# XOn NTX Toolkit

XOn Software GmbH http:\\www.xon.de

# Table of Contents

# Index

# 1 Introduction


This add-on library for LabVIEW under NT/W2K/XP enables port I/O-access, as well as memory access. Hardware access can also be realized with compatibility-VIs similar to the LabVIEW functions under Win9x.

Under Windows NT access to port-I/O is only permitted for drivers. The system allocates a certain address range for a driver. The XOn port I/O-driver allows the configuration of max 10 address ranges of variable size which can be accessed simultaneously. For the configuration of these address areas the "XOn Port I/O Manager" is shipped with the driver. Thus you can access address ranges that are normally managed by the system.

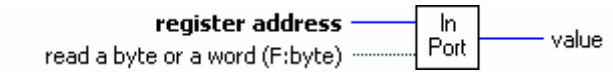
# 2 XOnNTX_PortIO32_VIs

## 2.1 In Port

This VI allows you to read either a single bye or word from the specified register address. The value is returned, in hexadecimal format, to Value.

---

### Connector Pane




---

### Controls and Indicators

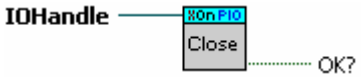
**register address**

**read a byte or a word (F:byte)**

**value**

## 2.2 IOClose

### Connector Pane



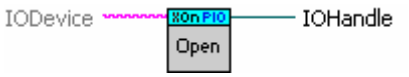

### Controls and Indicators

**IOHandle**

**OK?**

# 2.3 IOOpen

### Connector Pane



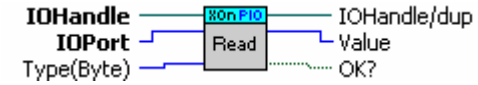

### Controls and Indicators

**IODevice**

**IOHandle**

## 2.4 IORead

### Connector Pane




### Controls and Indicators

**IOHandle**

**IOPort**

**Type(Byte)**

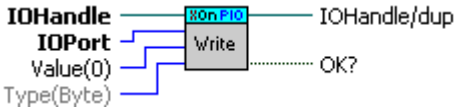
**OK?**

**Value**

**IOHandle/dup**

# 2.5 IOWrite

### Connector Pane




### Controls and Indicators

**IOHandle**

**IOPort**

**Value(0)**

**Type(Byte)**

**OK?**

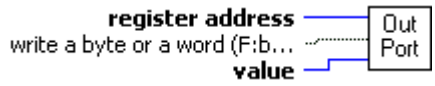
**IOHandle/dup**

## 2.6 Out Port

This VI allows you to write a single byte or word to the specified register address.

---

### Connector Pane




---

### Controls and Indicators

**register address**

**value**

**write a byte or a word (F:byte)**

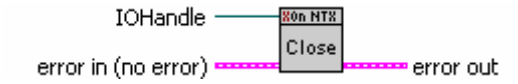
# 3 XOnNTX_BaseIO_VIs

## 3.1 XOnNTX_Close

Closes an explicit or implicit driver handle. See XOnNTX_Open for more information.

---

**Connector Pane**




---

**Controls and Indicators**

**IOHandle**
Driver handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

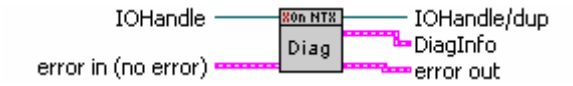
**source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 3.2 XOnNTX_GetDiagnosis

Returns diagnostic information on the last call of a driver-function.

### Connector Pane




### Controls and Indicators

**IOHandle**
Driver-handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**DiagInfo**
Diagnostic information type.

| | |
|---|---|
| **IoctlCode** | Internal code of last called driver function |
| **Status** | Status-Code of call (0=SUCCESS) |
| **Description** | Textual description ("OK" on success) |

**IoctlCode**
Internal code of last called driver function.

**Status**
Values are 32 bit values layed out as follows:


**Sev**
Bit 30-31
is the severity code
- 00 - Success
- 01 - Informational
- 10 - Warning
- 11 - Error

**c**
Bit 29
is the Customer code flag

**r**
Bit 28
is a reserved bit

**Facility**
Bit 16-27
is the facility code

**Code**
Bit 0-15
is the facility's status code

**Define the facility codes**
x2=FACILITY_RPC_RUNTIME
x3=FACILITY_RPC_STUBS
x4=FACILITY_IO_ERROR_CODE
**Define the severity codes**
x0=STATUS_SEVERITY_SUCCESS
x1=STATUS_SEVERITY_INFORMATIONAL
x2=STATUS_SEVERITY_WARNING
x3=STATUS_SEVERITY_ERROR

**Description**
Textual description ("OK" on success).

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

 **source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

 **IOHandle/dup**

Duplicated Driver-handle.

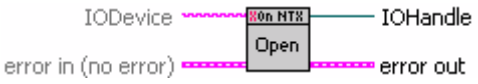
## 3.3 XOnNTX_Open

Opens the NT-Extension Driver and returns a handle to it usable from other XOnNTX functions.

Thus recommended, opening the driver is optional and will be done implicit by the driver DLL on the first call of any XOnNTX function that accepts a driver handle as input. The driver DLL always remembers the last implicit or explicit opened driver handle and uses this on any other XOnNTX functions called with a NULL-handle (NOT connected handle).
If not explicitly closed using XOnNTX_Close, the driver DLL will close the remembered driver-handle (and only this) on DLL-unload (end of application).
WARNING: To avoid unused opened driver handles (wasting system resources) after end of application, either never use XOnNTX_Open, or balance open calls with calls to XOnNTX_Close.

---

### Connector Pane




---

### Controls and Indicators

**IODevice**
Optional parameter for future use. Do NOT use it.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle**
Handle to the opened driver. Use this an other XOnNTX functions that accept a handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred

before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.


**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.


**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.


**source**

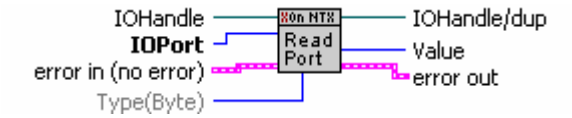
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 3.4 XOnNTX_ReadPort

Reads a single value from the given IO-Port.
You may optionally define the Port-Type Byte, Word or Doubleword (defaults to Byte). In any case the returned value is of type Doubleword.
WARNING: Thus not necessary, using Porttypes other than Byte you should aligne the IOPort appropriatly; 2-Byte for Word-, 4-Byte for Doubleword-Type.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**IOPort**
IO-Port to read a Byte, Word or Doubleword from.

For types other than Byte you should aligne the IOPort; 2-Byte for Word-, 4-Byte for Doubleword-Type.

**Type(Byte)**
Optional porttype, Byte, Word or Doubleword.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

 **Value**
Value read from the given IOPort.

Independently of the give Port-Type always as Doubleword.

 **IOHandle/dup**
Duplicated Driver-handle.

 **error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

 **status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

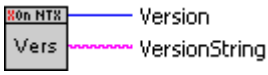
 **source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 3.5 XOnNTX_Version

Returns the Driver-Version as string and value.

---

### Connector Pane




---

### Controls and Indicators

**VersionString**

**Version**

| | |
|---|---|
| **Low-Word** | Minor-Version-Number |
| **High-Word** | Major-Version-Number |
| **BIT-31** | Windows-9x Emulation-Flag<br>0 = Windows-2000/-NT Version<br>1 = Windows-9x Emulation |
| **BIT-30** | Demo-Flag Full-Demo<br>0 = NOT a Demo-Version<br>1 = DEMO-Version |
| **BIT-29** | Demo-Flag HardIO<br>0 = NOT a Demo-Version<br>1 = DEMO-Version |
| **BIT-28** | Demo-Flag Shared-Memory<br>0 = NOT a Demo-Version<br>1 = DEMO-Version |

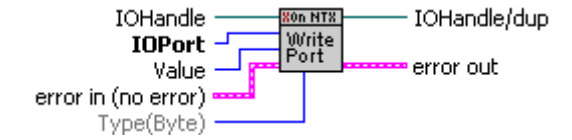
# 3.6 XOnNTX_WritePort

Writes a single value to the given IO-Port.
You may optionally define the Port-Type Byte, Word or Doubleword (defaults to Byte).

WARNING: Thus not necessary, using Porttypes other than Byte you should aligne the IOPort appropriatly; 2-Byte for Word-, 4-Byte for Doubleword-Type.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-Handle.

**IOPort**
IO-Port to write a Byte, Word or Doubleword to.

Types other than Byte requires alignment; 2-Byte for Word-, 4-Byte for Doubleword-Type.

**Value**
Value to write.

**Type(Byte)**
Optional porttype, Byte, Word or Doubleword.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-Handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

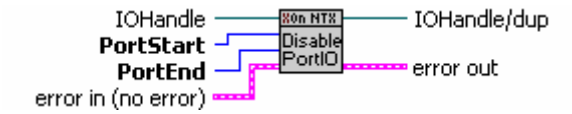
# 4 XOnNTX_HardIO_VIs

## 4.1 XOnNTX_DisablePortIO

Disables direct port access in the range from 'PortStart' to 'PortEnd'.

---

**Connector Pane**




---

**Controls and Indicators**

**IOHandle**
Driver-handle.

**PortStart**
Start port address to disallow direct port access.

**PortEnd**
End port address.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error,

if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.


**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.


**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.


**source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**

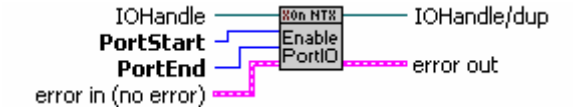
Duplicated Driver-handle.

# 4.2 XOnNTX_EnablePortIO

Enables direct port access in the range from 'PortStart' to 'PortEnd'.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**PortStart**
Start port address to allow direct port access.

**PortEnd**
End port address.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error

input of the next.

 **status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

 **source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

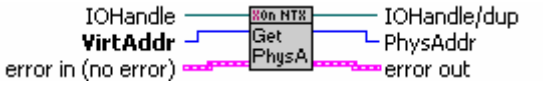
 **IOHandle/dup**
Duplicated Driver-handle.

# 4.3 XOnNTX_GetPhysAddress

Returns the phys. address associated with the given virtual address.
The virtual address should be mapped via XOnNTX_MapPhysMemory.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**VirtAddr**
Virtual start address that should be mapped via XOnNTX_MapPhysMemory.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**PhysAddr**
Phys. address associated with the given virtual address. Zero if phys. address is not available for the current process.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to

check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

 **status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

 **source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

 **IOHandle/dup**
Duplicated Driver-handle.

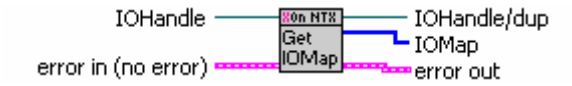
# 4.4 XOnNTX_GetPortIOMap

Returns the current Port-IO mapping state for the current process.

Use this function before using XOnNTX_EnablePortIO or XOnNTX_DisablePortIO to save the current mapping state. Use XOnNTX_SetPortIOMap after your direct port access to reset the mapping state.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOMap**
Port mapping state.

WARNING:
As this is an opaque data structure do not modify it's contents. Use the IOMap in calls to XOnNTX_SetPortIOMap only.

 **IOHandle/dup**
Duplicated Driver-handle.

 **error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

 **status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

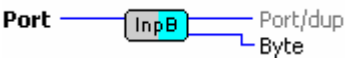
 **source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 4.5 XOnNTX_InpB

Reads a single byte value (u8) from the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane




---

### Controls and Indicators

**Port**
Port to read the value from.

**Byte**
Read value.

**Port/dup**
Duplicated port.

## 4.6 XOnNTX_InpD

Reads a single doubleword value (u32) from the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane


---

### Controls and Indicators

**Port**
Port to read the value from.

**DWord**
Read value.

**Port/dup**
Duplicated port.

# 4.7 XOnNTX_InpW

Reads a single word value (u16) from the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane


---

### Controls and Indicators

**Port**
Port to read the value from.

**Word**
Read value.

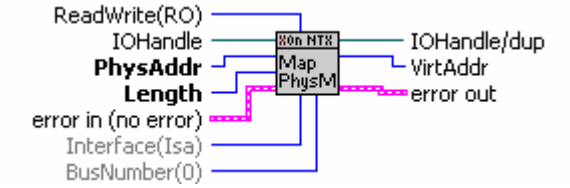
**Port/dup**
Duplicated port.

## 4.8 XOnNTX_MapPhysMemory

Maps a physical address range into a process specific virtual address range to allow reading/writing directly from/to the memory. You should unmap the memory with XOnNTX_UnmapPhysMemory after using the memory, not to block the virtual address range to long.
Use the utility functions to read/write from/to the virtual address range.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**Length**
Nr. of bytes to be mapped.

**PhysAddr**
Physical start address to be mapped.

**Interface(Isa)**
The bus-system over this physical address is accessable.

Mapping the computer-RAM the bus is not necessary. Mapping memory from a board it depends on the bus where board has been plugged in.

**BusNumber(0)**
Some computers do have more than one bus of the same type (PCI). In this case you have to select the correct busnr. Your PC-documentation should have information which slot belongs to what busnr.

**ReadWrite(RO)**
Allowed accesstype.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr**
Mapped virtual start address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

## 4.9 XOnNTX_OutB

Writes a single byte value (u8) to the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane


---

### Controls and Indicators

**Port**
Port to write the value to.

**Byte(0)**
Value to write.

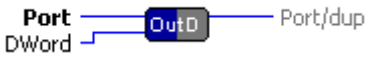
**Port/dup**
Duplicated port.

# 4.10 XOnNTX_OutD

Writes a single doubleword value (u32) to the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane




---

### Controls and Indicators

**Port**
Port to write the value to.

**DWord**
Value to write.

**Port/dup**
Duplicated port.

## 4.11 XOnNTX_OutW

Writes a single word value (u16) to the given port via fast direct port access.
WARNING: The given port MUST be in an allowed port-range.
Use XOnNTX_EnablePortIO to enable ports.

---

### Connector Pane


---

### Controls and Indicators

**Port**
Port to write the value to.

**Word**
Value to write.

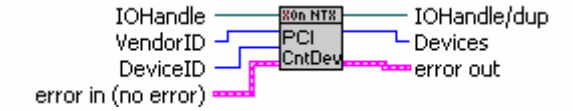
**Port/dup**
Duplicated port.

# 4.12 XOnNTX_PCICountDevice

Counts nr. of PCI-devices.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**VendorID**
Vendor-ID of the PCI-device to restrict counting for.

Use xFFFF not to restrict counting.

**DeviceID**
Device-ID to restrict the counting for.

Use xFFFF not to restrict counting on any Device-ID.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred

before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.


**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.


**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.


**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.


**IOHandle/dup**
Duplicated Driver-handle.

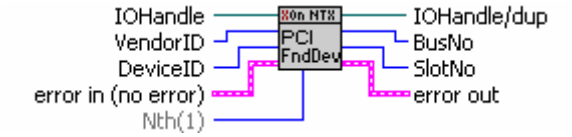

**Devices**
Nr. of devices found.

# 4.13 XOnNTX_PCIFindDevice

Find the Bus- and Slot-Number of the specified device.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**VendorID**
Vendor-ID of PCI-device to find.

Use xFFFF to find devices of any vendor.

**DeviceID**
Device-ID of PCI-device to find.

Use xFFFF to find any device.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Nth(1)**

Find the n'th [1..] PCI-device.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

**BusNo**
Busnr. the device has been found.

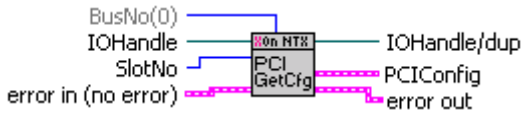
**SlotNo**

# 4.14 XOnNTX_PCIGetConfig

Get configuration of a specific PCI-device.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**BusNo(0)**
Busnr. of the CPI-device.

**SlotNo**
BIT-description

| | |
|---|---|
| **0..4** | Devicenumber |
| **5..7** | Functionnumber |
| **...** | Reserved |

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**PCICfg**
Reference notes for PCI configuration fields:

**ro** these field are read only. changes to these fields are ignored

**ro+** these field are intended to be read only and should be initialized by the system to their proper values. However, driver may change these settings.

All resources comsumed by a PCI device start as unitialized under NT. An uninitialized memory or I/O base address can be determined by checking it's corrisponding enabled bit in the PCI_COMMON_CONFIG.Command value.
An InterruptLine is unitialized if it contains the value of -1.

U16 **VendorID**
Vendor-ID // ro

U16 **DeviceID**
Device-Id // ro

U16 **Command**
Device Control

```
//
// Bit encodings for PCI_COMMON_CONFIG.Command
//
#define PCI_ENABLE_IO_SPACE                 0x0001
#define PCI_ENABLE_MEMORY_SPACE             0x0002
#define PCI_ENABLE_BUS_MASTER               0x0004
#define PCI_ENABLE_SPECIAL_CYCLES           0x0008
#define PCI_ENABLE_WRITE_AND_INVALIDATE     0x0010
#define PCI_ENABLE_VGA_COMPATIBLE_PALETTE   0x0020
#define PCI_ENABLE_PARITY                   0x0040  // (ro+)
#define PCI_ENABLE_WAIT_CYCLE               0x0080  // (ro+)
#define PCI_ENABLE_SERR                     0x0100  // (ro+)
#define PCI_ENABLE_FAST_BACK_TO_BACK        0x0200  // (ro)
```

U16 **Status**

```
//
// Bit encodings for PCI_COMMON_CONFIG.Status
//
#define PCI_STATUS_FAST_BACK_TO_BACK        0x0080  // (ro)
#define PCI_STATUS_DATA_PARITY_DETECTED     0x0100
#define PCI_STATUS_DEVSEL                   0x0600  // 2 bits  wide
#define PCI_STATUS_SIGNALED_TARGET_ABORT    0x0800
#define PCI_STATUS_RECEIVED_TARGET_ABORT    0x1000
#define PCI_STATUS_RECEIVED_MASTER_ABORT    0x2000
#define PCI_STATUS_SIGNALED_SYSTEM_ERROR    0x4000
#define PCI_STATUS_DETECTED_PARITY_ERROR    0x8000
```

U8 **RevID**

U8 **ProgIF**

U8 **SubCl**

U8 **BaseCl**

U8 **CacheLS**

U8 **LatTim**

U8 **HType**

```
//
// Bit encodings for  PCI_COMMON_CONFIG.HeaderType
//
#define PCI_MULTIFUNCTION                   0x80
#define PCI_DEVICE_TYPE                     0x00
#define PCI_BRIDGE_TYPE                     0x01
```

### BIST

### HEADER_TYPE_0

#### BaseAddr1

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

#### BaseAddr2

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

#### BaseAddr3

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

#### BaseAddr4

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr5**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr6**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**CIS**

**SubVenID**

**SubSysID**

**ROMBaseAddr**

// Bit encodes for ROMBaseAddr

x00000001 = PCI_ROMADDRESS_ENABLED

**Res1**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**Res2**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**IRQLn**

**IRQPin**

**MinGr**

**MaxLat**

**error out**

error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**

Duplicated Driver-handle.

**PCIConfig**
Reference notes for PCI configuration fields:

**ro** these field are read only. changes to these fields are ignored

**ro+** these field are intended to be read only and should be initialized by the system to their proper values. However, driver may change these settings.

All resources comsumed by a PCI device start as unitialized under NT. An uninitialized memory or I/O base address can be determined by checking it's corrisponding enabled bit in the PCI_COMMON_CONFIG.Command value.
An InterruptLine is unitialized if it contains the value of -1.

**VendorID**
Read only.
Identifies the manufacturer of the device.

**DeviceID**
Read only.
Identifies the particular device. This value is assigned by the manufacturer.

**Command**
Accesses the PCI device's control register. Writing a zero to this register renders the device logically disconnected from the PCI bus except for configuration access. Otherwise, the functionality of the register is device-dependent.
Bit encodings for PCI_COMMON_CONFIG.Command

```
//
// Bit encodings for PCI_COMMON_CONFIG.Command
//
#define PCI_ENABLE_IO_SPACE                 0x0001
#define PCI_ENABLE_MEMORY_SPACE             0x0002
#define PCI_ENABLE_BUS_MASTER               0x0004
#define PCI_ENABLE_SPECIAL_CYCLES           0x0008
#define PCI_ENABLE_WRITE_AND_INVALIDATE     0x0010
#define PCI_ENABLE_VGA_COMPATIBLE_PALETTE   0x0020
#define PCI_ENABLE_PARITY                   0x0040  // (ro+)
#define PCI_ENABLE_WAIT_CYCLE               0x0080  // (ro+)
#define PCI_ENABLE_SERR                     0x0100  // (ro+)
#define PCI_ENABLE_FAST_BACK_TO_BACK        0x0200  // (ro)
```

**Status**
Accesses the PCI device's status register. The functionality of this register is device-dependent.
Bit encodings for PCI_COMMON_CONFIG.Status

```
//
// Bit encodings for PCI_COMMON_CONFIG.Status
//
#define PCI_STATUS_FAST_BACK_TO_BACK        0x0080  // (ro)
#define PCI_STATUS_DATA_PARITY_DETECTED     0x0100
#define PCI_STATUS_DEVSEL                   0x0600  // 2 bits  wide
#define PCI_STATUS_SIGNALED_TARGET_ABORT    0x0800
#define PCI_STATUS_RECEIVED_TARGET_ABORT    0x1000
#define PCI_STATUS_RECEIVED_MASTER_ABORT    0x2000
#define PCI_STATUS_SIGNALED_SYSTEM_ERROR    0x4000
#define PCI_STATUS_DETECTED_PARITY_ERROR    0x8000
```

**RevID**
RevisionID
Specifies the revision level of the device described by the DeviceID member. This value is assigned by the manufacturer.

**ProgIF**
Identifies the register-level programming interface, if any, for the device, according to the PCI classification scheme.

**SubCl**
SubClass
Identifies the subtype, if any, of the device, according to the PCI classification scheme.

**BaseCl**
BaseClass
Identifies type of the device, according to the PCI classification scheme.

**CacheLS**
CacheLineSize
Contains the system cache line size in 32-bit units. This member is relevant only for PCI busmaster devices. The system determines this value during the boot process.

**LatTim**
LatencyTimer
Contains the value of the latency timer in units of PCI bus clocks. This member is relevant only for PCI busmaster devices. The system determines this value during the boot process.

**HType**
HeaderType
The system ORs the value of this member with PCI_MULTIFUNCTION, if appropriate to the device. The value of this member indicates the PCI_HEADER_TYPE_0 layout that follows.
Bit encodings for PCI_COMMON_CONFIG.HeaderType

| | |
|---|---|
| x80 | PCI_MULTIFUNCTION |
| x00 | PCI_DEVICE_TYPE |
| x01 | PCI_BRIDGE_TYPE |

**BIST**
Built In Self Test
Zero indicates that the device does not support built-in self test. Otherwise, the device supports built-in self test according to the PCI standard.

**HEADER_TYPE_0**

**BaseAddr1**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr2**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr3**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr4**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr5**

Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

 **BaseAddr6**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

 **CIS**

 **SubVenID**

 **SubSysID**

 **ROMBaseAddr**
// Bit encodes for ROMBaseAddr

x00000001 = PCI_ROMADDRESS_ENABLED

 **Res1**

 **Res2**

 **IRQLn**
Interruptline

 **IRQPin**
Interruptpin

 **MinGr**
MinimumGrant

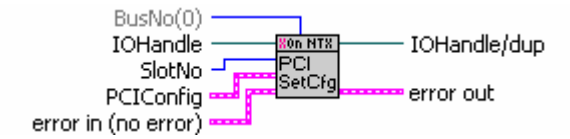
 **MaxLat**
MaximumLatency

 **DevSpecific**
Contains any device-specific initialization information that is available.

## 4.15 XOnNTX_PCISetConfig

Set configuration of a specific PCI-device.

### Connector Pane




### Controls and Indicators

**IOHandle**
Driver-handle.

**BusNo(0)**
Busnr. of the CPI-device.

**SlotNo**
BIT-description

| | |
|---|---|
| **0..4** | Devicenumber |
| **5..7** | Functionnumber |
| **...** | Reserved |

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**PCIConfig**

Reference notes for PCI configuration fields:

**ro** these field are read only. changes to these fields are ignored

**ro+** these field are intended to be read only and should be initialized by the system to their proper values. However, driver may change these settings.

All resources comsumed by a PCI device start as unitialized under NT. An uninitialized memory or I/O base address can be determined by checking it's corrisponding enabled bit in the PCI_COMMON_CONFIG.Command value.
An InterruptLine is unitialized if it contains the value of -1.

### U16 VendorID

Read only.
Identifies the manufacturer of the device.

### U16 DeviceID

Read only.
Identifies the particular device. This value is assigned by the manufacturer.

### U16 Command

Accesses the PCI device's control register. Writing a zero to this register renders the device logically disconnected from the PCI bus except for configuration access. Otherwise, the functionality of the register is device-dependent.
Bit encodings for PCI_COMMON_CONFIG.Command

```
//
// Bit encodings for PCI_COMMON_CONFIG.Command
//
#define PCI_ENABLE_IO_SPACE                 0x0001
#define PCI_ENABLE_MEMORY_SPACE             0x0002
#define PCI_ENABLE_BUS_MASTER               0x0004
#define PCI_ENABLE_SPECIAL_CYCLES           0x0008
#define PCI_ENABLE_WRITE_AND_INVALIDATE     0x0010
#define PCI_ENABLE_VGA_COMPATIBLE_PALETTE   0x0020
#define PCI_ENABLE_PARITY                   0x0040  // (ro+)
#define PCI_ENABLE_WAIT_CYCLE               0x0080  // (ro+)
#define PCI_ENABLE_SERR                     0x0100  // (ro+)
#define PCI_ENABLE_FAST_BACK_TO_BACK        0x0200  // (ro)
```

### U16 Status

Accesses the PCI device's status register. The functionality of this register is device-dependent.
Bit encodings for PCI_COMMON_CONFIG.Status

```
//
// Bit encodings for PCI_COMMON_CONFIG.Status
//
#define PCI_STATUS_FAST_BACK_TO_BACK        0x0080  // (ro)
#define PCI_STATUS_DATA_PARITY_DETECTED     0x0100
#define PCI_STATUS_DEVSEL                   0x0600  // 2 bits  wide
#define PCI_STATUS_SIGNALED_TARGET_ABORT    0x0800
#define PCI_STATUS_RECEIVED_TARGET_ABORT    0x1000
#define PCI_STATUS_RECEIVED_MASTER_ABORT    0x2000
#define PCI_STATUS_SIGNALED_SYSTEM_ERROR    0x4000
#define PCI_STATUS_DETECTED_PARITY_ERROR    0x8000
```

### U8 RevID

RevisionID

Specifies the revision level of the device described by the DeviceID member. This value is assigned by the manufacturer.

**ProgIF**
Identifies the register-level programming interface, if any, for the device, according to the PCI classification scheme.

**SubCl**
SubClass
Identifies the subtype, if any, of the device, according to the PCI classification scheme.

**BaseCl**
BaseClass
Identifies type of the device, according to the PCI classification scheme.

**CacheLS**
CacheLineSize
Contains the system cache line size in 32-bit units. This member is relevant only for PCI busmaster devices. The system determines this value during the boot process.

**LatTim**
LatencyTimer
Contains the value of the latency timer in units of PCI bus clocks. This member is relevant only for PCI busmaster devices. The system determines this value during the boot process.

**HType**
HeaderType
The system ORs the value of this member with PCI_MULTIFUNCTION, if appropriate to the device. The value of this member indicates the PCI_HEADER_TYPE_0 layout that follows.
Bit encodings for PCI_COMMON_CONFIG.HeaderType

| | |
|---|---|
| x80 | PCI_MULTIFUNCTION |
| x00 | PCI_DEVICE_TYPE |
| x01 | PCI_BRIDGE_TYPE |

**BIST**
Built In Self Test
Zero indicates that the device does not support built-in self test. Otherwise, the device supports built-in self test according to the PCI standard.

**HEADER_TYPE_0**

**BaseAddr1**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

### BaseAddr2
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

### BaseAddr3
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

### BaseAddr4
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

### BaseAddr5
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

### BaseAddr6
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**CIS**

**SubVenID**

**SubSysID**

**ROMBaseAddr**
// Bit encodes for ROMBaseAddr

x00000001 = PCI_ROMADDRESS_ENABLED

**Res1**

**Res2**

**IRQLn**
Interruptline

**IRQPin**
Interruptpin

**MinGr**
MinimumGrant

**MaxLat**
MaximumLatency

**DevSpecific**
Contains any device-specific initialization information that is available.

**PCICfg**
Reference notes for PCI configuration fields:

**ro** these field are read only. changes to these fields are ignored

**ro+** these field are intended to be read only and should be initialized by the system to their proper values. However, driver may change these settings.

All resources comsumed by a PCI device start as unitialized under NT. An uninitialized memory or I/O base address can be determined by checking it's corrisponding enabled bit in the

PCI_COMMON_CONFIG.Command value.
An InterruptLine is unitialized if it contains the value of -1.

**U16 VendorID**
Vendor-ID // ro

**U16 DeviceID**
Device-Id // ro

**U16 Command**
Device Control

```
//
// Bit encodings for PCI_COMMON_CONFIG.Command
//
#define PCI_ENABLE_IO_SPACE                  0x0001
#define PCI_ENABLE_MEMORY_SPACE              0x0002
#define PCI_ENABLE_BUS_MASTER                0x0004
#define PCI_ENABLE_SPECIAL_CYCLES            0x0008
#define PCI_ENABLE_WRITE_AND_INVALIDATE      0x0010
#define PCI_ENABLE_VGA_COMPATIBLE_PALETTE    0x0020
#define PCI_ENABLE_PARITY                    0x0040  // (ro+)
#define PCI_ENABLE_WAIT_CYCLE                0x0080  // (ro+)
#define PCI_ENABLE_SERR                      0x0100  // (ro+)
#define PCI_ENABLE_FAST_BACK_TO_BACK         0x0200  // (ro)
```

**U16 Status**

```
//
// Bit encodings for PCI_COMMON_CONFIG.Status
//
#define PCI_STATUS_FAST_BACK_TO_BACK         0x0080  // (ro)
#define PCI_STATUS_DATA_PARITY_DETECTED      0x0100
#define PCI_STATUS_DEVSEL                    0x0600  // 2 bits  wide
#define PCI_STATUS_SIGNALED_TARGET_ABORT     0x0800
#define PCI_STATUS_RECEIVED_TARGET_ABORT     0x1000
#define PCI_STATUS_RECEIVED_MASTER_ABORT     0x2000
#define PCI_STATUS_SIGNALED_SYSTEM_ERROR     0x4000
#define PCI_STATUS_DETECTED_PARITY_ERROR     0x8000
```

**U8 RevID**

**U8 ProgIF**

**U8 SubCl**

**U8 BaseCl**

**U8 CacheLS**

**U8 LatTim**

**U8 HType**

```
//
// Bit encodings for  PCI_COMMON_CONFIG.HeaderType
//
#define PCI_MULTIFUNCTION                    0x80
#define PCI_DEVICE_TYPE                      0x00
#define PCI_BRIDGE_TYPE                      0x01
```

**U8 BIST**

**HEADER_TYPE_0**

**BaseAddr1**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr2**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr3**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr4**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr5**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**BaseAddr6**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**CIS**

**SubVenID**

**SubSysID**

**ROMBaseAddr**
// Bit encodes for ROMBaseAddr

x00000001 = PCI_ROMADDRESS_ENABLED

**Res1**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**Res2**
Bit encodes for PCI_COMMON_CONFIG.BaseAddrx

| | | |
|---|---|---|
| x00000001 | PCI_ADDRESS_IO_SPACE | ro |
| x00000006 | PCI_ADDRESS_MEMORY_TYPE_MASK | ro |
| x00000008 | PCI_ADDRESS_MEMORY_PREFETCHABLE | ro |

| | |
|---|---|
| 0 | PCI_TYPE_32BIT |
| 2 | PCI_TYPE_20BIT |
| 4 | PCI_TYPE_64BIT |

**IRQLn**

**IRQPin**

**MinGr**

**MaxLat**

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

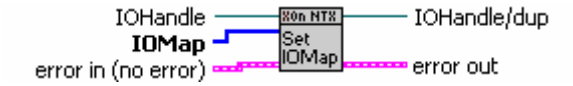
# 4.16 XOnNTX_SetPortIOMap

Resets the Port-IO mapping state of the current process.

Use this function as counterpart to XOnNTX_GetPortIOMap to reset the mapping state. This will revoke all state changes from calls to XOnNTX_EnablePortIO or XOnNTX_DisablePortIO.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**IOMap**
Port mapping state to set.

WARNING:
Use IOMap's returned from calls to XOnNTX_GetPortIOMap only.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

**error out**

error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

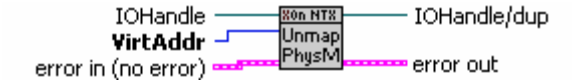
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 4.17 XOnNTX_UnmapPhysMemory

Unmaps the physical address range of the given virtual address range. The given start address (VirtAdr) MUST be the result of a mapping operation via XOnNTX_MapPhysMemory.
After unmapping, the virtual address range is no longer allowed to be read or written.

---

### Connector Pane




---

### Controls and Indicators

**IOHandle**
Driver-handle.

**VirtAddr**
Virtual start address that has been mapped via XOnNTX_MapPhysMemory.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

# 5 XOnNTX_ShareM_VIs

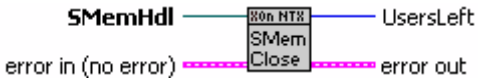
## 5.1 XOnNTX_CloseSharedMemory

Closes a shared memory object handle. Does not remove the shared memory object if other users of the shared memory exist.

Returns the nr. of users left.

---

### Connector Pane




---

### Controls and Indicators

**SMemHdl**
Handle of an opened shared memory object.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**UsersLeft**
Nr. of users left after closing.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to

check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

 **status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

 **source**

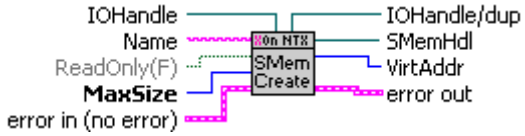
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 5.2 XOnNTX_CreateSharedMemory

Creates a new or opens an existing shared memory object (named or unnamed).
Returns a handle to the shared memory object and a virtual address pointer to the shared memory.
Use the utility functions to read/write from/to the shared memory.

---

### Connector Pane




---

### Controls and Indicators

**Name**
Name of shared-memory object (optional).
Named shared memory objects can be opened from other processes in the system.

**MaxSize**
Max. size of the shared memory in bytes.

**ReadOnly(F)**
True : Create/Open for reading from the shared memory only
False: Create/Open for reading and writing.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle**
Driver-handle.

**SMemHdl**
Handle to the created/opened shared memory object.

**VirtAddr**
Virtual address of the shared memory.

Use this address with the utility functions to access the memory.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

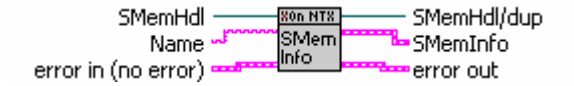
# 5.3 XOnNTX_InfoSharedMemory

Returns information on an existing shared memory object.

You may give an open shared memory handle or the name of a shared memory object to ask for information.

---

### Connector Pane




---

### Controls and Indicators

**SMemHdl**

**Name**
Name of shared-memory object (optional).

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**SMemHdl/dup**

**SMemInfo**
Structure with information on shared memory objects.

- **Name**
  Name of shared memory object. Empty if unnamed.

**Users**
Nr. of users of the shared memory object.

**Size**
Max. nr. of bytes the shared memory can hold.

**VirtAddr**
Virtual address of the shared memory.

May be zero if shared memory object exists but has not been opened from the current process.

**ReadOnly**
True : Shared memory object is opened for reading only
False: Shared memory object can be read or written.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

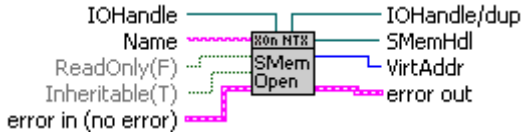
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 5.4 XOnNTX_OpenSharedMemory

Opens an existing shared memory object (named or unnamed).
Returns a handle to the shared memory object and a virtual address pointer to the shared memory.
Use the utility functions to read/write from/to the shared memory.

---

### Connector Pane




---

### Controls and Indicators

**Name**
Name of shared-memory object (optional).
Named shared memory objects can be opened from other processes in the system.

**ReadOnly(F)**
True : Create/Open for reading from the shared memory only
False: Create/Open for reading and writing.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle**
Driver-handle.

**Inheritable(T)**
True : The shared memory handle can be inherited by newly created processes.
False: The shared memory object handle cannot be inherited.

**SMemHdl**
Handle to the opened shared memory object.

**VirtAddr**
Virtual address of the shared memory.

Use this address with the utility functions to access the memory.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

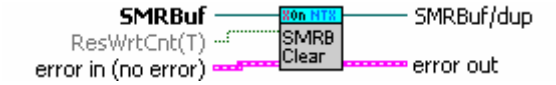
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

# 5.5 XOnNTX_SMRBufClear

Clears the ring-buffer.

### Connector Pane




### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**ResWrtCnt(T)**
True : Reset write-counter to zero.
False: Do not reset the write-counter

The write-counter counts the nr. of bytes ever written to the buffer.

**SMRBuf/dup**
Duplicated ring-buffer handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error,

if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

 **status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

 **code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

 **source**

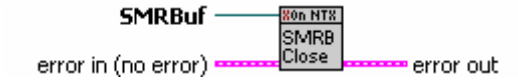
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 5.6 XOnNTX_SMRBufClose

Closes an open ring-buffer.
Closing a ring-buffer removes the buffer only if it is the last process that uses the buffer. If not, the buffer-contents will be kept and only administration structures are cleared.

---

### Connector Pane




---

### Controls and Indicators

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

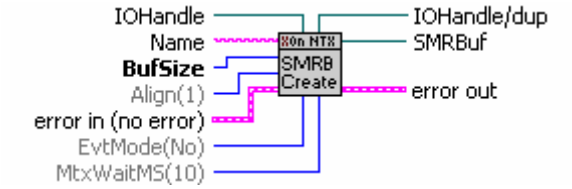
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 5.7 XOnNTX_SMRBufCreate

Creates a new or opens an existing ring-buffer based on shared memory. Returns a handle to the ring-buffer.

---

### Connector Pane




---

### Controls and Indicators

**Name**
Name of ring-buffer (optional).
Named ring-buffers can be opened from other processes in the system.

**BufSize**
Size in bytes the ring-buffer can hold.
The size will be internally aligned according to the 'Align' parameter.

**EvtMode(No)**
Eventmode.
On Eventmodes other than 'NoEvt' the ring-buffer administration creates a named Read- and/or Write-Eventobject that may be used to synchronize read- and write-operations.
A read-event will be signaled if any ring-buffer user reads data or clears the ring-buffer.
A write-event will be signaled on write-operations.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle**
Driver-handle.

**Align(1)**
Alignment of the ring-buffer.
The ring-buffer administration forces sizes on read-/write-operations to be aligned according to this parameter.

Example: If buffer should hold only 32-BIT integers set 'Align' to 4
(the byte size of int32)

**MtxWaitMS(10)**
Mutexwait Timeout (optional).
The buffer administration always creates a named mutex-object for a ring-buffer. It is used to synchronize the access to internal structures for parallel processes. While one process is reading or writing to the buffer other processes wait on this mutex-object.
The 'MtxWaitMS' parameter sets the timeout value in milliseconds for this wait-operation.
If a wait timed out on a read or write, the function will return an error.

**SMRBuf**
Handle of the new or opened shared memory based ring-buffer.
Use this handle with other XOnNTX_SMRB.. functions.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

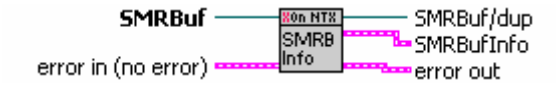
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle/dup**
Duplicated Driver-handle.

# 5.8 XOnNTX_SMRBufInfo

Returns informations on a ring-buffer.

### Connector Pane




### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**SMRBuf/dup**
Duplicated ring-buffer handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**SMRBufInfo**
Ring-buffer info structure.

**Align**
Current buffer alignment in bytes.

**WrtCount**
Nr. of bytes ever written to the ring-buffer.

**BufSize**
Max. nr. of bytes the buffer can hold.

**Overrun**
Nr. of overrun-bytes.

**ReadAvail**
Nr. of bytes currently available for read-operations.

**WriteAvail**
Nr. of bytes currently free for write-operations.

**SMemHdl**
Handle of the shared-memory object the ring-buffer is based on.

WARNING:
You should NOT write directly to the shared memory.This destroys administration structures and will lead to errors and/or system-crashes.

**hMutex**
Handle of the mutex-object internally used to synchronize buffer-access.

**hReadEvt**
Handle of the read-event-object that will be signaled on read-operations (may be 0 if not created).
WARNING: Use this handle only for wait-operations.

**hWriteEvt**
Handle of the write-event-object that will be signaled on write-operations (may be 0 if not created).
WARNING: Use this handle only for wait-operations.

**Mutex**
Name of mutex object.

**ReadEvt**
Name of read-event-object. May be empty if not created.

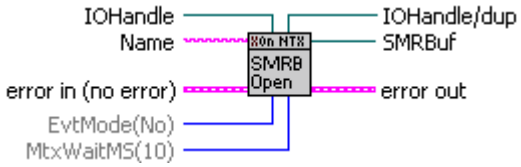
**WriteEvt**
Name of write-event-object. May be empty if not created.

## 5.9 XOnNTX_SMRBufOpen

Opens an existing ring-buffer (named or unnamed).
Returns an error if ring-buffer could not be opened (does not exist).

---

### Connector Pane




---

### Controls and Indicators

**Name**
Name of existing ring-buffer (optional).
Unnamed ring-buffers must be created by the same process to be opened by this functions.
Named ring-buffers can be created by other processes in the system.

**EvtMode(No)**
Eventmode.
On Eventmodes other than 'NoEvt' the ring-buffer administration creates a named Read- and/or Write-Eventobject that may be used to synchronize read- and write-operations.
A read-event will be signaled if any ring-buffer user reads data or clears the ring-buffer.
A write-event will be signaled on write-operations.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**IOHandle**

Driver-handle.

**MtxWaitMS(10)**
Mutexwait Timeout (optional).
The buffer administration always creates a named mutex-object for a ring-buffer. It is used to synchronize the access to internal structures for parallel processes. While one process is reading or writing to the buffer other processes wait on this mutex-object.
The 'MtxWaitMS' parameter sets the timeout value in milliseconds for this wait-operation.
If a wait timed out on a read or write, the function will return an error.

**SMRBuf**
Handle of the opened shared memory based ring-buffer.
Use this handle with other XOnNTX_SMRB.. functions.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

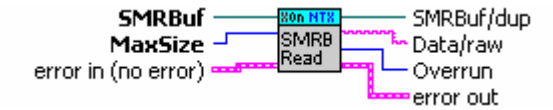
**IOHandle/dup**
Duplicated Driver-handle.

## 5.10 XOnNTX_SMRBufRead

Reads up to 'MaxSize' bytes from the ring-buffer.
The function does not wait until enough data are in the buffer. If less then 'MaxSize' bytes are available only the entire buffer contents will be returned.

---

### Connector Pane




---

### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**MaxSize**
Max. nr. of bytes to be read from the ring-buffer.

**SMRBuf/dup**
Duplicated ring-buffer handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error,

if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.


**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.


**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.


**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.


**Data/raw**
Read data in raw-format.

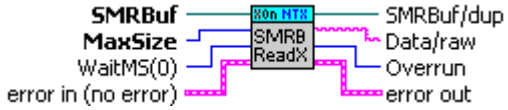

**Overrun**
Nr. of overrun-bytes since last read operation.

## 5.11 XOnNTX_SMRBufReadEx

Reads up to 'MaxSize' bytes from the ring-buffer.
The function waits upto 'WaitMS' milliseconds until enough data is in the buffer. If the wait timed out only the entire buffer-contents will be returned.
While waiting, the function allowes write-operations from other processes (threads). This allowes to read more data as the ring-buffer can hold.

---

### Connector Pane




---

### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**WaitMS(0)**
Max. nr. of milliseconds to wait until enough data are in the buffer.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**MaxSize**
Max. nr. of bytes to be read from the ring-buffer.

**SMRBuf/dup**

Duplicated ring-buffer handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Data/raw**
Read data in raw-format.

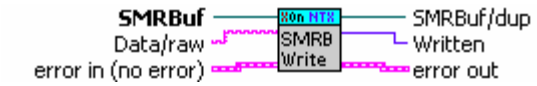
**Overrun**
Nr. of overrun-bytes since last read operation.

## 5.12 XOnNTX_SMRBufWrite

Writes (appends) data into the ring-buffer.
The function will overwrite existing data if not enough free space is available. The function does not write more than the max. size the buffer can hold.

---

### Connector Pane




---

### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Data/raw**
Data in raw format to write.

**SMRBuf/dup**
Duplicated ring-buffer handle.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display

the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

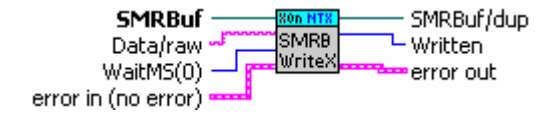
**Written**
Nr. of bytes written.

## 5.13 XOnNTX_SMRBufWriteEx

Writes (appends) data into the ring-buffer.
The function waites up to 'WaitMS' milliseconds until enough space is available to write the data. While waiting, the function allowes read-operations from other processes (threads). This allowes to write more data than the ring-buffer can hold.

---

### Connector Pane




---

### Controls and Indicators

**SMRBuf**
Handle of a ring-buffer.
XOnNTX_SMRBufCreate or XOnNTX_SMRBufOpen returns this handle.

**WaitMS(0)**
Max. nr. of milliseconds to wait until enough free space is available.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Data/raw**
Data in raw format to write.

**SMRBuf/dup**
Duplicated ring-buffer handle.

**error out**

error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Written**

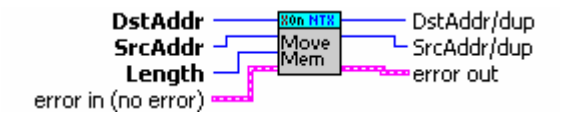
Nr. of bytes written.

# 6 XOnNTX_Utility_VIs

## 6.1 XOnNTX_MoveMemory

---

**Connector Pane**




---

**Controls and Indicators**

**DstAddr**

**SrcAddr**

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**Length**

**DstAddr/dup**

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display

the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.


**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.


**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.


**source**

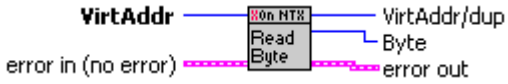
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.


**SrcAddr/dup**

## 6.2 XOnNTX_ReadByte

Read a byte (u8) from given virtual address.

### Connector Pane




### Controls and Indicators

**VirtAddr**
Virtual start address.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**Byte**
Read value.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

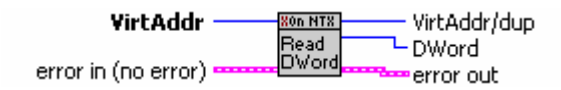
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 6.3 XOnNTX_ReadDWord

Read a doubleword (u32) from given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**DWord**
Read value.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

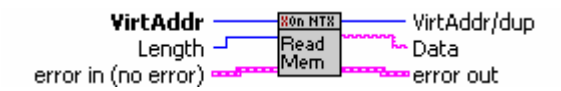
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 6.4 XOnNTX_ReadMemory

Read 'Length' bytes starting at given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**Length**
Nr. of bytes to read.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated start address.

**Data**
Read data.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display

the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

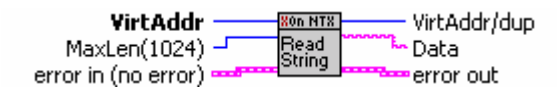
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 6.5 XOnNTX_ReadString

Read a string of max. length 'MaxLen' starting at given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**MaxLen(1024)**
Max. stringlength.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**Data**
Read string.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display

the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**

status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**

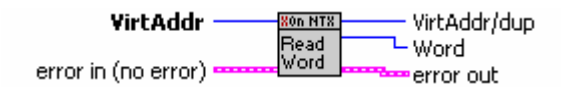
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 6.6 XOnNTX_ReadWord

Read a word (u16) from given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**Word**
Read value.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

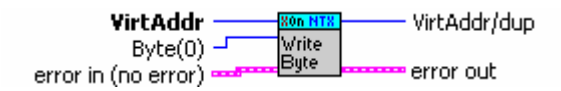
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

## 6.7 XOnNTX_WriteByte

Write a byte (u8) to the given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**Byte(0)**
Value to write.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

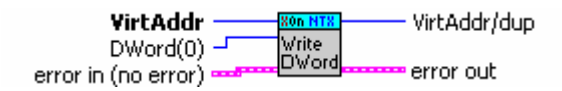
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 6.8 XOnNTX_WriteDWord

Write a doubleword (u32) to the given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**DWord(0)**
Value to write.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

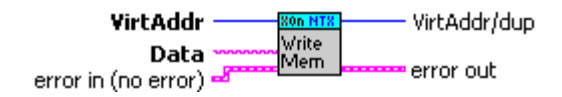
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 6.9 XOnNTX_WriteMemory

Writes data to the given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**Data**
Data to write.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated start address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

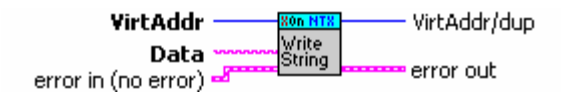
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 6.10 XOnNTX_WriteString

Write string 'Data' to the given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**Data**
String to write.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated start address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

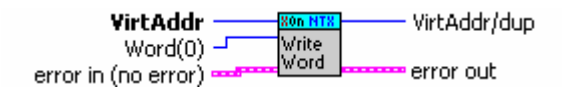
**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 6.11 XOnNTX_WriteWord

Write a word (u16) to the given virtual address.

---

### Connector Pane




---

### Controls and Indicators

**VirtAddr**
Virtual start address.

**Word(0)**
Value to write.

**error in (no error)**
error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status**
  status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code**
  code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source**
  source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

**VirtAddr/dup**
Duplicated virtual address.

**error out**
error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

**status**
status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

**code**
code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

**source**
source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

# 7 Constants and Types

## 7.1 XOnNTX_HardIO

### 7.1.1 PCI_COMMON_CONFIG

---

**Connector Pane**

### 7.1.2 PCI_CONFIG

**Connector Pane**

## 7.2 XOnNTX_BaseIO

### 7.2.1 IOHANDLE

---

**Connector Pane**

## 7.2.2 XONNTX_DIAGINFO

**Connector Pane**

# 7.3 XOnNTX_ShareM

## 7.3.1 SMEMHDL

Handle of shared memory.

---

**Connector Pane**

## 7.3.2 SMEMINFO

Infostructure for inforamtion on shared memory objects.

**Connector Pane**

### 7.3.3 SMRBUFHDL

Handle of a shared memory based ring-buffer.

---

**Connector Pane**

## 7.3.4 SMRBUFINFO

Ring-Buffer info.

### Connector Pane

# 8 Contact


**XOn Software GmbH**
Luisenstraße 43
80333 München
Telefon: +49 (0)89/542716-0
Telefax: +49 (0)89/542716-78
E-mail: info@xon.de
Internet: http://www.xon.de

**Your partner for ...**
LabVIEW, LabWindows/CVI, Microsoft Visual C++, National Instruments Hardware, GPIB, CAN, Profibus, RS232, RS 485...

# Index

## - C -



## - D -



## - E -



## - G -



## - I -



## - M -



## - N -

# - O -



# - P -



# - R -



# - S -



# - U -



# - V -



# - W -



# - X -