnyMsg`
- `PXMsgNormalMsg`
- `PXMsgPrioMsg`

After call

The function returns -1 if the number of messages in a mailbox could not be determined (C).

Code of practice

No restrictions

17.11. PxSysInfoGetNumberOfObjects

Function

```
PxUInt_t PxSysInfoGetNumberOfObjects(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions

17.12. PxSysInfoGetObjType

Function

```
_PxObjType_t PxSysInfoGetObjType(PxObj_t objid)
```

Before call

`objid` must be a valid PXROS-HR object, which may be checked with one of the following macros:

- `PxObjIdIsValid()` must be true.
- `PxObjIdGet()` must not be `_PXIllegalObjId`.
- `PxObjIdError()` must be PXERR_NOERROR otherwise the returned error code has to be interpreted (C).

After call

The function returns _PXObjIllObject if the given object is invalid (C).

Code of practice

No restrictions

17.13. PxVersion

Function

```
const PxChar_t * PxVersion(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

17.14. PxGetCoreId

Function

```
PxCoreId_t PxGetCoreId(void)
```

Before call

No checks necessary.

After call

No checks necessary.

Code of practice

No restrictions.

18. Special PXROS-HR Functions

18.1. PxServiceTaskInit

Function

`PxError_t PxServiceTaskInit(void)`

Before call

The calling task must have the right to act as service task (`PXACCESS_SYSTEM_CONTROL`). (V)

After call

The function returns `PXERR_NOERROR` if the task could be installed as service task. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

18.2. PxMbxRequestMbx

Function

`PxMbx_t PxMbxRequestMbx(PxMbxReq_t mbxreqid)`

Before call

`mbxreqid` must be a valid PXROS-HR service mailbox (V).

After call

The function returns the requested service mailbox. This mailbox id may be checked with one of the following macros:

- `PxMbxIdIsValid()` must be true.
- `PxMbxIdGet()` must not be `_PXIllegalObjId`.

Code of practice

No restrictions

18.3. PxMbxRegisterMbx

Function

`PxError_t PxMbxRegisterMbx(PxMbxReq_t mbxreqid, PxMbx_t mbxid)`

Before call

`mbxreqid` must be a valid PXROS-HR service mailbox (V). `mbxid` must be a valid mailbox object (V).

After call

The function returns `PXERR_NOERROR` if the mailbox could be registered for the service mailbox. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

18.4. PxGetGlobalServerMbx

Function

```
PxError_t PxGetGlobalServerMbx(PxUInt8_t ServerCore, PxMbxReq_t mbxreqid)
```

Before call

`ServerCore` must not exceed the number of available cores. (V) `mbxreqid` must be a valid PXROS-HR service mailbox (V).

After call

The function returns `PXERR_NOERROR` if the mailbox could be requested for the service mailbox. Any other return value describes an error, which has to be interpreted (C).

Code of practice

No restrictions

Disclaimer

Please Read Carefully:

This document is furnished by HighTec EDV-Systeme GmbH (HIGHTEC) and contains descriptions for copyrighted products that are not explicitly indicated as such. The absence of the ™ symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this document.

This document is provided "as is" and with all faults. HIGHTEC disclaims all other warranties or representations, express or implied, regarding this document or use thereof, including but not limited to accuracy or completeness, title and any implied warranties of merchantability, fitness for a particular purpose, and non-infringement of any third party intellectual property rights.

HIGHTEC reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

HIGHTEC shall not be liable for and shall not defend or indemnify you against any claim, including but not limited to any infringement claim that relates to or is based on any HIGHTEC product even if described in this document or otherwise. In no event shall HIGHTEC be liable for any actual, direct, special, collateral, indirect, punitive, incidental, consequential, or exemplary damages in connection with or arising out of this document or use thereof, and regardless of whether HIGHTEC has been advised of the possibility of such damages.

HighTec EDV-Systeme GmbH
Europaallee 19, D-66113 Saarbrücken
info@hightec-rt.com
+49-681-92613-16
www.hightec-rt.com