

# **Custom Interface Driver Help**

© 2009 Kepware Technologies

# Table of Contents

<b>1</b>	<b>Getting Started.....</b>	<b>3</b>
	Help Contents.....	3
	Overview .....	3
<b>2</b>	<b>Channel Setup.....</b>	<b>3</b>
	Channel Setup.....	3
	Channel Configuration.....	3
	Support Information.....	5
<b>3</b>	<b>Device Setup.....</b>	<b>6</b>
	Device Setup .....	6
	Device Configuration.....	6
	Settings .....	6
<b>4</b>	<b>Data Types Description.....</b>	<b>7</b>
	Data Types Description.....	7
<b>5</b>	<b>Address Descriptions.....</b>	<b>8</b>
	Address Descriptions.....	8
<b>6</b>	<b>Automatic Device/Tag Generation.....</b>	<b>9</b>
	Automatic Device/Tag Generation.....	9
<b>7</b>	<b>Error Descriptions.....</b>	<b>9</b>
	Error Descriptions.....	9
	<b>General Errors.....</b>	<b>10</b>
	General Errors.....	10
	Cannot open Shared Memory file associated with Configuration '<configuration name>'.....	10
	<b>Read Errors .....</b>	<b>10</b>
	Read Errors.....	10
	Unable to read from register '<register offset>' on device '<device name>'. Register is not configured for read access.....	10
	Unable to read from register '<register offset>' on device '<device name>'. Register value type is not configured for read data.....	11
	Unable to read to register '<register offset>' on device '<device name>'. Register corrupted.....	11
	Unable to read from register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'.....	11
	<b>Write Errors .....</b>	<b>12</b>
	Write Errors.....	12
	Unable to write to register '<register offset>' on device '<device name>'. Register is not configured for write access.....	12
	Unable to write to register '<register offset>' on device '<device name>'. Register value type is not configured for write data.....	12
	Unable to write to register '<register offset>' on device '<device name>'. Register corrupted.....	13
	Unable to write to register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'.....	13
<b>8</b>	<b>Developer Information.....</b>	<b>13</b>
	Developer Information.....	13
	<b>CIDA Overview.....</b>	<b>13</b>
	CIDA Overview.....	13
	CIDA Requirements.....	14
	<b>Shared Memory Interface.....</b>	<b>16</b>
	Shared Memory Interface.....	16

REGISTER Structure.....	16
DATA Structure.....	16
VALUE Structure.....	17
STRINGARRAY Structure.....	18
<b>Reference Implementation.....</b>	<b>18</b>
Reference Implementation.....	18
Reference Implementation Architecture.....	19
CID/CIDA Reference Implementation Demonstration.....	20
<b>Troubleshooting.....</b>	<b>24</b>
Channel Diagnostics.....	24

<b>Index</b>	<b>29</b>
--------------	-----------

## Custom Interface Driver Help

---

Help version 1.018

### CONTENTS

#### [Overview](#)

What is the Custom Interface Driver?

#### [Channel Setup](#)

How do I configure custom channel properties for this driver?

#### [Device Setup](#)

How do I configure a device for use with this driver?

#### [Data Types Description](#)

What data types does this driver support?

#### [Address Description](#)

How do I address a data location on a Custom Interface Driver device?

#### [Automatic Device/Tag Generation](#)

How can I easily configure tags for the Custom Interface Driver?

#### [Error Descriptions](#)

What error messages does the Custom Interface Driver produce?

#### [Developer Information](#)

As a developer, where can I find supplemental in-depth information and examples of implementation?

### Overview

---

The Custom Interface Driver (CID) provides OPC and native connectivity for third party custom driver data without using a toolkit. The custom drivers are called Custom Interface Driver Applications (CIDA) and they interface with the CID through the inter-process communication method called shared memory. The CIDA is responsible for creating the shared memory file as well as an XML configuration file that fully defines the data mapped in shared memory. The configuration file is then imported into the CID in order for automatic device/tag generation within the OPC server.

Communication via shared memory is achieved through standardized structures mapped onto shared memory. For more information on CID and CIDA data flow, refer to [CIDA Overview](#).

**Note:** The Custom Interface Driver help file is meant to supplement the help file provided with the CID Application. For more information on the Reference Implementation provided with the driver, refer to [Reference Implementation](#).

### Channel Setup

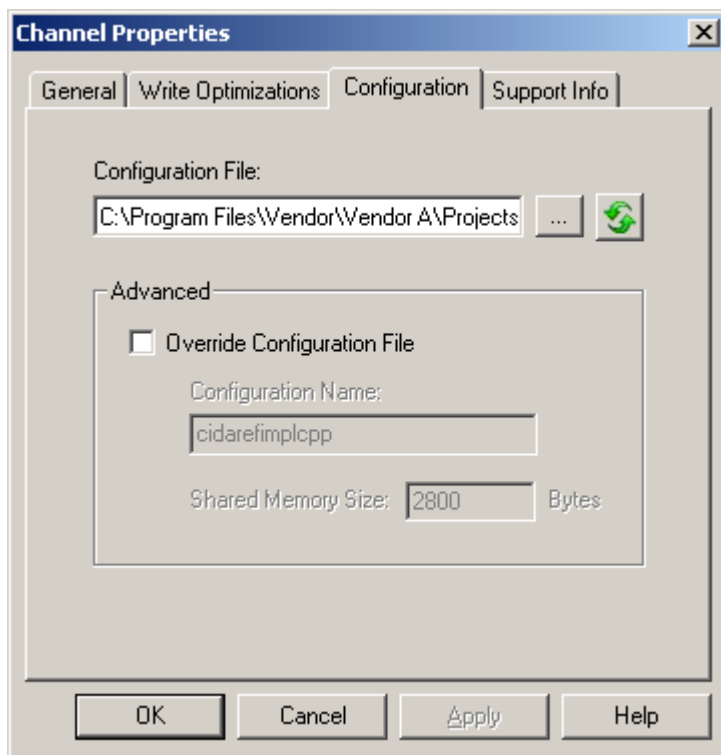
---

A channel represents a single CID Application Configuration. Each CID Application must provide a configuration file that specifies the configuration name used to associate with a Shared Memory file (as well as a list of the devices and tags corresponding to that configuration). The CID can support multiple configurations simultaneously; however, each configuration must contain a unique configuration name. For more information, refer to [Configuration File](#).

### Channel Configuration

---

The Configuration dialog is used to specify the location and behavior of the configuration file, as well as any Advanced settings.



### Configuration File

The configuration file is an xml file that contains the name and size of the shared memory file, as well as its associated devices and appropriate tag definitions. This parameter is used to specify its location. To access the **Open File** dialog, click the browse button (...) and then set the filter to "\*.xml." Select the desired file and then click **Open**. It is imperative that the configuration name within the file be unique in order to prevent one channel from interfering with another channel's access to a register. The file will be checked for uniqueness upon being set and prior to automatic device/tag generation.

**Caution:** Users must regenerate the configuration file whenever any changes have been made to it. For more information, refer to "Regenerate Button" below.

**Important:** When running in System Service Mode, the configuration file must be located in a folder that is accessible to a System Service in order for it to be loaded by the Runtime. For example, when running as a System Service, a file residing in a network drive that requires authentication will cause the loading to fail. For more information on System Service Mode, refer to the OPC Server's help file.

### Regenerate Button

The **Regenerate** button is used to manually start automatic device/tag generation. The button appears as shown below. For more information, refer to [Automatic Device/Tag Generation](#).



### Advanced

The Advanced settings should only be used by CID Application Developers. Descriptions of the parameters are as follows.

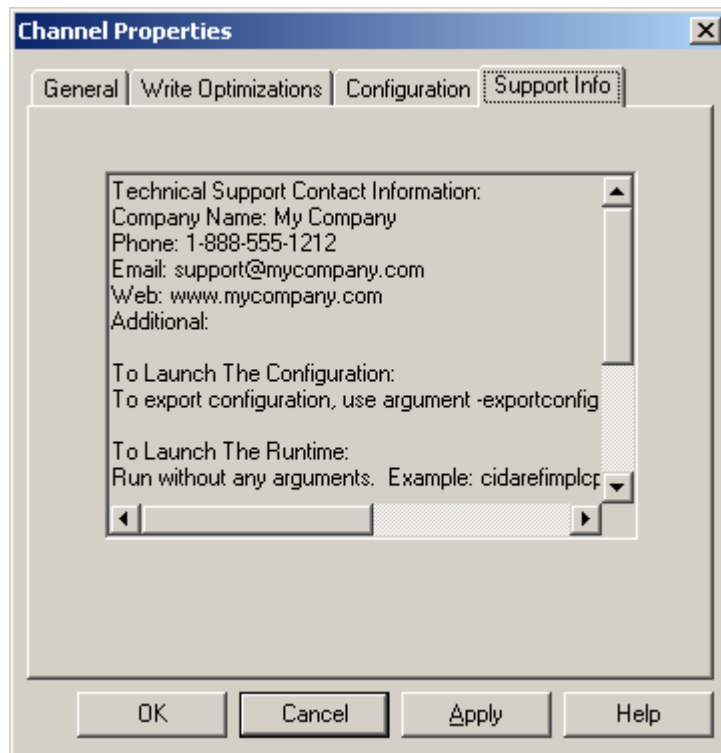
- **Override Configuration File:** This parameter is used to override the configuration name and shared memory size as previously defined by the configuration file. In the absence of a configuration file, this parameter may be used to manually define the configuration name and shared memory size.
- **Configuration Name:** This parameter contains the configuration name that the channel uses in accessing shared memory. The configuration name must be unique across channels. It is imperative that the configuration name be unique in order to prevent one channel from interfering with another channel's access to a register. The driver will automatically check for uniqueness and warn the user/override when necessary.

- **Shared Memory Size \_\_\_\_ Bytes:** This parameter contains the shared memory size that the channel uses in tag address validation. When the Override Configuration File setting is unchecked, this property will hold the value last imported during automatic device/tag generation, despite being disabled. When the Override Configuration File setting is checked, this property will become enabled and modifiable. The range is from 0-2,147,483,648. The default value is 0.

**Note:** If the Override Configuration File parameter is unchecked after either the configuration name or shared memory size settings have been changed, the settings will return to their defaults (which are the values last loaded from the configuration file).

## Support Information

The following dialog contains support information for the supporting party. The information presented may vary, depending on what they have chosen to include.



Descriptions of the parameters are as follows.

1. **Technical Support Contact Information:** This text includes the supporting party's company name, phone, email address, web address and any additional information that they wish to include.
2. **To Launch the Configuration:** This text includes brief instructions on how to start the supporting party's Configuration component.
3. **To Launch the Runtime:** This text includes brief instructions on how to start the supporting party's Runtime component.
4. **To Launch the Help:** This text includes brief instructions on how to access the supporting party's help documentation.
5. **Additional:** This text includes optional miscellaneous information.

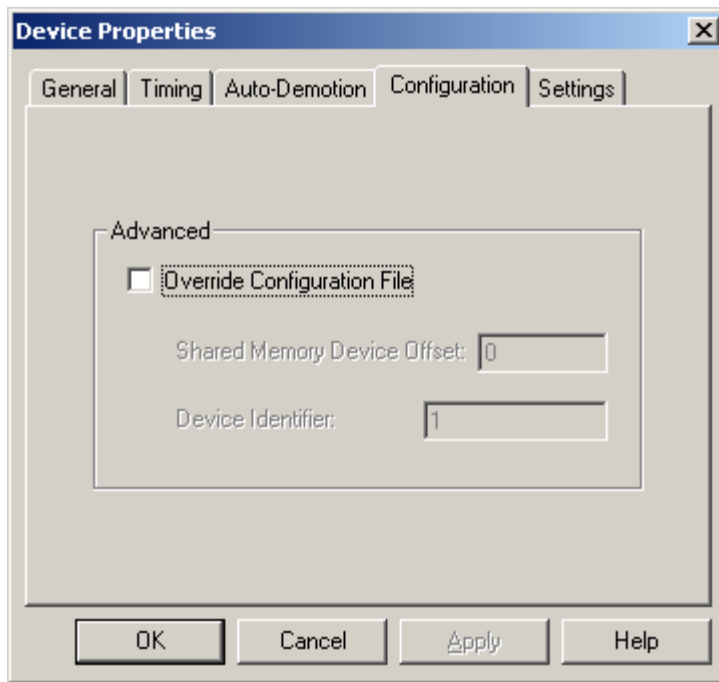
## Device Setup

A device is a logical collection of tags that corresponds to the physical device/data provider being polled by the CID Application.

## Device Configuration

### Advanced

The Advanced settings should only be used by CID Application Developers.



Descriptions of the parameters are as follows.

- **Override Configuration File:** This parameter is used to override the shared memory device offset and device identifier as previously defined by the configuration file. In the absence of a configuration file, this parameter may be used to manually define the shared memory device offset and device identifier.
- **Shared Memory Device Offset:** This parameter is used to define the starting byte offset into the shared memory map for the given device. All register offsets (i.e., tag addresses) are relative to the shared memory device offset. If the configuration is designed such that all devices are assigned a shared memory device offset of 0, then all register offsets must be unique. This is because the register offsets are relative to the beginning of the file. If all devices are assigned a non-zero shared memory device offset, then there may be multiple tags with the same offset as these register offsets are relative to the beginning of the device under the channel. Setting this property incorrectly poses a safety hazard, because overlapping registers may result in timeouts or corrupt data.

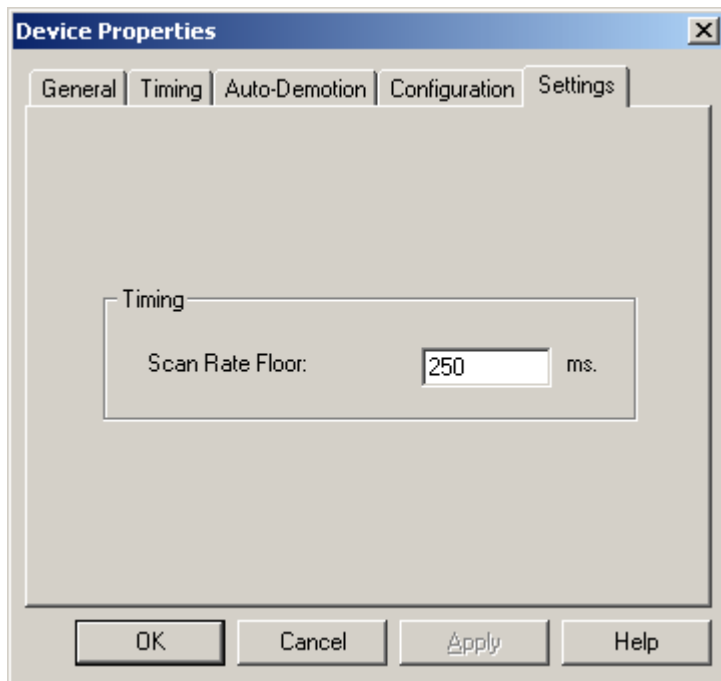
When the Override Configuration File setting is unchecked, this property will hold the value last imported during automatic device/tag generation, despite being disabled. When the Override Configuration File setting is checked, this property will become enabled and modifiable. The range is from 0-2147483647. The default value is 0.

- **Device Identifier:** The device identifier is an optional label used to identify the device. It is useful in associating the device with a physical device/data provider and may contain up to 256 characters.

**Note:** If the Override Configuration File parameter is unchecked after either the shared memory device offset or the device identifier settings have been changed, the settings will return to their defaults (which are the values last loaded from the configuration file).

## Settings

### Timing



#### Scan Rate Floor

This parameter specifies the minimum scan rate for this device. Any client scan rate that is faster than the device's scan rate floor setting will be capped at the device's setting. The minimum value is 250ms; the maximum value is 60000ms. The default setting is 250.

#### Data Types Description

Data Type	Description
Boolean	Single bit
Char	Signed 8 bit value  bit 0 is the low bit bit 6 is the high bit bit 7 is the sign bit
Byte	Unsigned 8 bit value  bit 0 is the low bit bit 7 is the high bit
Short	Signed 16 bit value  bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
Word	Unsigned 16 bit value  bit 0 is the low bit bit 15 is the high bit
DWord	Unsigned 32 bit value  bit 0 is the low bit bit 31 is the high bit
Long	Signed 32 bit value  bit 0 is the low bit



	bit 30 is the high bit bit 31 is the sign bit
BCD	Two byte packed BCD Value range is 0-9999.  Behavior is undefined for values beyond this range.
LBCD	Four byte packed BCD Value range is 0-99999999.  Behavior is undefined for values beyond this range.
Float	32 bit floating point value  bit 0 is the low bit bit 31 is the high bit
Double	64 bit floating point value  bit 0 is the low bit bit 63 is the high bit
String	Typically null terminated, null padded or blank padded ASCII string.  For some drivers only, Strings include HiLo LoHi byte order selection.
Date	64 bit floating point value

## Address Descriptions

Data is represented in a structure called a Shared Memory Data Register. Each register maintains separate Read and Write values in addition to corresponding error, quality, timestamp and status information. The following information describes how to access Read and Write values.

The syntax is as follows:

D<byte offset> [/ <string length>] [ [<row>] [<column>] ]

Component	Action	Description
<Byte Offset>	Required	Represents the register's byte offset relative to the device's memory map offset.
<String Length>	Optional	String length.
<Row>	Optional	Array row count.
<Column>	Optional	Array column count.

## Ranges

The maximum shared memory file size is limited to 2,147,483,648 bytes (2 gigabytes). This limitation will be enforced at project load (schema) and address validation.

## Valid Combinations/Hints

D<byte offset>

D<byte offset> / <string length>

D<byte offset> [<row>] [<column>]

## Examples

1. 8-bit value at byte offset 0

Address = D0

Data Type = Byte

2. Bit 5 of an 8-bit value at byte offset 0

Address = D0.5

Data Type = Byte

3. 32-character string value starting at byte offset 1000

Address = D1000/32

Data Type = String

## Automatic Device/Tag Generation

---

The CIDA will be responsible for generating a configuration file that will be imported by the CID and used for automatically generating devices and tags. When running as a System Service, this file should reside locally and not in a network location. For more information, refer to [Channel Setup](#).

The driver will perform automatic device/tag generation under the following circumstances.

1. Upon completion of the Channel Setup Wizard with a valid configuration file specified.
2. If the configuration file name changes in Dialog mode and is then applied.

### Device Import

Devices will be automatically generated based on the following information.

- Device Name
- Device Identifier
- Custom Properties

**Note:** The device identifier will not be validated. It is included for organizational purposes only.

### Tag Import

Tags will be automatically generated based on the following information.

- Tag Name
- Tag Address
- Tag Data Type
- Tag Read/Write Access
- Tag Scan Rate (Milliseconds)
- Tag Description

## Manually Starting Automatic Device/Tag Generation

To manually start automatic device/tag generation, open the **Configuration** tab in **Channel Properties** and then click the **Regenerate** button.

## Error Descriptions

---

The following error/warning messages may be generated. Click on the link for a description of the message.

### General Errors

[Cannot open Shared Memory file associated with Configuration '<configuration name>'. Please verify CIDA is running with proper permissions and Configuration name is correct](#)

### Read Errors

[Unable to read from register '<register offset>' on device '<device name>'. Register is not configured for read access](#)

[Unable to read from register '<register offset>' on device '<device name>'. Register value type is not configured for read data](#)

[Unable to read from register '<register offset>' on device '<device name>'. Register corrupted](#)

[Unable to read from register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'](#)

### Write Errors

[Unable to write to register '<register offset>' on device '<device name>'. Register is not configured for write access](#)

[Unable to write to register '<register offset>' on device '<device name>'. Register value type is not configured for write data](#)

[Unable to write to register '<register offset>' on device '<device name>'. Register corrupted](#)

[Unable to write to register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'](#)

---

## General Errors

The following error/warning messages may be generated. Click on the link for a description of the message.

### General Errors

[Cannot open Shared Memory file associated with Configuration '<configuration name>'. Please verify CIDA is running with proper permissions and Configuration name is correct](#)

---

### Cannot open Shared Memory file associated with Configuration '<configuration name>'. Please verify CIDA is running with proper permissions and Configuration name is correct

#### Error Type:

Warning

#### Possible Cause:

No Shared Memory file and/or global mutex has been created or associated with Configuration Name '<configuration name>'.

#### Solution:

Make sure that the CIDA is running and creating the Shared Memory file/global mutex associated with the Configuration Name '<configuration name>'. Third party utilities (such as Microsoft's Process Explorer) can be used to determine if these resources have been created.

#### Notes:

For launch instructions, help location and support contact information, refer to the Support Information tab in Channel Properties.

#### See Also:

[Support Information](#)

---

## Read Errors

The following error/warning messages may be generated. Click on the link for a description of the message.

### Read Errors

[Unable to read from register '<register offset>' on device '<device name>'. Register is not configured for read access](#)

[Unable to read from register '<register offset>' on device '<device name>'. Register value type is not configured for read data](#)

[Unable to read from register '<register offset>' on device '<device name>'. Register corrupted](#)

[Unable to read from register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'](#)

---

### Unable to read from register '<register offset>' on device '<device name>'. Register is not configured for read access

#### Error Type:

Warning

#### Possible Cause:

1. The register was not properly configured by the CIDA.
2. An incorrect register offset was provided for the address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Unable to read from register '<register offset>' on device '<device name>'. Register value type is not configured for read data**

---

**Error Type:**

Warning

**Possible Cause:**

1. The register was not properly configured by the CIDA.
2. An incorrect register offset was provided for this address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Unable to read to register '<register offset>' on device '<device name>'. Register corrupted**

---

**Error Type:**

Warning

**Possible Cause:**

1. The register was not properly configured by the CIDA.
2. An incorrect Register offset was provided for this address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Unable to read from register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'**

---

**Error Type:**

Warning

**Possible Cause:**

The device/data provider being polled/written by the CIDA has returned an error.

**Solution:**

For the help location and support contact information to determine the cause and solution for this error, refer to the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

**Write Errors**

---

The following error/warning messages may be generated. Click on the link for a description of the message.

**Write Errors**

[Unable to write to register '<register offset>' on device '<device name>'. Register is not configured for write access](#)

[Unable to write to register '<register offset>' on device '<device name>'. Register value type is not configured for write data](#)

[Unable to write to register '<register offset>' on device '<device name>'. Register corrupted](#)

[Unable to write to register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'](#)

**Unable to write to register '<register offset>' on device '<device name>'. Register is not configured for write access**

---

**Error Type:**

Warning

**Possible Cause:**

1. The register was not properly configured by the CIDA.
2. An incorrect register offset was provided for this address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

**Unable to write to register '<register offset>' on device '<device name>'. Register value type is not configured for write data**

---

**Error Type:**

Warning

**Possible Cause:**

1. The register was not properly configured by the CIDA.
2. An incorrect register offset was provided for this address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support

Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Unable to write to register '<register offset>' on device '<device name>'. Register corrupted**

---

**Error Type:**

Warning

**Possible Cause:**

1. The register was not properly configured by the CIDA.
2. An incorrect register offset was provided for this address.

**Solution:**

Confirm that the latest configuration file associated with the CID Application Configuration has been imported.

**Notes:**

For more information on launch instructions, help location and support contact information, refer the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Unable to write to register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>'**

---

**Error Type:**

Warning

**Possible Cause:**

The device/data provider being polled/written by the CIDA has returned an error.

**Solution:**

For the help location and support contact information to determine the cause and solution for this error, refer to the Support Information tab in Channel Properties.

**See Also:**

[Support Information](#)

---

**Developer Information**

---

The Developer Information pages provide in-depth information for understanding and implementing a CID Application. For information on a specific topic, select a link from the list below.

[CIDA Overview](#)

[Shared Memory Interface](#)

[Reference Implementation](#)

---

**CIDA Overview**

---

The Custom Interface Driver (CID) has two components: the CID driver plug-in (which runs within the OPC server's process) and the Custom Interface Driver Application (CIDA). The Custom Interface Driver (CID) allows external parties to push data into and out of the OPC server for reads and writes, while relying on the OPC server to provide connectivity to all of its client/server protocols. The CIDA will be responsible for creating, initializing and freeing the shared memory area. It will also be responsible for creating a configuration file, which must conform to a schema defined by the OPC server for use with the CID. Users will be able to import the configuration file from within the CID

channel settings, which will automatically generate devices and tags in the server.

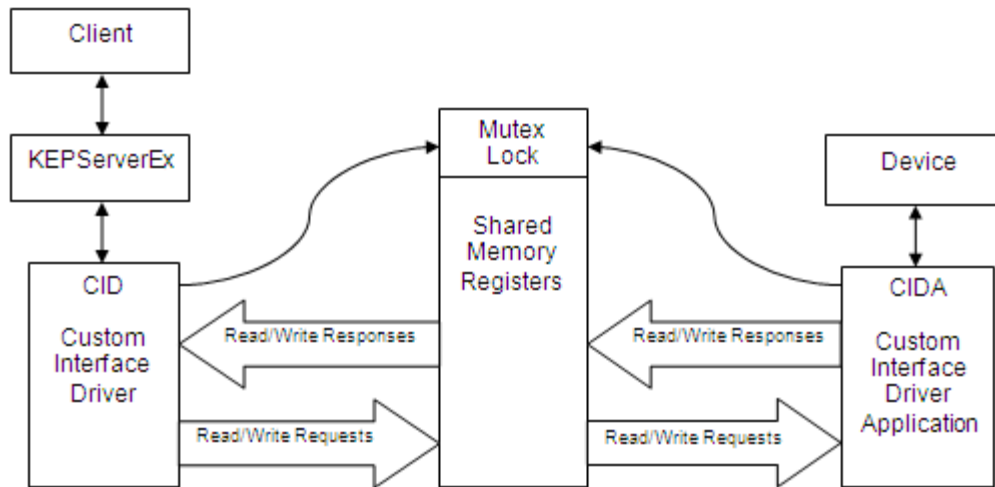


Figure 1 – CID CIDA Overview

### Shared Memory

In order to gain access to shared memory, both the shared memory client and the server must acquire a lock. Once acquired, the process can read and write to shared memory. Since only one process can hold the lock at a time, the lock shouldn't be held for too long because it starves the other processes from operating. Once the process is done working with the shared memory, it must release the lock immediately.

### CIDA Requirements

The CIDA must use Win32 API for Shared Memory creation, initialization, usage and destruction. Thus, developers must use a language that supports Win32 API calls, such as C++, VB.NET, C# or Java Native Interface (JNI).

### CIDA Functions

The following information describes specific functions and the order in which these actions must be performed.

1. The CIDA must create the shared memory file if one does not exist.
  - a. It must use **OpenFileMapping** with or without security\* to open the shared memory file. If the file cannot be opened, use **CreateFileMapping** with or without security\* to create the shared memory file.

**Note:** Normally the CIDA creates the shared memory file and the CID opens the shared memory file. In the case where the CIDA is restarted after the CID has gained access to the shared memory file, the CIDA will be required to open the file (since it is already created).

- b. File name must be of the form **Global\<Configuration Name>\_sm** and cannot exceed 100 characters.
  - c. File size cannot exceed 2GB.
  - d. It must use **MapViewOfFile/UnMapViewOfFile** to map shared memory file into the CIDA's process space.
  - e. It must use **CloseHandle** on exit to release the shared memory file.

2. The CIDA must create a global mutex for the purpose of locking access to the shared memory file.
  - a. It must use **CreateMutex** with or without security to create the mutex\*.

**Note:** The mutex should be created and locked prior to creating and initializing the shared memory to ensure that the CID does not access the memory prematurely.

- b. Mutex name must be of the form **Global\ <Configuration Name>\_sm\_lock**.

3. The CIDA must conform to the Shared Memory Data Register structure in the following ways. **See Also:** [Shared](#)

**Memory Interface.**

- a. If the Register supports reads, it must be configured as follows.
  - a.i. Assigns **REGISTER.ReadOffset**.
  - a.ii. Initializes **DATA.STATUS** flags to 0.
  - a.iii. Initializes **VALUE.Type**.
  - a.iv. Initializes **VALUE.\_Value**.
  - a.v. Initializes **VALUE.ExtSize** if register is a string or an array.
  - a.vi. Reserves **VALUE.ExtSize** bytes starting at **VALUE.ExtValue** for string and array data.
  - a.vii. Initializes **VALUE.ExtValue** for string and array data.
  - a.viii. Initializes **STRINGARRAY.StringSize** if register is a string array. **STRINGARRAY** must be cast onto **VALUE.ExtValue** for string arrays.
- b. If the Register supports writes, it must be configured as follows.
  - b.i. Assigns **REGISTER.WriteOffset**.
  - b.ii. Initializes **DATA.STATUS** flags to 0.
  - b.iii. Initializes **VALUE.Type**.
  - b.iv. Initializes **VALUE.\_Value**.
  - b.v. Initializes **VALUE.ExtSize** if register is a string or an array.
  - b.vi. Reserves **VALUE.ExtSize** bytes starting at **VALUE.ExtValue** for string and array data.
  - b.vii. Initializes **VALUE.ExtValue** for string and array data.
  - b.viii. Initializes **STRINGARRAY.StringSize** if register is a string array. **STRINGARRAY** must be cast onto **VALUE.ExtValue** for string arrays.
- c. Checks **WriteData.RequestPending** to determine if a write request is available. When true:
  - c.i. Caches the write value.
  - c.ii. Clears **WriteData.RequestPending** before performing the write.
  - c.iii. Performs the write.
- d. Sets **WriteData.Error/ErrorCode** based on a successful or failed write response.
- e. Sets **WriteData.ResponsePending** after **WriteData.Error/ErrorCode** is set on a successful or failed write response.
- f. Checks **ReadData.RequestPending** to determine if a read request is available. When true:
  - f.i. Clears **ReadData.RequestPending** before performing the read.
  - f.ii. Performs the read.
- g. Sets **ReadData.Error/ErrorCode** based on a successful or failed read response.
- h. Sets **ReadData.Quality** on a successful or failed read response.
- i. Sets **ReadData.Timestamp** on a successful or failed read response.
- j. Sets **ReadData.Value** on a successful read response.



k. Sets **ReadData.ResponsePending** after items g. through j. are set on a successful or failed read response.

4. The CIDA must release resources for global mutex and shared memory file using the following:

- a. CloseHandle returned from **CreateFileMapping**.
- b. CloseHandle returned from **CreateMutex**.

\*Creating the shared memory file and mutex with security attributes is optional but recommended. The only requirement is that the CID must be able to open these resources from where it is running.

**Note:** The CID will be running in the context of a Windows Service under the System Account by default.

## Shared Memory Interface

The CID and CIDA communicate via Shared Memory by agreeing on a protocol that will be used to exchange information (in the same way that an application communicates with hardware). This protocol or interface is defined through structures, which are mapped onto the shared memory file's byte memory.

The main structure that defines a data item is the REGISTER structure. Each data item requires its own REGISTER. Its location within the shared memory is set by the CIDA and is termed the byte offset in [Address Descriptions](#). The byte offset must be unique for each REGISTER. When calculating the byte offset, users must take the footprint consumed by the REGISTER and its nested structures into account in order to prevent overlap with other registers. This is important because overlapping registers pose a safety hazard: they can result in timeouts and corrupt data.

For information on a specific structure, select a link from the list below.

[REGISTER Structure](#)

[DATA Structure](#)

[VALUE Structure](#)

[STRINGARRAY Structure](#)

## REGISTER Structure

The following table describes how a register is arranged in Shared Memory.

Register Member	Offset	Size	Description
DWORD ReadOffset	0	4	Value of 0 in this field is Write Only, otherwise byte offset (R) from beginning of this header to ReadData. Typically this is 12 bytes.
DWORD WriteOffset	4	4	Value of 0 in this field is Read Only, otherwise byte offset (W) from beginning of this header to WriteData.
BYTE Reserved [4]	8	4	Reserved.
DATA ReadData*	R	Variable.	Read data for this register.
DATA WriteData*	W	Variable.	Write data for this register.

\*For more information, refer to [DATA Structure](#).

## DATA Structure

Data Member	Offset	Size	Description
typedef struct _tagSTATUS { WORD RequestPending : 1;  WORD ResponsePending : 1;	0	2	Status bit field.  True for new Read or Write request. Set by the CID, cleared by the CIDA.  True when new Read or Write response (success or failure)

WORD Error : 1;  WORD Reserved : 13; } STATUS			is available. Set by the CIDA, cleared by the CID.  True when ErrorCode is applicable. CID will not look at ErrorCode if error is false. Set by the CIDA upon error condition, cleared by CIDA when error condition no longer exists.  Reserved.
DWORD ErrorCode	2	4	Vendor-specific error that will be posted in OPC server's event log. Corresponding tag will be set to Quality. CID will not look at ErrorCode if error is false.
WORD Quality	6	2	For valid OPC quality values, refer to the <i>Visual C++ Custom Interface VS2008 Example</i> for a complete list of OPC Quality values.
FILETIME Timestamp	8	8	64-bit structure representing the number of 100-nanosecond intervals since 1/1/1601.  typedef struct { DWORD LowDateTime DWORD HighDateTime } FILETIME
VALUE Value*	16	13**	The data value.

\*For more information, refer to [VALUE Structure](#).

\*\*Size will exceed this value for string and array data.

## VALUE Structure

Value Member	Offset	Size	Description
VALTYPE Type	0	2	Where VALTYPE is a WORD with possible values:  0 T_UNDEFINED 1 T_BOOL 2 T_BYTE 3 T_CHAR 4 T_WORD 5 T_SHORT 6 T_DWORD 7 T_LONG 8 T_FLOAT 9 T_DOUBLE 10 T_DATE 11 T_STRING 0x1000 T_ARRAY  Value cannot be of type T_ARRAY by itself, it must be masked with one of the above types.
WORD Reserved	2	2	Reserved.
union { BOOL boolVal BYTE bVal CHAR cVal WORD wVal SHORT iVal DWORD dwVal LONG lVal FLOAT fltVal DOUBLE dblVal DATE dateVal	4	8	The data value will be stored in this union to facilitate easy access without type conversion. The only exceptions are string and array data, which will utilize ExtValue.

} _Value;			
WORD ExtSize	12	2	Maximum number of bytes ExtValue is capable of storing.
ExtValue [ ]	14	Variable.	The data value for string and array data will be stored at this placeholder.

## Strings and Arrays

String length and array data are unknown at compile time. They are application-specific and chosen at runtime. String length and array data are stored in the byte array "ExtValue" in order to allow for variable length. The number of bytes in ExtValue is stored in "ExtSize."

### String Data

ExtValue will store the packed, wide character, null terminated string data. Wide characters use 2 bytes per character, thereby allowing for Unicode strings. It is recommended that users be cautious when copying string data into ExtValue, since ExtSize is the maximum number of bytes that can be stored in ExtValue, not the number of characters. The number of characters that can be stored is  $\text{ExtSize} / 2$ .

**Important:** ExtSize is not the current length of the string.

### Array Data

ExtValue will store the array data in the order given in the examples below.

#### Example 1

1 Dimensional Array (1 row, y columns)  
Element[0], Element[1], ...Element[n]

#### Example 2

2 Dimensional Arrays (x rows, y columns)  
Element[0][0], ... Element[0][y], Element [1][0], ... Element[1][y], ... Element[x][y]

### String Arrays

A string array is defined as an array of equal length strings. They are stored similar to standard arrays with the exception that a WORD precedes the array data, representing the maximum number of characters in each string (not the length). The structure STRINGARRAY encapsulates this value and the array data to follow. For more information, refer to [STRINGARRAY Structure](#).

## STRINGARRAY Structure

STRINGARRAY Member	Offset	Size	Description
WORD StringSize	0	2	The size of each string in the array as a wide character count. Each string in the array is of equal length.
BYTE Data [ ]	2	Variable.	Placeholder for string array data.

### Example

String Array D0/10 [5]  
This is an array of 5, 10 character strings.

VALUE.ExtSize would equal (10 characters \* 2 bytes per wide character) \* 5 elements + sizeof (StringSize) = 102 bytes.

The STRINGARRAY would be cast onto VALUE.ExtValue where STRINGARRAY.StringSize would be set to 10. Assuming the string data = "hello", "world", the first two array elements of STRINGARRAY.Data would be stored as:

'h' 'e' 'l' 'l' 'o' 0x00 0x00 0x00 0x00 0x00, 'w' 'o' 'r' 'l' 'd' 0x00 0x00 0x00 0x00 0x00

## Reference Implementation

A reference implementation is provided with the driver. It will do the following.

1. Create a shared memory file.
2. Create a mutex available to multiple processes.

3. Define two devices, each with a name, device identifier and tag table.
4. Define a tag table for each device that contains at least 5 tags. Implementation should allow for expanding the available number of tags by adding to the table only. Include string and array references so data serialization can be fully demonstrated.

Each tag will have the following attributes:

- Name (such as Valve1).
- Register offset.
- Data type. The data types included are as follows:

- 4.a.Boolean
- 4.b.Char
- 4.c.Byte
- 4.d.Short
- 4.e.Word
- 4.f.Long
- 4.g.DWord
- 4.h.Float
- 4.i.Double
- 4.j.Date
- 4.k.String

- Array Size is -1 if not applicable.
- Description (such as Slurry output).
- Group (such as X Axis).

5. Simulate reading from a device by reading from a cached value on every request. This value is used as the Read response.
6. Simulate writing to a device by writing to a cached value with the value in the Write request.
7. Perform the simulated Reads and Writes outside the shared memory lock.
8. Accept command line argument "exportconfig" to export the configuration file that meets the requirements of the configuration schema. Shared memory name and size will be hardcoded in the sample. The following support information is included:

- 8.a.**Company Name:** My Company
- 8.b.**Phone:** 1-888-555-1212
- 8.c.**Email:** support@mycompany.com
- 8.d.**Configuration Launch Hint:** At the command prompt...
- 8.e.**Runtime Launch Hint:** At the command prompt...
- 8.f.**Help Launch Hint**
- 8.g.**Additional**

9. Output XML in rudimentary fashion (i.e. string output), rather than utilizing XML DOM/SAX parser in this reference implementation.

## Reference Implementation Architecture

The Shared Memory Server requirements were listed in [CIDA Overview](#). The following is for implementing a reference sample only and not a requirement of all CIDs. The four main classes are CRuntime, CDevice, CTag and CValue.

### CRuntime

- Creates CDevices and CTags from table.
- Exports configuration settings to configuration file when requested.
- Thread for processing reads and writes.
- Owns list of CDevices.

**CDevice**

- Exports its settings to configuration file when requested.
- Provides next tag to process in CRuntime thread.
- Owns list of CTags.

**CTag**

- Exports its settings to configuration file when requested.
- Read and write data members for caching data from Shared Memory.

**Note:** Both Write data from Shared Memory and Simulated Read data are stored at this object.

**CValue**

- Performs value simulation.
- Manages extended value (arrays and strings) for local storage (CTag).
- Provides VALTYPE helper functions.
- Provides Date and Time conversion functions.

**Threads**

The Reference Implementation has two threads: **Main Application Thread** and **Read/Write Thread**. The Main Application Thread is responsible for creating and destroying the Read/Write thread. It also fires a quit event when the quit character is entered at the command line. The Read/Write Thread is responsible for reading from or writing to the device and shared memory.

**Read/Write to Device**

1. If the current tag has its "Write request pending" flag set, the following will occur:
  - A write will be performed to the device.
  - Upon completion, the write result will be cached and the tag flag will be set as "Write response pending."
2. If the current tag has its "Read request pending" flag set, the following will occur:
  - A read will be performed from the device.
  - Upon completion, the read result will be cached and the tag flag will be set as "Read response pending."

**Read/Write to Shared Memory**

1. If the current tag has its "Write response pending" flag set, the following will occur:
  - The Shared Memory will be updated with the Write result.
  - The Write ResponsePending flag will be set.
2. If the current tag has its "Read response pending" flag set, the following will occur:
  - The Shared Memory will be updated with the Read result.
  - The Read ResponsePending flag will be set.
3. Get a new tag to process. The walk list of devices and device's list of tags are as follows: first device, first tag; first device, second tag and so on.
4. If the Shared Memory Register for the current tag has a Write RequestPending, set the "Write request pending" flag on tag.
5. If the Shared Memory Register for the current tag has a Read RequestPending, set the "Read request pending" flag on tag.
6. Read/Write to device.

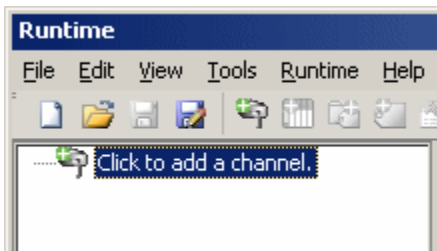
**CID/CIDA Reference Implementation Demonstration**

For information on implementing CID/CIDA, follow the instructions below.

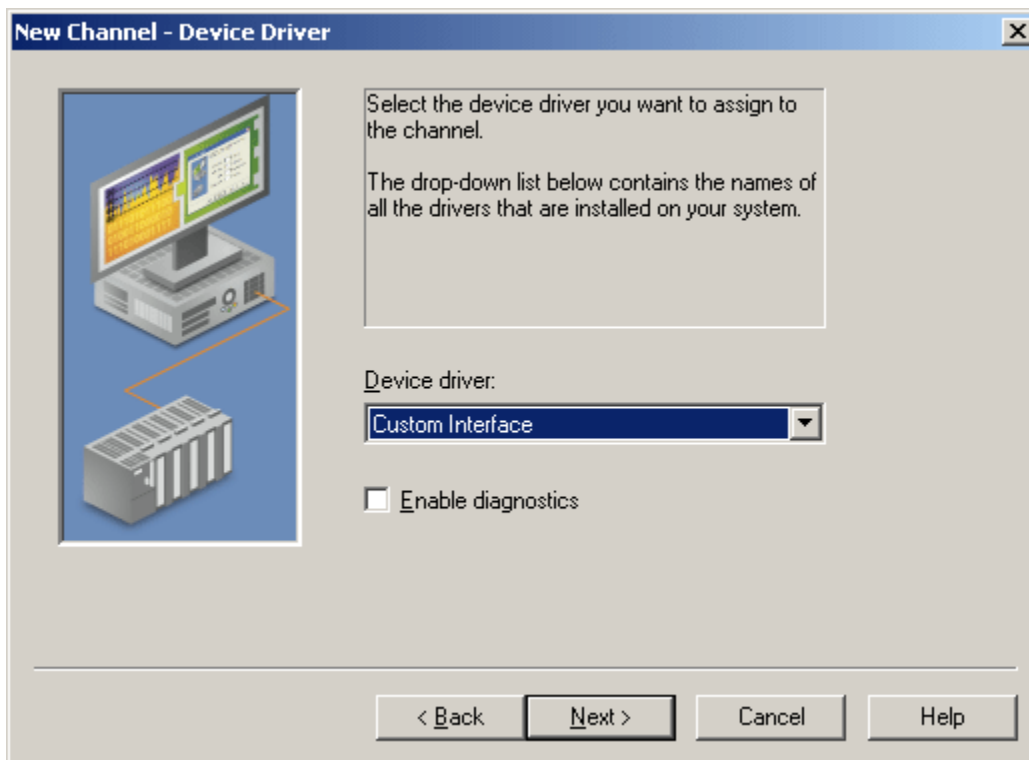
1. Click **Start** and then navigate to the **Example Source Code** menu. Then, select **Custom Interface Example**

**Code | Visual C++ Custom Interface VS2008 Example.**

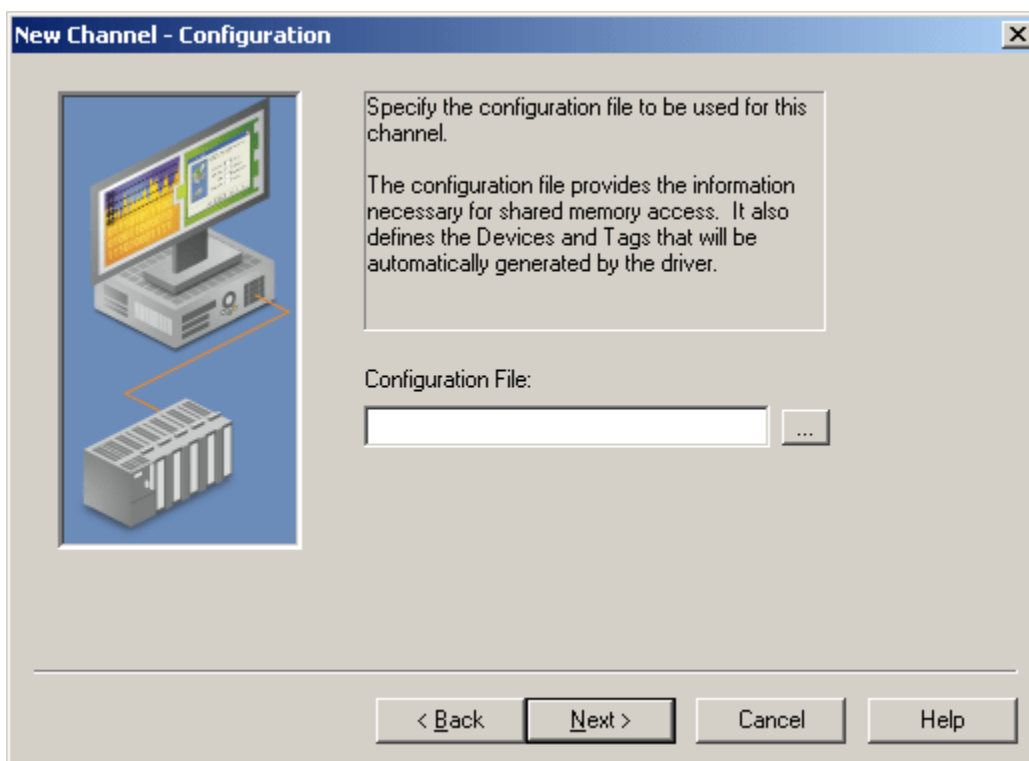
2. Build **cidarefimplcpp** for **Release Mode**.
3. Run **cidarefimplcpp.exe**. A console will appear that says "Enter q to quit."
4. Next, launch the OPC server.
5. In the **Administrator** tool, select **Configuration**. Then, click **File | New**.
6. To invoke a new Channel Wizard, click on **Click to add a channel**.



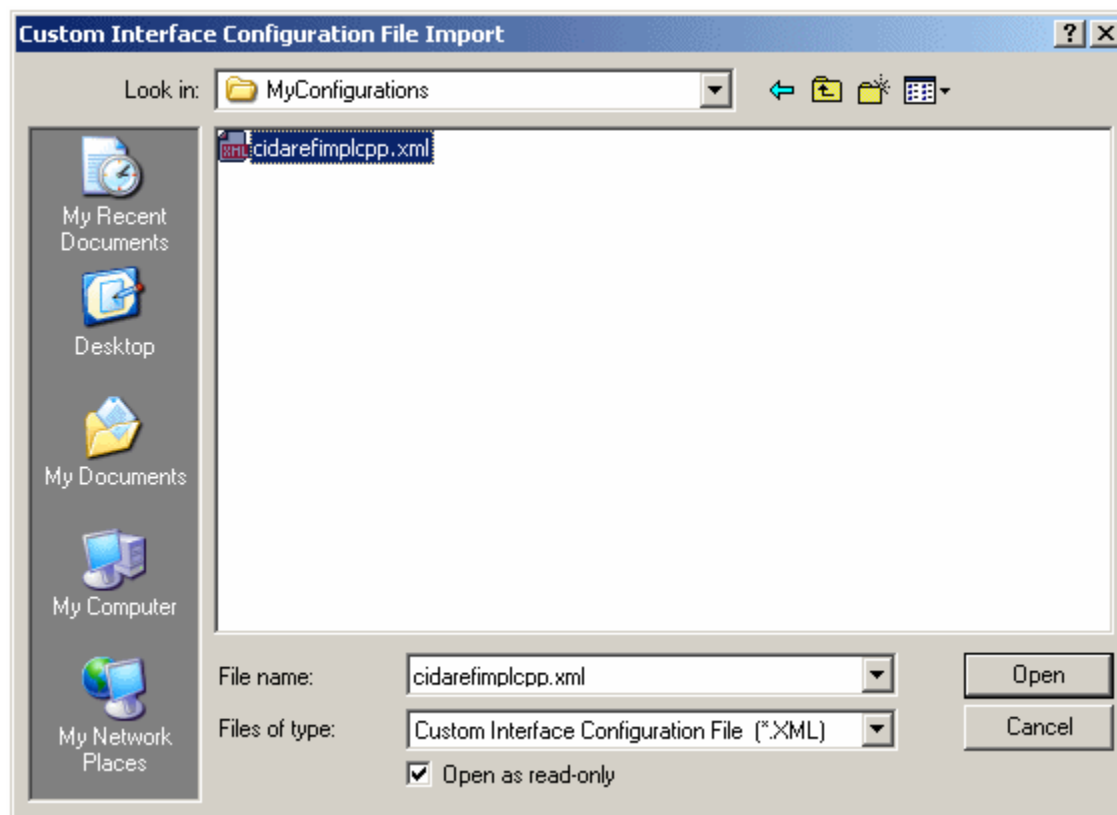
7. Click **Next**. In the drop-down box, select **Custom Interface**.



8. Click **Next**.
9. Click **Next**. When prompted to enter or browse for a configuration file, click on the browse button (designated by ellipses) in order to invoke the **File Open** dialog.

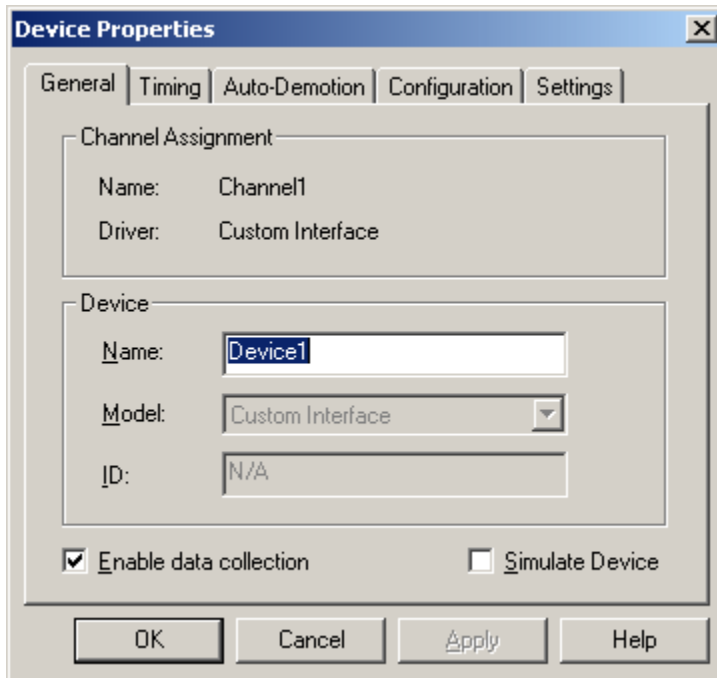


10. In the OPC server's directory, browse to the projects directory and then select "cidarefimplcpp.xml." Then, click **Next**.



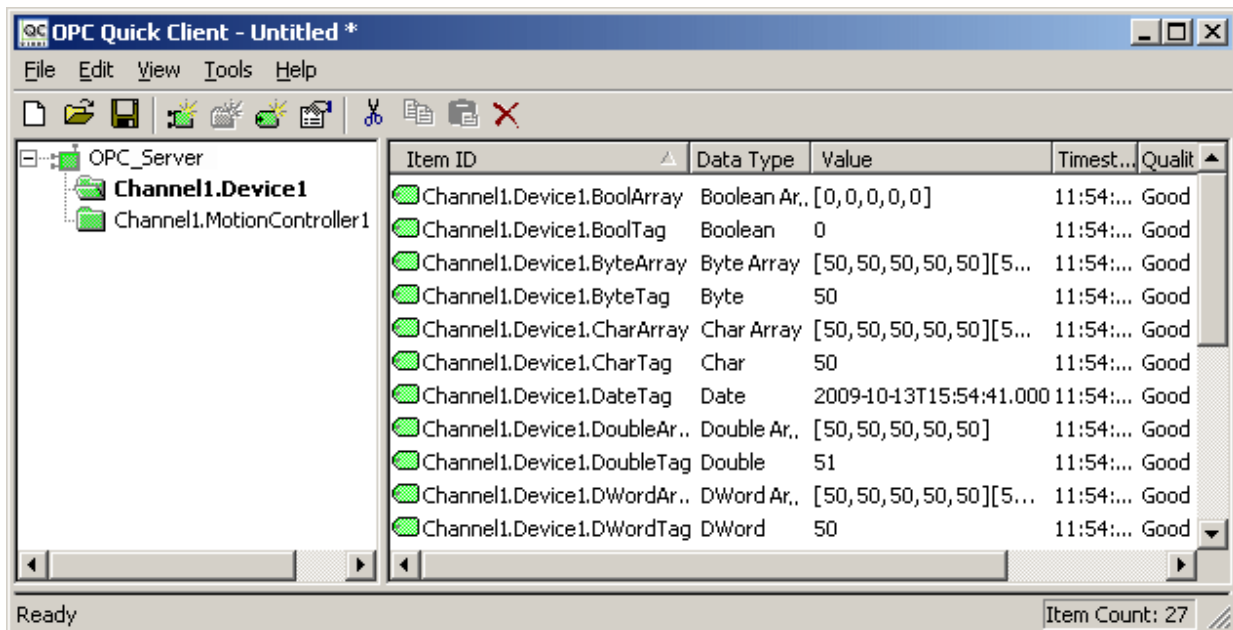
**Note:** If the file imported correctly, the Support Information will be visible in the configuration file.

11. To complete the Channel Wizard, click **Next** and then click **Finish**. Devices and tags will be generated automatically based on the configuration file.
12. Expand the channel: two devices called "Device1" and "MotionController1" should be visible. Expand the devices: tags and tag groups should be visible.
13. Right-click on one of the devices and then select **Properties**.
14. In the Device Properties dialog, users will note that the Device Name was automatically imported, that there are no models to choose from and that the ID is disabled with "N/A."



15. Click on the **Settings** tab to display the **Timing** parameters. Click on the **Configuration** tab to display the **Configuration File** and **Advanced** settings. For more information, refer to [Device Configuration](#).
16. Next, launch the OPC Quick Client that comes with the OPC server by clicking **Tools | Launch OPC Quick Client**.
17. Browse to **Channel1.Device1** and **Channel1.MotionController1**. Users will see items mapping to the tags within the OPC server.





18. Since "cidarefimplcpp.exe" was left running, all items should have Good quality with values ramping every second. The exception will be the Boolean tags (which will toggle) and the String/String Array tags (which will display an empty string).

## Channel Diagnostics

Channel Diagnostics are available to help users troubleshoot CID and CIDA issues. Prior to performing any function on shared memory, the CID will output a TX diagnostic frame detailing the nature of the function. Upon completion, the CID will output an RX diagnostic frame detailing the results of the function. Channel Diagnostics must be enabled in order to view these diagnostic frames. For more information, refer to "Channel Diagnostics" in the OPC server's help file.

A code has been defined for each function and is derived from the following bit field. For more information on specific function codes, refer to the Read and Write Transaction Frames below.

Bit 8 (Read/Write)	Bit 7 (Get/Set)	Bits 0-6 (Function Type)
0 (Read)	0 (Get)	0 (Reserved)
1 (Write)	1 (Set)	1 (Request Pending)
		2 (Response Pending)
		3 (Request)
		4 (Response)

The shared memory interface functions in the CID return a code to denote the success or failure of that function.

Shared Memory Return Code (SMRC)	Definition
0	No Error
1	Invalid byte offset specified and/or Register corrupt.
2	For reads, REGISTER.ReadOffset is 0. For writes, REGISTER.WriteOffset is 0.
3	Register DATA does not have a valid VALTYPE specified.

## Read Transaction Frames

The possible read transaction frames that can be exchanged are as follows.

**Set Read Request (0x43)**

Request a read of the register at the specified offset.

**TX:**

Function	1 Byte	0x43
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x43
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*

\*Refer to [Shared Memory Return Code](#) table above.

**Get Read Response Pending (0x02)**

Determine if a read response is available. If Pending Flag is not set in the defined timeout period, this read attempt will be considered failed. It will retry according to the attempt count. For more information, refer to "Device Properties - Timing" page in the OPC server's help file.

**TX:**

Function	1 Byte	0x02
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x02
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*
Pending Flag	2 Bytes	0 (False) or 1 (True)

\*Refer to [Shared Memory Return Code](#) table above.

**Get Read Response Data Block (0x04)**

Response from the last read request to the register at the specified offset.

**TX:**

Function	1 Byte	0x04
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x04
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*
Data	x Bytes	Read DATA structure of size x bytes

\*Refer to [Shared Memory Return Code](#) table above.

**Set Read Request Pending (0x41)**

Used to clear the read request that timed out. This will prevent the CIDA from servicing the request at a later time.

**TX:**

Function	1 Byte	0x41
Register Byte Offset	4 Bytes	0-2,147,483,647

Pending Flag	2 Bytes	0 (False) or 1 (True)
--------------	---------	-----------------------

**RX:**

Function	1 Byte	0x41
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*

\*Refer to [Shared Memory Return Code](#) table above.

**Set Read Response Pending (0x42)**

Used to clear the read response when the last read request timed out. This will prevent a new request from potentially using an old response.

**TX:**

Function	1 Byte	0x42
Register Byte Offset	4 Bytes	0-2,147,483,647
Pending Flag	2 Bytes	0 (False) or 1 (True)

**RX:**

Function	1 Byte	0x42
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*

\*Refer to [Shared Memory Return Code](#) table above.

**Write Transaction Frames**

The possible write transaction frames that can be exchanged in the Channel Diagnostics window are as follows.

**Get Write Request Pending (0x81)**

Determine if the last write request has been processed.

**TX:**

Function	1 Byte	0x81
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x81
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*
Pending Flag	2 Bytes	0 (False) or 1 (True)

\*Refer to [Shared Memory Return Code](#) table above.

**Set Write Request (0xC3)**

Request a write to the register at the specified offset.

**TX:**

Function	1 Byte	0xC3
Register Byte Offset	4 Bytes	0-2,147,483,647
DATA.Quality	2 Bytes	*
DATA.Timestamp	8 Bytes	*
Value Size	4 Bytes	Number of ASCII bytes to follow
Value	x Bytes	Value in ASCII form

\*Refer to [DATA Structure](#).

**RX:**

Function	1 Byte	0xC3
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	**

\*\*Refer to [Shared Memory Return Code](#) table above.

**Get Write Response Pending (0x82)**

Determine if a write response is available. If Pending Flag is not set in the defined timeout period, this write attempt will be considered failed. It will retry according to the attempt count. For more information, refer to "Device Properties - Timing" page in the OPC server's help file.

**TX:**

Function	1 Byte	0x82
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x82
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*
Pending Flag	2 Bytes	0 (False) or 1 (True)

\*Refer to [Shared Memory Return Code](#) table above.

**Get Write Response (0x84)**

Response from the last write request to the register at the specified offset.

**TX:**

Function	1 Byte	0x84
Register Byte Offset	4 Bytes	0-2,147,483,647

**RX:**

Function	1 Byte	0x84
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*
STATUS.Error	1 Byte	0 (False) or 1 (True)
DATA.ErrorCode	4 Bytes	CIDA/Device Specific
DATA.Quality	2 Bytes	**
DATA.Timestamp	8 Bytes	**

\*Refer to [Shared Memory Return Code](#) table above.

\*\*Refer to [DATA Structure](#).

**Set Write Request Pending (0xC1)**

Used to clear the write request that timed out. This will prevent the CIDA from servicing the request at a later time.

**TX:**

Function	1 Byte	0xC1
Register Byte Offset	4 Bytes	0-2,147,483,647
Pending Flag	2 Bytes	0 (False) or 1 (True)

**RX:**

Function	1 Byte	0xC1
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*

\*Refer to [Shared Memory Return Code](#) table above.

**Set Write Response Pending (0xC2)**

Used to clear the write response when the last write request timed out. This will prevent a new request from potentially using an old response.

**TX:**

Function	1 Byte	0xC2
Register Byte Offset	4 Bytes	0-2,147,483,647
Pending Flag	2 Bytes	0 (False) or 1 (True)

**RX:**

Function	1 Byte	0xC2
Register Byte Offset	4 Bytes	0-2,147,483,647
SMRC	1 Byte	*

\*Refer to [Shared Memory Return Code](#) table above.

# Index

## - A -

Address Descriptions 8  
Automatic Device/Tag Generation 9

## - C -

Cannot open Shared Memory file associated with Configuration '<configuration name>' 10  
Channel Configuration 3  
Channel Diagnostics 24  
Channel Setup 3  
CID/CIDA Reference Implementation Demonstration 20  
CIDA Overview 13  
CIDA Requirements 14

## - D -

DATA Structure 16  
Data Types Description 7  
Developer Information 13  
Device Configuration 6  
Device Setup 6

## - E -

Error Descriptions 9

## - G -

General Errors 10

## - H -

Help Contents 3

## - O -

Overview 3

## - R -

Read Errors 10  
Reference Implementation 18  
Reference Implementation Architecture 19  
REGISTER Structure 16

## - S -

Settings 6  
Shared Memory Interface 16  
STRINGARRAY Structure 18  
Support Information 5

## - U -

Unable to read from register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>' 11  
Unable to read from register '<register offset>' on device '<device name>'. Register is not configured for read access 10  
Unable to read from register '<register offset>' on device '<device name>'. Register value type is not configured for read data 11  
Unable to read to register '<register offset>' on device '<device name>'. Register corrupted 11  
Unable to write to register '<register offset>' on device '<device name>'. CIDA '<CIDA name>' returned error code '<error code>' 13  
Unable to write to register '<register offset>' on device '<device name>'. Register corrupted 13  
Unable to write to register '<register offset>' on device '<device name>'. Register is not configured for write access 12  
Unable to write to register '<register offset>' on device '<device name>'. Register value type is not configured for write data 12

## - V -

VALUE Structure 17

## - W -

Write Errors 12