Custom Interface Driver

© 2016 PTC Inc. All Rights Reserved.

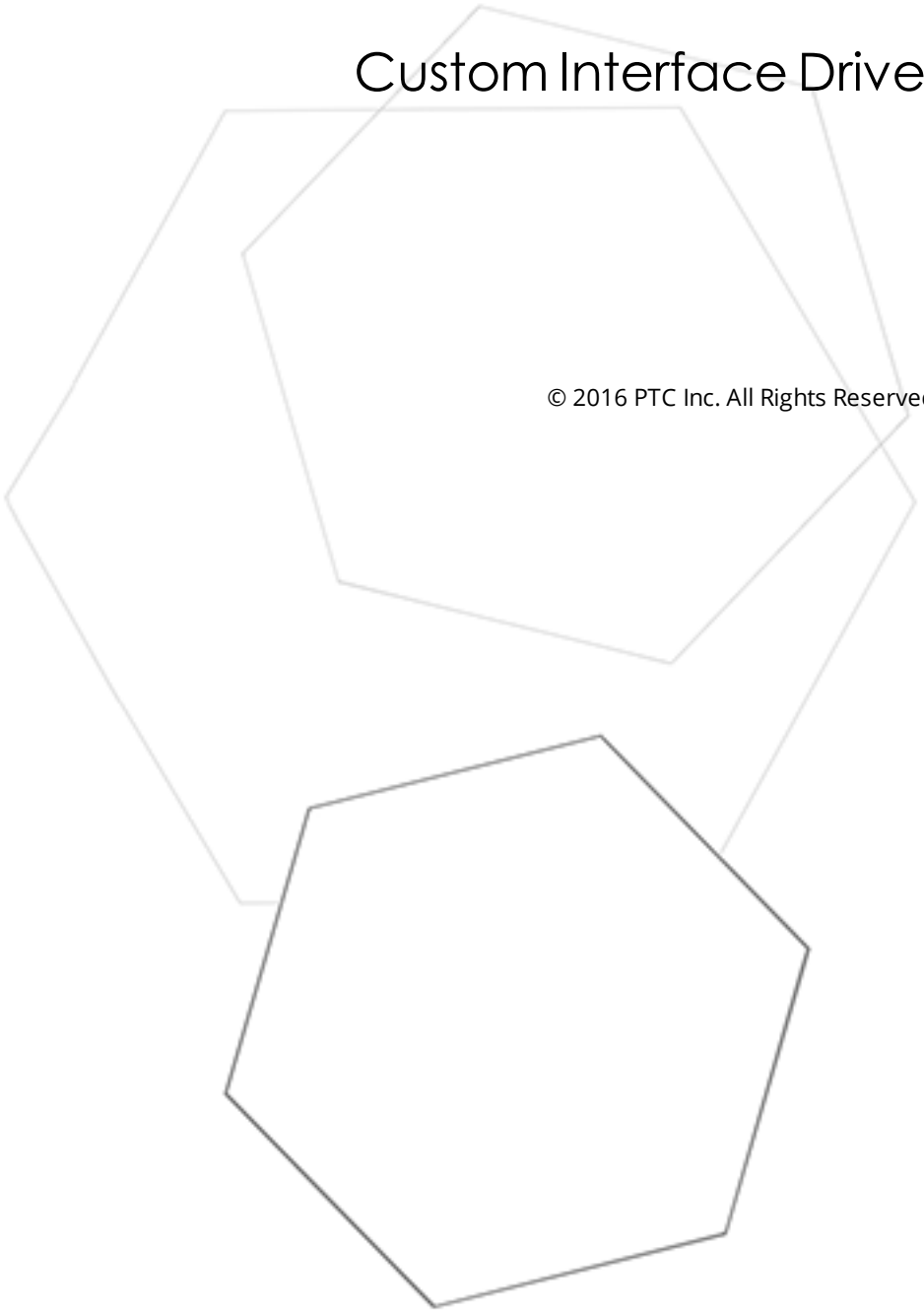


Table of Contents

Custom Interface Driver Help

Help version 1.027

CONTENTS

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 in addition to an XML configuration file that fully defines the data mapped in shared memory. The configuration file will then be imported into the CID 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**.

- *This 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**.*

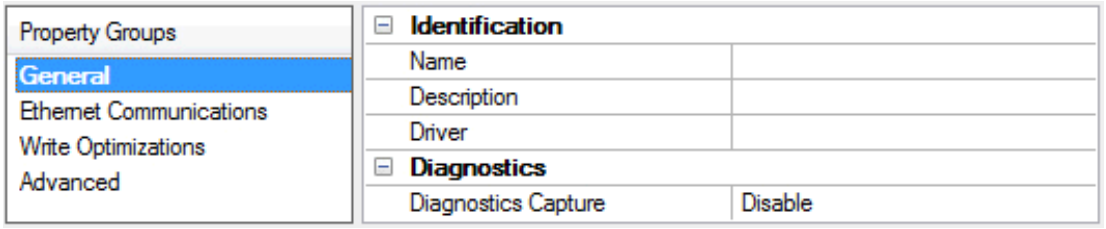
Channel Setup

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 (in addition to a list of the configuration's corresponding devices and tags). The CID can support multiple

configurations simultaneously; however, each configuration must contain a unique configuration name. For more information, refer to **Channel Configuration**.

Channel Properties - General

This server supports the use of simultaneous multiple communications drivers. Each protocol or driver used in a server project is called a channel. A server project may consist of many channels with the same communications driver or with unique communications drivers. A channel acts as the basic building block of an OPC link. This group is used to specify general channel properties, such as the identification attributes and operating mode.



Identification

Name: User-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information.
- *For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.*

Description: User-defined information about this channel.
- Many of these properties, including Description, have an associated system tag.

Driver: Selected protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties.

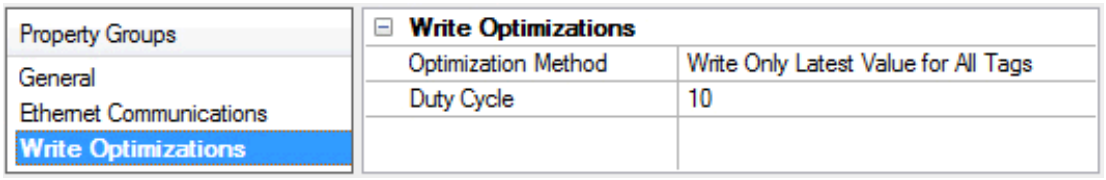
- **Note**: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

Diagnostics

Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.
- *For more information, refer to "Communication Diagnostics" in the server help.*
- **Note**: Not all drivers support diagnostics. To determine whether diagnostics are available for a particular driver, open the driver information and locate the "Supports device level diagnostics" statement.

Channel Properties - Write Optimizations

As with any OPC server, writing data to the device may be the application's most important aspect. The server intends to ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties that can be used to meet specific needs or improve application responsiveness.



Write Optimizations

Optimization Method: controls how write data is passed to the underlying communications driver. The options are:

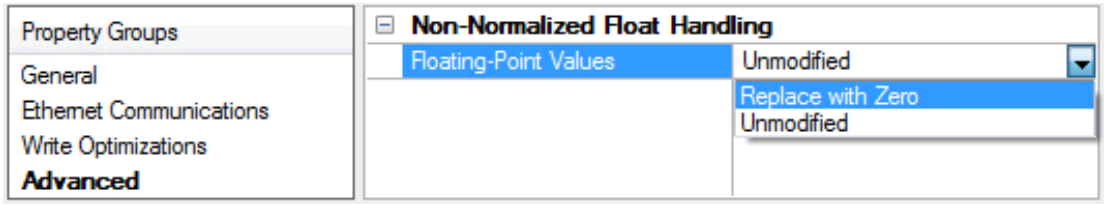
- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags**: Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.
  Note: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

Duty Cycle: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

Note: It is recommended that the application be characterized for compatibility with the write optimization enhancements before being used in a production environment.

Channel Properties - Advanced

This group is used to specify advanced channel properties. Not all drivers support all properties; so the Advanced group does not appear for those devices.



Non-Normalized Float Handling: Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. A non-normalized value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. The default is Replace with Zero. Drivers that have native float handling may default to Unmodified. Descriptions of the options are as follows:

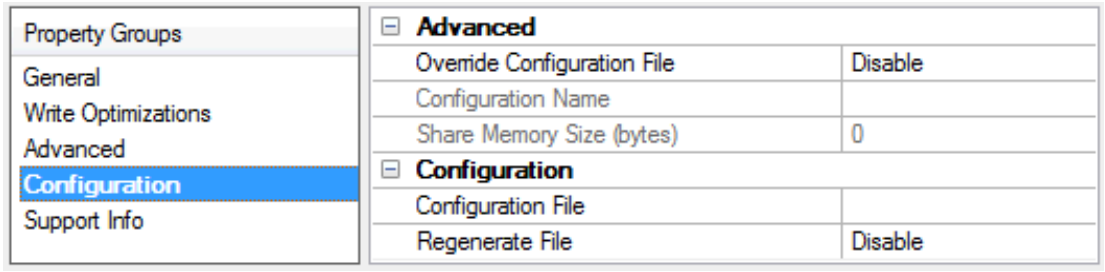
- **Replace with Zero**: This option allows a driver to replace non-normalized IEEE-754 floating point values with zero before being transferred to clients.
- **Unmodified**: This option allows a driver to transfer IEEE-754 denormalized, normalized, non-number, and infinity values to clients without any conversion or changes.

Note: This property is disabled if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM data is not affected by this setting.lin

For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.

Channel Properties - Configuration

The Configuration property group 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 shared memory file's name, size, associated devices, and appropriate tag definitions. This property is used to specify its location. To access the **File Import** property group, click the custom action button in the Configuration File property and set the filter to "*.xml." Select the desired file and then click **Open**. The configuration name within the file must be unique so that one channel does not interfere with another channel's access to a register. Once set, the file will be checked for uniqueness.

Caution: Users must regenerate the configuration file every time a change is made to it. For more information, refer to "Regenerate" below.

Important: When running in System Service Mode, the configuration file must be located in an accessible folder in order to be loaded by the Runtime. For example, 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 server help file.

Regenerate

The **Regenerate File** property is used to manually start automatic device/tag generation. To trigger the action, set the property to Enable and click **Apply**. For more information, refer to **Automatic Device/Tag Generation**.

Advanced

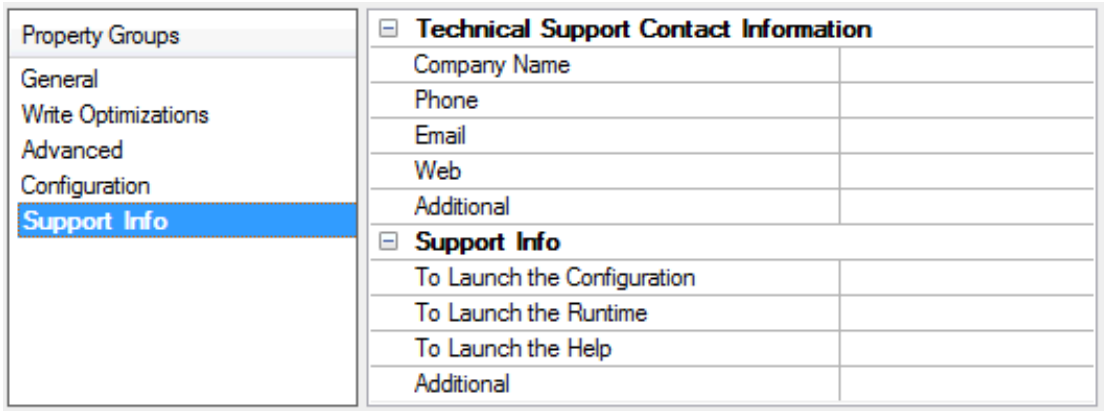
The Advanced settings should only be used by CID Application Developers. Descriptions of the properties are as follows:

- **Override Configuration File:** When enabled, this option overrides the configuration name and shared memory size as defined by the configuration file. In the absence of a configuration file, the associated properties may be used to manually define the configuration name and shared memory size. The default setting is disabled.
- **Configuration Name:** This property contains the configuration name that the channel will use when accessing shared memory. It must be unique across channels so that one channel will not interfere with another channel's access to a register. The driver will check for uniqueness automatically, and warn the user/override when necessary. The default setting is "cidarefimplcpp".
- **Shared Memory Size (bytes:** This property contains the shared memory size that the channel will use during tag address validation. When the Override Configuration File setting is disabled, this property will hold the value last imported during automatic device/tag generation despite being disabled. When the Override Configuration File setting is enabled, this property will be enabled and capable of modification. The range is 0 to 2,147,483,648. The default setting is 0.

Note: If the Override Configuration File property is disabled after either the configuration name or shared memory size settings have been changed, the settings will return to their default settings. The default settings are the values last loaded from the configuration file.

Channel Properties - Support Information

The property group contains support information. The information may vary depending on what the supporting party has decided to include.



Technical Support Contact Information

This information includes the supporting party's company name, phone, email address, web address, and any additional information.

Support Info

To Launch the Configuration: This text includes brief instructions on how to start the supporting party's Configuration component.

To Launch the Runtime: This text includes brief instructions on how to start the supporting party's Runtime component.

To Launch the Help: This text includes brief instructions on how to access the supporting party's help documentation.

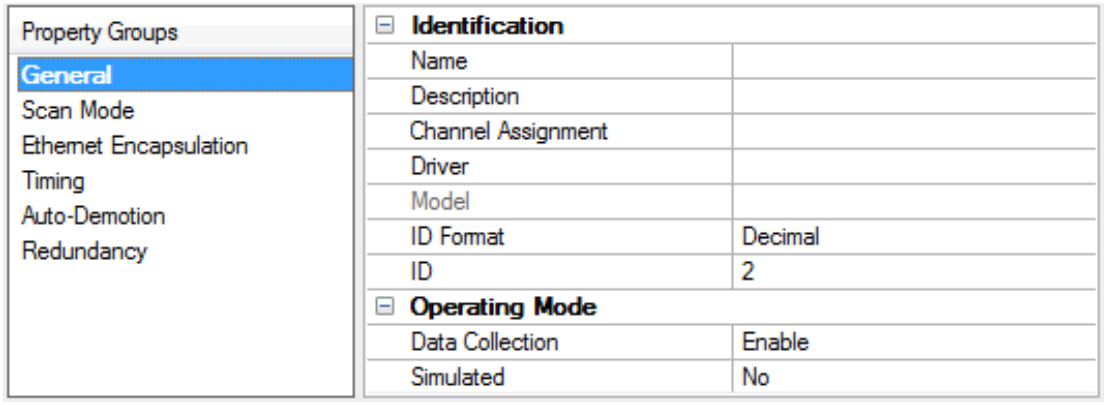
Additional: This text includes optional miscellaneous information.

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 Properties - General

A device represents a single target on a communications channel. If the driver supports multiple controllers, users must enter a device ID for each controller.



Identification

Name: This property specifies the name of the device. It is a logical user-defined name that can be up to 256 characters long, and may be used on multiple channels.

Note: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".
For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.

Description: User-defined information about this device.
Many of these properties, including Description, have an associated system tag.

Channel Assignment: User-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

Model: This property specifies the specific type of device that is associated with this ID. The contents of the drop-down menu depends on the type of communications driver being used. Models that are not supported by a driver are disabled. If the communications driver supports multiple device models, the model selection can only be changed when there are no client applications connected to the device.

Note: If the communication driver supports multiple models, users should try to match the model selection to the physical device. If the device is not represented in the drop-down menu, select a model that conforms closest to the target device. Some drivers support a model selection called "Open," which allows users to communicate without knowing the specific details of the target device. For more information, refer to the driver help documentation.

ID: This property specifies the device's driver-specific station or node. The type of ID entered depends on the communications driver being used. For many communication drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The ID format can be Decimal, Octal, and Hexadecimal.

Note: If the driver is Ethernet-based or supports an unconventional station or node name, the device's TCP/IP address may be used as the device ID. TCP/IP addresses consist of four values that are separated by periods, with each value in the range of 0 to 255. Some device IDs are string based. There may be additional properties to configure within the ID field, depending on the driver. For more information, refer to the driver's help documentation.

Operating Mode

Data Collection: This property controls the device's active state. Although device communications are enabled by default, this property can be used to disable a physical device. Communications are not attempted when a device is disabled. From a client standpoint, the data is marked as invalid and write operations are not accepted. This property can be changed at any time through this property or the device system tags.

Simulated: This option places the device into Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The item's memory map is based on the group Update Rate. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

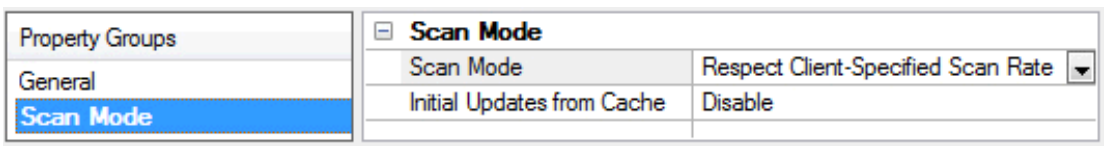
Notes:

1. This System tag (_Simulated) is read only and cannot be written to for runtime protection. The System tag allows this property to be monitored from the client.
2. In Simulation mode, the item's memory map is based on client update rate(s) (Group Update Rate for OPC clients or Scan Rate for native and DDE interfaces). This means that two clients that reference the same item with different update rates return different data.

Simulation Mode is for test and simulation purposes only. It should never be used in a production environment.

Device Properties - Scan Mode

The Scan Mode specifies the subscribed-client requested scan rate for tags that require device communications. Synchronous and asynchronous device reads and writes are processed as soon as possible; unaffected by the Scan Mode properties.



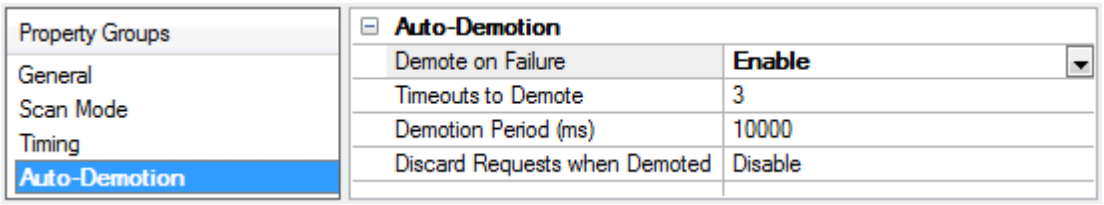
Scan Mode: specifies how tags in the device are scanned for updates sent to subscribed clients. Descriptions of the options are:

- **Respect Client-Specified Scan Rate**: This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate**: This mode specifies the maximum scan rate to be used. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  Note: When the server has an active client and items for the device and the scan rate value is increased, the changes take effect immediately. When the scan rate value is decreased, the changes do not take effect until all client applications have been disconnected.
- **Request All Data at Scan Rate**: This mode forces tags to be scanned at the specified rate for subscribed clients. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
- **Do Not Scan, Demand Poll Only**: This mode does not periodically poll tags that belong to the device nor perform a read to get an item's initial value once it becomes active. It is the client's responsibility to poll for updates, either by writing to the _DemandPoll tag or by issuing explicit device reads for individual items. *For more information, refer to "Device Demand Poll" in server help*.
- **Respect Tag-Specified Scan Rate**: This mode forces static tags to be scanned at the rate specified in their static configuration tag properties. Dynamic tags are scanned at the client-specified scan rate.

Initial Updates from Cache: When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. A device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

Device Properties - Auto-Demotion

The Auto-Demotion properties can temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device offline for a specific time period, the driver can continue to optimize its communications with other devices on the same channel. After the time period has been reached, the driver re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.



Demote on Failure: When enabled, the device is automatically taken off-scan until it is responding again.
Tip: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

Timeouts to Demote: Specify how many successive cycles of request timeouts and retries occur before the device is placed off-scan. The valid range is 1 to 30 successive failures. The default is 3.

Demotion Period: Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

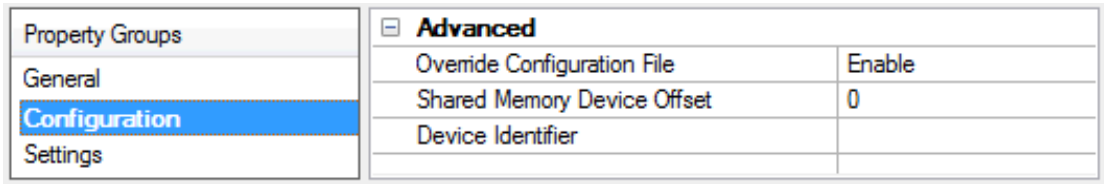
Discard Requests when Demoted: Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard

writes; the server automatically fails any write request received from a client and does not post a message to the Event Log.

Device Properties - Configuration

Advanced

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



Override Configuration File: When enabled, this option overrides the shared memory device offset and device identifier as defined by the configuration file. In the absence of a configuration file, the associated properties may be used to manually define the shared memory device offset and device identifier. The default setting is disabled.

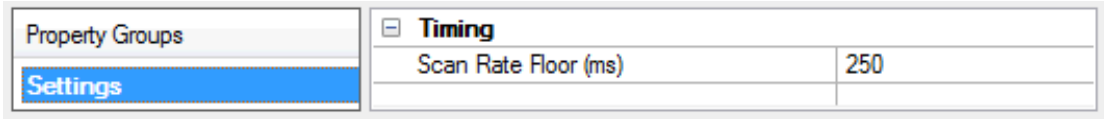
- **Note**: If the Override Configuration File property is disabled after either the shared memory device offset or the device identifier settings have been changed, the settings will return to their default settings. The default settings are the values last loaded from the configuration file.

Shared Memory Device Offset: This property is used to define the starting byte offset into the shared memory map for the given device. All register offsets (such as tag addresses) are relative to the shared memory device offset. If the configuration is designed so 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.

- **Note**: When the Override Configuration File setting is disabled, this property will hold the value last imported during automatic device/tag generation despite being disabled. When the Override Configuration File setting is enabled, this property will be enabled and capable of modification. The range is 0 to 2147483647. The default setting 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. The default setting is 1.

Device Properties - Settings



Timing

Scan Rate Floor: Specify the device's minimum scan rate. 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 250 milliseconds. The maximum value is 60000 milliseconds. The default is 250 milliseconds.

Data Types Descriptions

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.

Data Type	Description
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 registers byte offset relative to the devices 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

8-bit value at byte offset 0
Address = D0
Data Type = Byte

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** property group in **Channel Properties** and enable **Regenerate File**.

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>.

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

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.

Note:
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 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.

Note:
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

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.

Note:
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 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.

Note:
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.

Note:
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

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.

Note:

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

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.

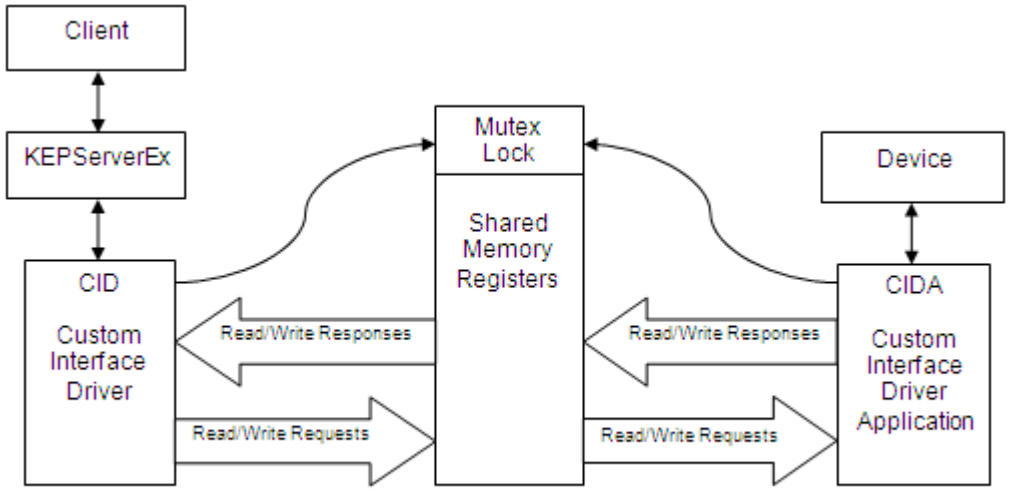


Figure 1 – CID CIDA Overview

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. For more information, refer to **Shared Memory Interface**.
    a. If the Register supports reads, it must be configured as follows.

        ii. Assigns **REGISTER.ReadOffset**.

        iii. Initializes **DATA.STATUS** flags to 0.

        iv. Initializes **VALUE.Type**.

        v. Initializes **VALUE._Value**.

        vi. Initializes **VALUE.ExtSize** if register is a string or an array.

        vii. Reserves **VALUE.ExtSize** bytes starting at **VALUE.ExtValue** for string and array data.

        viii. Initializes **VALUE.ExtValue** for string and array data.

        ix. 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.

        ii. Assigns **REGISTER.WriteOffset**.

        iii. Initializes **DATA.STATUS** flags to 0.

        iv. Initializes **VALUE.Type**.

v. Initializes **VALUE._Value**.

vi. Initializes **VALUE.ExtSize** if register is a string or an array.

vii. Reserves **VALUE.ExtSize** bytes starting at **VALUE.ExtValue** for string and array data.

viii. Initializes **VALUE.ExtValue** for string and array data.

ix. 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:

ii. Caches the write value.

iii. Clears **WriteData.RequestPending** before performing the write.

iv. 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:

ii. Clears **ReadData.RequestPending** before performing the read.

iii. 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. This is usually 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) 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

Data Member	Offset	Size	Description
WORD Error : 1; WORD Reserved : 13; } STATUS			when error condition no longer exists. Reserved.
DWORD ErrorCode	2	4	Vendor-specific error that will be posted in OPC server's Event Log. The 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 *Visual C++ Custom Interface VS2008 Example* 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 } _Value;	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.
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 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 + size of (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:

     a. Boolean
     b. Char
     c. Byte
     d. Short
     e. Word
     f. Long
     g. DWord

h. Float

i. Double

j. Date

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 hard-coded in the sample. The following support information is included:

   a. **Company Name:** My Company

   b. **Phone:** 1-888-555-1212

   c. **Email:** support@mycompany.com

   d. **Configuration Launch Hint:** At the command prompt...

   e. **Runtime Launch Hint:** At the command prompt...

   f. **Help Launch Hint**

   g. **Additional**

9. Output XML in rudimentary fashion (such as 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 CIDAs. 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 peformed 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

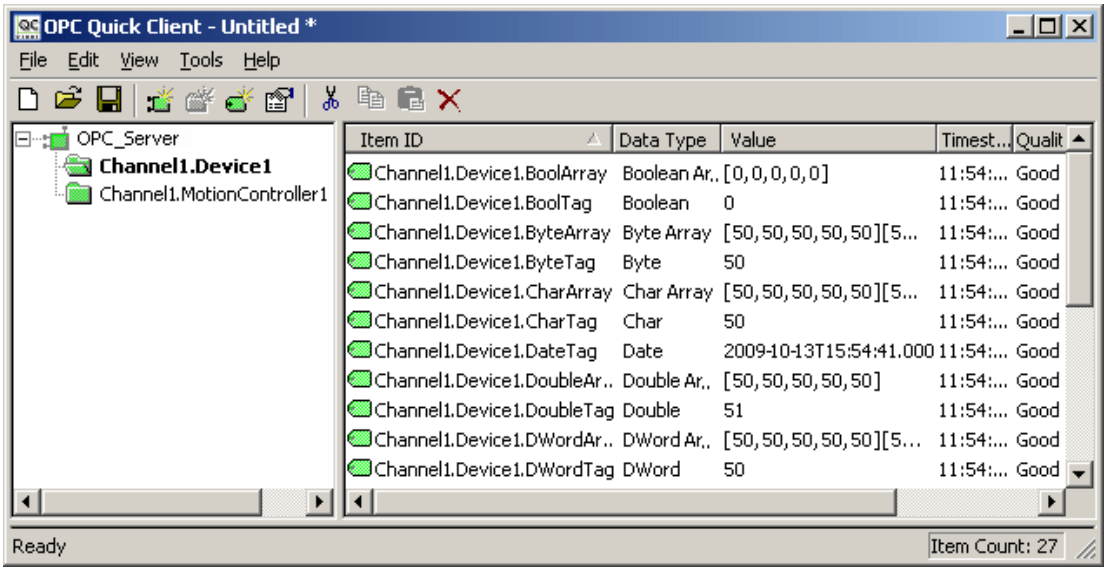
1. If the current tag has its "Write response pending" flag set, the following will occur:
   - The Shared Memory will be updated with the Write result.
   - The Write ResponsePending flag will be set.
2. If the current tag has its "Read response pending" flag set, the following will occur:
   - The 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. Once the CIDA executable file has been created, run the "exportconfig" command line against it to generate the XML configuration file for the server. For example, *<CIDA Filename>.exe -exportconfig*.
4. Run "cidarefimplcpp.exe". A console will appear that says "Enter q to quit."
5. Next, launch the OPC server.
6. In the **Administrator** tool, select **Configuration**. Then, click **File** | **New**.
7. To invoke a new Channel Wizard, click on **Click to add a channel**.
8. Click **Next**. In the drop-down box, select **Custom Interface**.
9. Click **Next**.
10. 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.
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, and that there are no models to choose.
15. Click on the **Settings** property group to display the **Timing** properties. Click on the **Configuration** property group 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". Items will be 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) 1 (Write)	0 (Get) 1 (Set)	0 (Reserved) 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.

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) 1 (Write)	0 (Get) 1 (Set)	0 (Reserved) 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

C

D

E

H

I

M

N

O

R

S

T

U

V

W