AVEVA™ AVEVA Communication Drivers SDK User Guide

User Guide

Publication date: Tuesday, May 9, 2023

Publication ID: 979681

Contact Information

AVEVA Group Limited
High Cross
Madingley Road
Cambridge
CB3 0HB. UK

https://sw.aveva.com/

For information on how to contact sales and customer training, see https://sw.aveva.com/contact.

For information on how to contact technical support, see https://sw.aveva.com/support.

To access the AVEVA Knowledge and Support center, visit https://softwaresupport.aveva.com.

Acknowledgements

Contents

Chapter 1

AVEVA™ Communication Drivers Software Development Kit (SDK) Help

Getting Started with the Communication Drivers SDK

Introduction

The AVEVA™ Communication Drivers SDK provides the sample code and instructions necessary for you to create a Communication Driver (previously called OI Servers or Data Access Servers). The Communication Drivers support OPC and SuiteLink connectivity to link client programs such as HMI, DCS, and SCADA systems to physical hardware devices such as PLCs.

The AVEVA™ Communication Drivers SDK provides a sample project. Sample project is just for reference in developing custom Communication Driver. It should not be modified to extend any new functionality. You have to start with a new Communication Driver to develop your own Communication Driver.

Important: The sample Communication Driver is not meant for production or cannot be used directly.

Included in this guide are:

- Procedures for developing a Communication Driver.
- Procedure to convert a Legacy DAServer to a Communication Driver.
- Utilities like ResMan and PLCPanel.

Licensing

A single instance of the Communication Driver developed using the SDK can be activated and configured in the OCMC without an AVEVA Communication Drivers Pack license. If you want to implement your own licensing scheme, refer to Adding Custom License support.

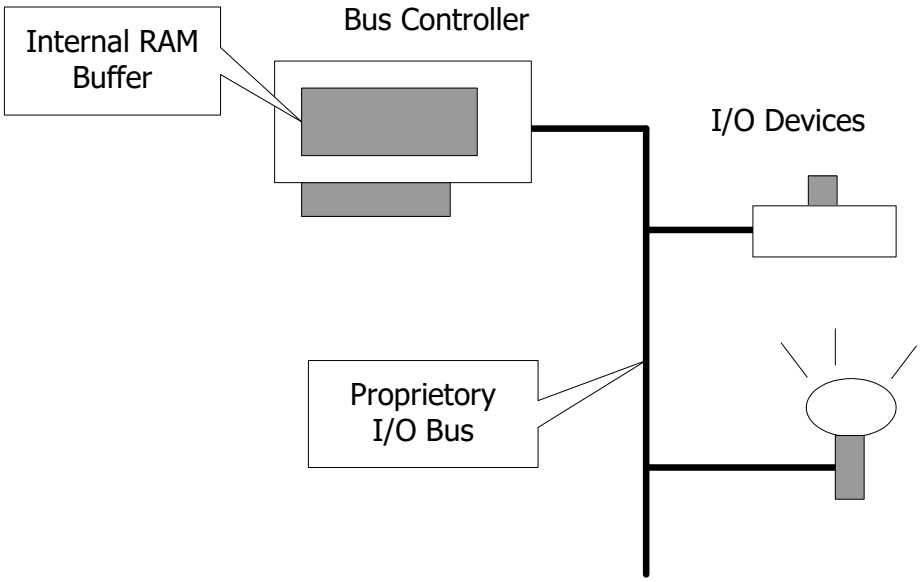
Note: Configuring and activating multiple instances of the Communication Driver requires an AVEVA Communication Drivers Pack Professional License.

Utilities

A Quick Overview of PLCPanel

You can build the sample Communication Driver solution provided by the Communication Drivers SDK to create a Communication Driver. Most of the sample Communication Driver are designed to communicate with a PLC emulator called PLCPanel. PLCPanel supports communication by serial port, TCP, UDP, and shared memory, all at one time. Except for shared memory, all the protocols work on the same computer or a different computer from the Communication Driver. Using shared memory, PLCPanel must run on the same computer as the Communication Driver.

PLCPanel emulates the bus controller for a distributed I/O system. The bus controller communicates with the actual I/O devices, and copies information gathered from the devices into a RAM buffer inside the bus controller.



The figure above shows one version of the bus controller that PLCPanel emulates, communicating with I/O devices over its proprietary I/O bus. This version of the bus controller is designed to plug in to a computer, and share its internal RAM buffer directly as dual port RAM. Other versions of the bus controller (all emulated by PLCPanel, all the time) include one that shares its internal RAM buffer with the host computer using TCP, another that uses UDP, and one that uses a serial port.

Note: The PLCPanel communicates with the COM1 serial port.

At all times, the Communication Driver simply references PLCPanel's internal RAM buffer. It is up to the bus controller (as emulated by PLCPanel) to map the inputs and outputs to the internal RAM buffer. The mapping is controlled by configuring the PLCPanel.

There are four types of I/O devices. Each of the four types inherently supports one or more items. Up to 128 each of Discrete Input, Discrete Output, Analog Input, and Analog Output devices may be configured. Each configured device has a unique name.

Each of the four device types supports a specific set of items. Every device of a specific type supports the same item set, always with the same name, type, and access rights.

Discrete Input (DI), devices support only one item:

- Input, VT_BOOL, Read-Only, Offset+0.

Discrete Output (DO), devices support only one item:

- Output, VT_BOOL, Read/Write, Offset+0.

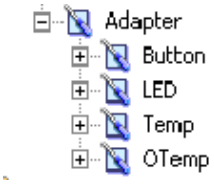
Analog Input (AI), devices support three items:

- Input, VT_R8, Read-Only, Offset+0.
- HighLimit, VT_BOOL, Read-Only, Offset+8.
- LowLimit, VT_BOOL, Read-Only, Offset+9.

Analog Output (AO), devices support three items:

- Output, VT_R8, Read/Write, Offset+0.
- HighLimit, VT_BOOL, Read/Write, Offset+8.
- LowLimit, VT_BOOL, Read/Write, Offset+9.

Each I/O Device must be configured with a base offset into the bus controller's internal RAM buffer. This base offset is referred to as "Offset" in the list above. The individual items supported by each device are found at "Offset" plus a constant, as indicated.



A sample configuration is shown here. In this simple example, the bus controller is named "Adapter", and only one instance of each device type exists. The DI device is named "Button", the DO is named "LED", the AI is named "Temp", and the AO is named "OTemp".

The fully-qualified item names for these items is generated by concatenating the name of the adapter, the name of the device, and one of the valid item names for that device. For the example configuration shown above, some examples are:

- Adapter.Button.Input
- Adapter.Temp.HighLimit
- Adapter.OTemp.Output

This configuration is supported by the installed PLCPanel configuration file; so it is commonly used as a starting point when configuring each of the Communication Drivers in this guide. The following table relates the Communication Driver item names to their base offsets, and the names as configured into PLCPanel.

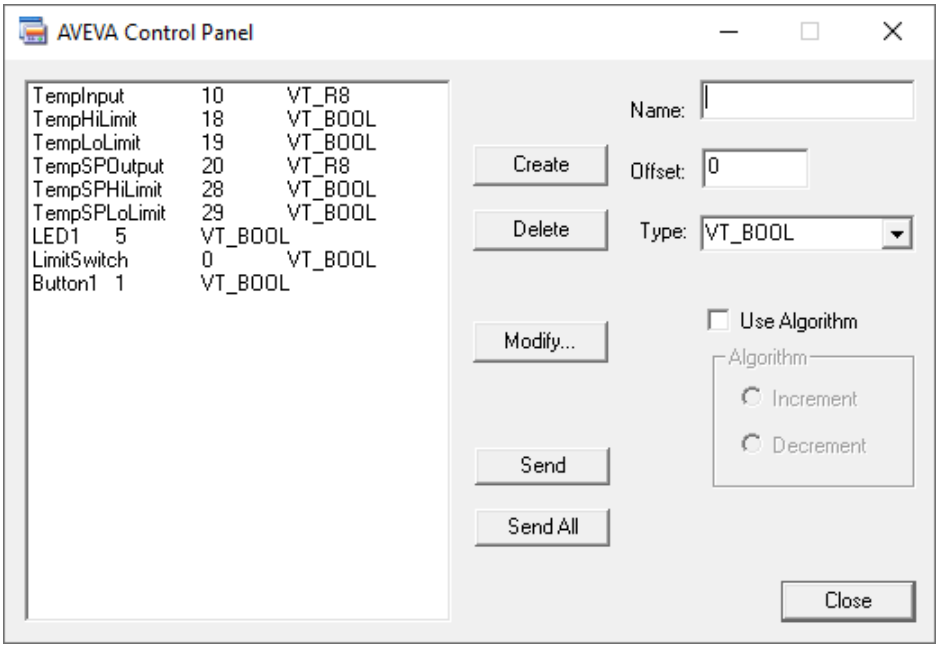
I/O Device	Item Name Adapter	Base Offset	Final Offset	Type	PLCPanel Name
Button		1			
	Button.Input		1	VT_BOOL	Button
LED		5			

I/O Device	Item Name Adapter	Base Offset	Final Offset	Type	PLCPanel Name
	LED.Output		5	VT_BOOL	LED1
Temp		10			
	Temp.Input		10	VT_R8	TempInput
	Temp.HighLimit		18	VT_BOOL	TempHiLimit
	Temp.Lowlimit		19	VT_BOOL	TempLoLimit
OTemp		20			
	OTemp.output		20	VT_R8	TempSPOutput
	OTemp.HighLimit		28	VT_BOOL	TempSPHiLimit
	OTemp.LowLimit		29	VT_BOOL	TempSPLoLimit

PLCPanel is available under:

Sample -> **Utilities** folder

It can be run directly from this location or copied to some other location. If you copy it, be sure to copy its configuration file (PLCItems.dat) and its shared memory file (Sharedmemory.h) into the same location.



Resource Manager Utility

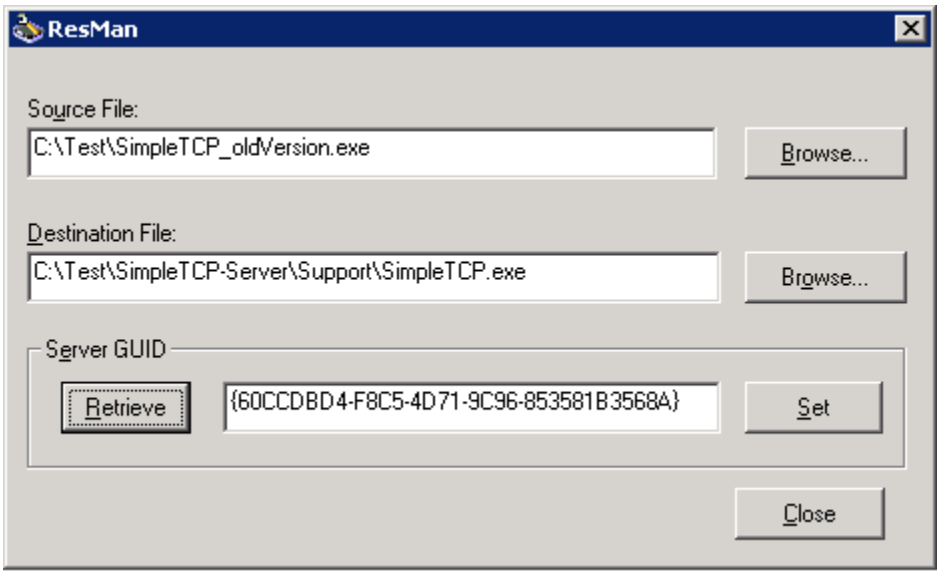
On rare occasions, you may need to examine or modify the GUID of a server shell .exe Communication Driver module. Use the Resource Manager (ResMan) utility to do this.

Note: Modifying a module's GUID should be done with great care because the GUID embedded in a module must remain in sync with the GUID inserted in associated code for the Communication Driver to operate properly.

ResMan can be used for the following purposes:

- To examine the GUID of an existing server shell .exe container module (for instance, an EXE file that controls DLL files).
- To generate a server shell .exe container for a specific Communication Driver with a new GUID.
- To insert a new GUID value into an existing server shell .exe container module.

Run ResMan by clicking **ResMan.exe** from th Utility folder. A dialog box appears in which you can access all of the program's functions.



Source File specifies the filename of the "template" container module that is to be copied to generate the new server shell .exe container module. If you are examining the GUID of an existing server shell .exe container module, use this element to specify the file that contains the GUID to be examined.

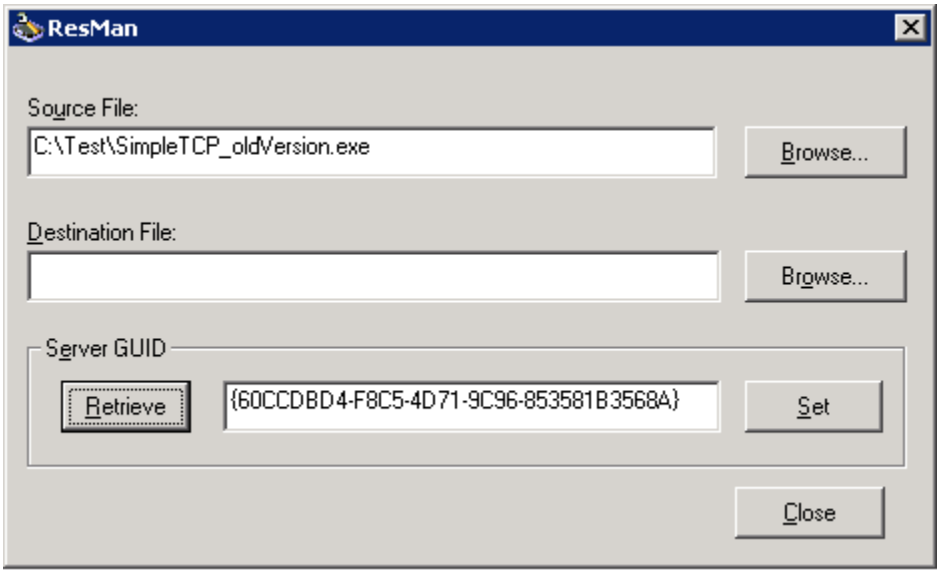
Destination File specifies the filename of a server shell .exe DAServer module. If you are inserting a new GUID value into an existing server shell .exe container module, Destination File is an existing file into which a new GUID will be set. When inserting a new GUID into an existing server shell the Source File field should be empty.

Use the **Browse** buttons to navigate to filenames that already exist or to a folder location for a new filename.

Click **Retrieve** to extract the GUID from the file specified in the **Source File** field and display it in the **Server GUID** box. The **Retrieve** button is disabled when **Source File** is empty.

Note: If you choose to edit the GUID displayed, it must remain in valid "Registry format" ({XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}).

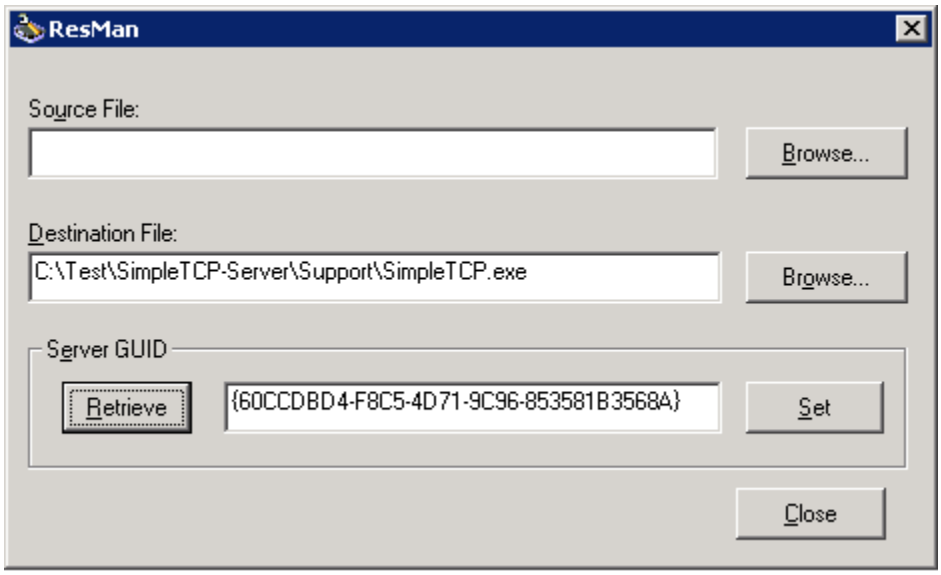
The following figure shows the result of typing a valid name in **Source File** and clicking **Retrieve**. For retrieve operation , clear **Destination File** text box.



The **Set** button is enabled only if both the **Server GUID** text box contains a validly formatted GUID and **Destination File** is not empty. Press **Set** to do one of the following:

1. Copy the source file into the destination file (if you are generating a server shell .exe container for a specific Communication Driver with a new GUID).
2. Set the GUID in the **Server GUID** box into the newly copied file (if you are generating a server shell .exe container for a specific DAServer with a new GUID or inserting a new GUID value into an existing server shell .exe container module).

   The following figure shows the result of entering a valid name in **Destination File**. Note that the **Set** button is enabled. For Set Operation, clear **Source File** text box.



3. Click **Close** to close the ResMan program.

Note: If you are using the ServerExe.exe shell and Resource Manager utility to update an existing shell, then you must embed a manifest into the resulting executable after resman has embedded the server GUID. To do this you must run the following command from the Visual Studio 2008 command prompt: "mt.exe -manifest

serverexe.exe.manifest -outputresource:<ServerName>.exe;1," where <ServerName> is the output of the resman utility after the GUID is embedded.

Using the Communication Drivers SDK to Create a Driver

SDK Communication Driver Creation Workflow

You can create a new solution in Visual Studio and compile a working (but empty) Communication Driver by following these steps. Once the solution is created, you can implement your own protocol as per your Communication Driver requirements.

The following are the high level steps to create a Communication Driver:

1. Setting up the Communication Drivers SDK
2. Developing a Communication Driver
3. Adding a New Configuration Parameter

Prerequisites

- Microsoft Visual Studio 2022

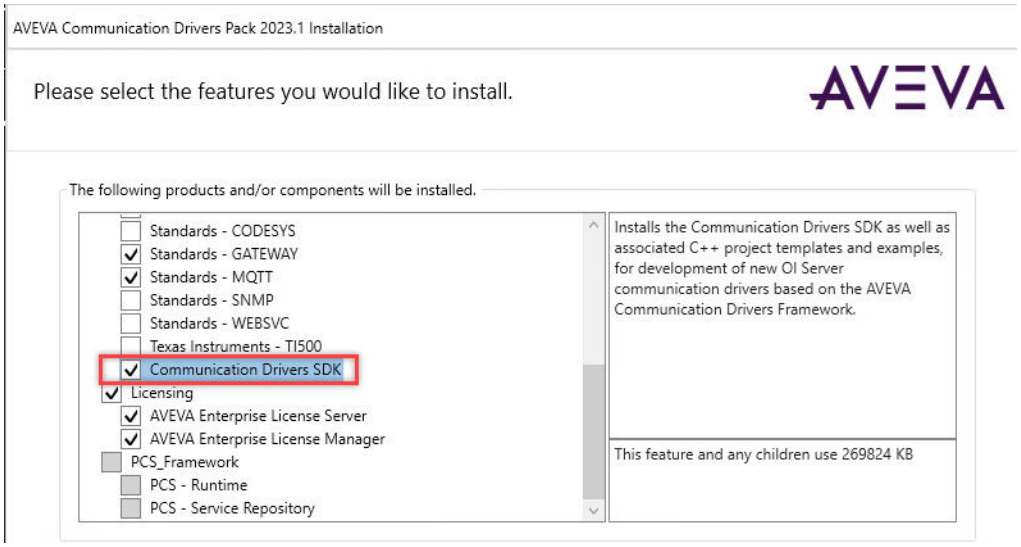
  Visual Studio 2022 components:

  - Desktop Development with C++
  - Visual C++ MFC x86 and x64
  - Windows 10 SDK (10.0.20348.0)
- Windows Installer XML (WiX) v3.11
- WiX Toolset Visual Studio 2022 Extension

Setting up the Communication Drivers SDK

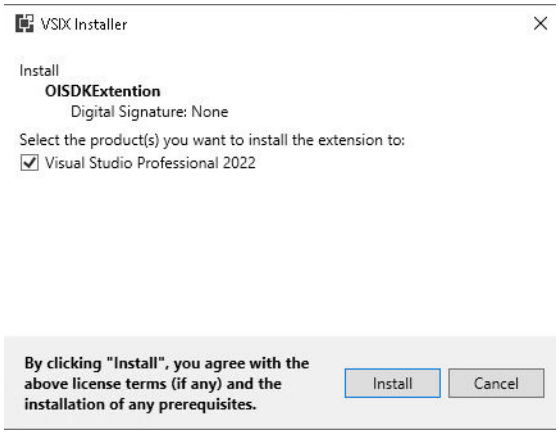
To set up the Communication Drivers SDK in Visual Studio 2022

1. Ensure that you select Communication Drivers SDK during the CDP 2023.1 installation.



2. Go to **C:** > **Program Files(x86)** > **Wonderware** > **OI-Server** > **OI-SDK** > **Bin**.
3. Double-click the **OISDKExtention.vsix**.

   The **VSIX Installer** appears.
4. Select **Visual Studio Professional 2022** as the product you want to install the extension to.

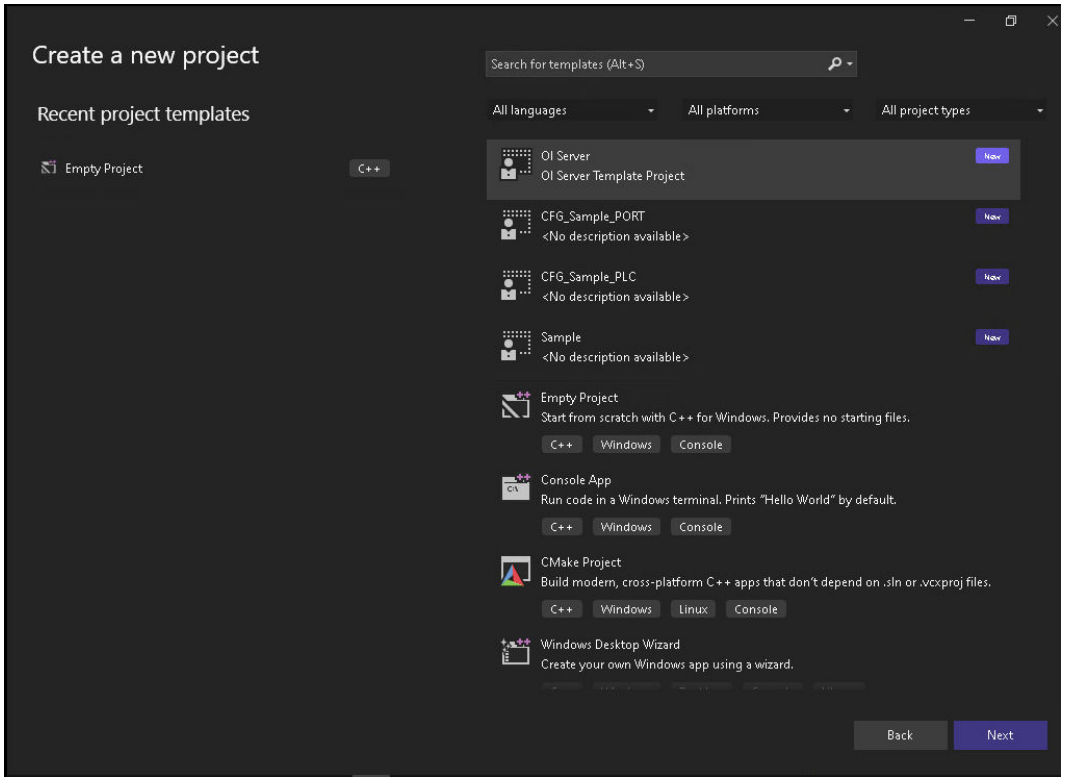


After the extension is installed, you can develop your own Communication Driver. Refer to the section below on 'Developing a Communication Driver' for more details.

Developing a Communication Driver

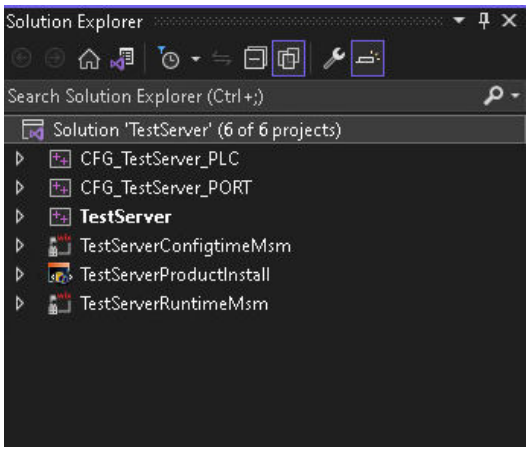
1. Open Visual Studio 2022 as administrator.
2. Click **File** > **New** > **Project.**

   The **New Project** window appears.
3. In the **New Project** window, under **Installed** > **Visual C++**, select **OI Server** as a project type.

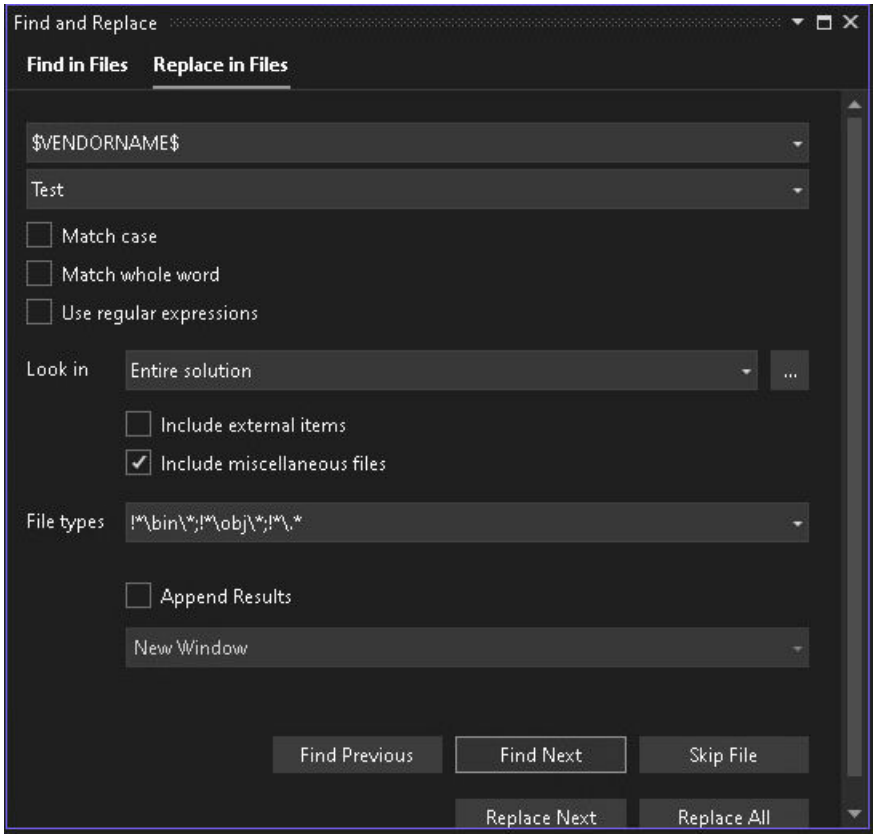


4. In the **Name** field, enter the name of the Communication Driver that you want to create. Ensure that the name:
   - does not exceed 15 characters.
   - does not contain any special characters like _, $, #, @, etc.
   - is not prefixed with 'OI'.
   - does not have any spaces.
5. In the **Location** field, click **Browse** to select the location where you want to save the output folder.
6. Click **OK**.

   In the Solution Explorer, an empty but working solution of the Communication Driver is created.



7. Find and replace all instances of **$VENDORNAME$** in the entire solution with the correct name of the vendor of the Communication Driver and save the solution.

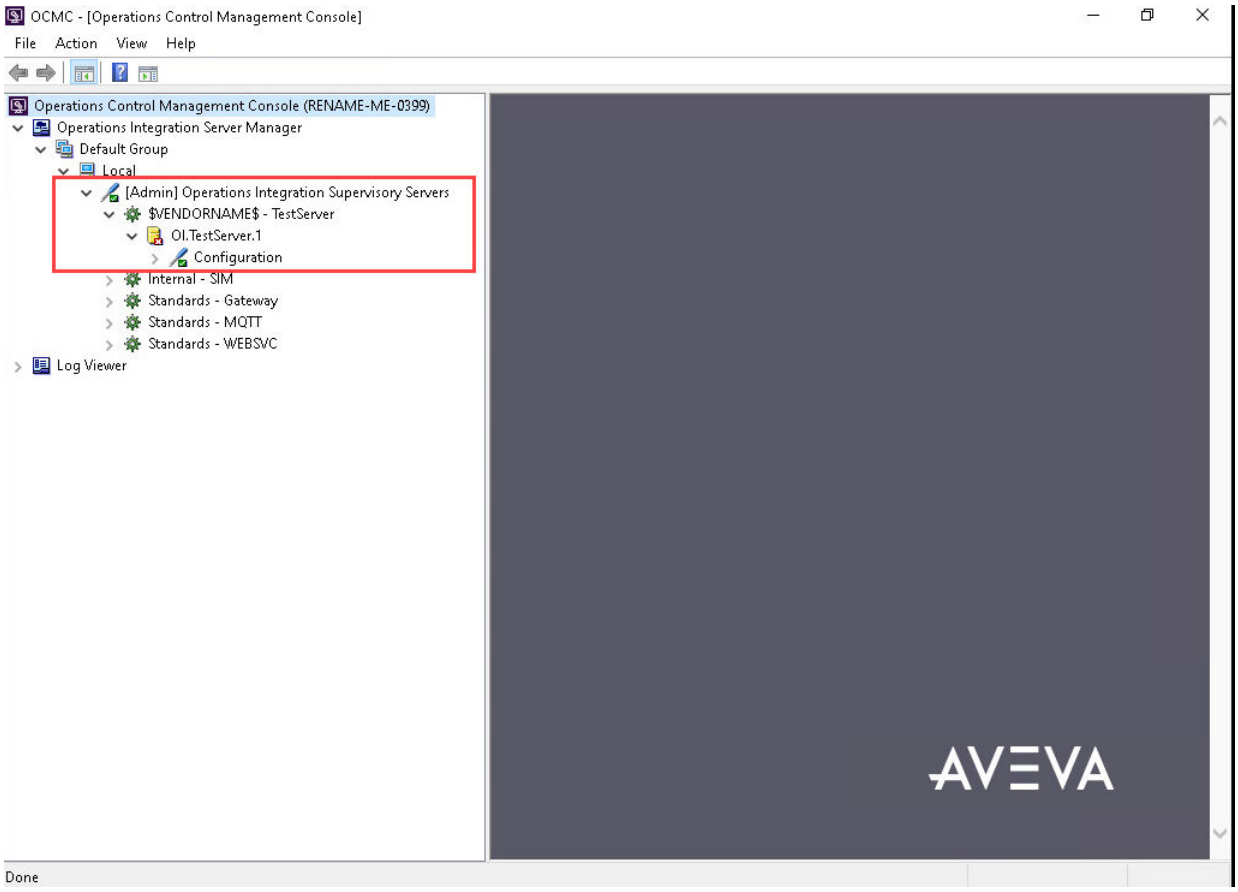


8. To replace the default splash screen of the Communication Driver, go the location where you have saved the output folder (refer to step 5), and open **<Communication Driver>** > **<Communication Driver>ProductInstall** > **Support** and replace the **SplashScreen.bmp** file with your own splash screen. This is an optional step and can be followed only if you want the splash screen to contain the branding of your Communication Driver.
9. In release mode, right-click on each project in the Solution Explorer and click **Build**. Ensure that you build the projects in the following order:
   a. <ServerName>
   b. CFG_<ServerName>_PORT
   c. CFG_<ServerName>_PLC
   d. <ServerName>ConfigtimeMsm
   e. <ServerName>RuntimeMsm
   f. <ServerName>ProductInstall
10. After all the project builds are completed, open the output folder in the location that you had provided while creating the new project. Go to **Stage** > **Install** and copy the **CD-<Communication Driver>** folder on the machine where you want to install the Communication Driver.

Note: The Communication Driver can be installed only on a machine which has Communication Drivers Pack 2023.1 or above.

11. Double-click and run the **Setup-OI-<CommunicationDriver>.msi** file and click **Install**.

    You can see the newly installed Communication Driver in OCMC with the default configuration options.



    If you want to add additional configuration parameters, refer to the section below on 'Adding a New Configuration Parameter.'

Adding a New Configuration Parameter

To add a new configuration parameter

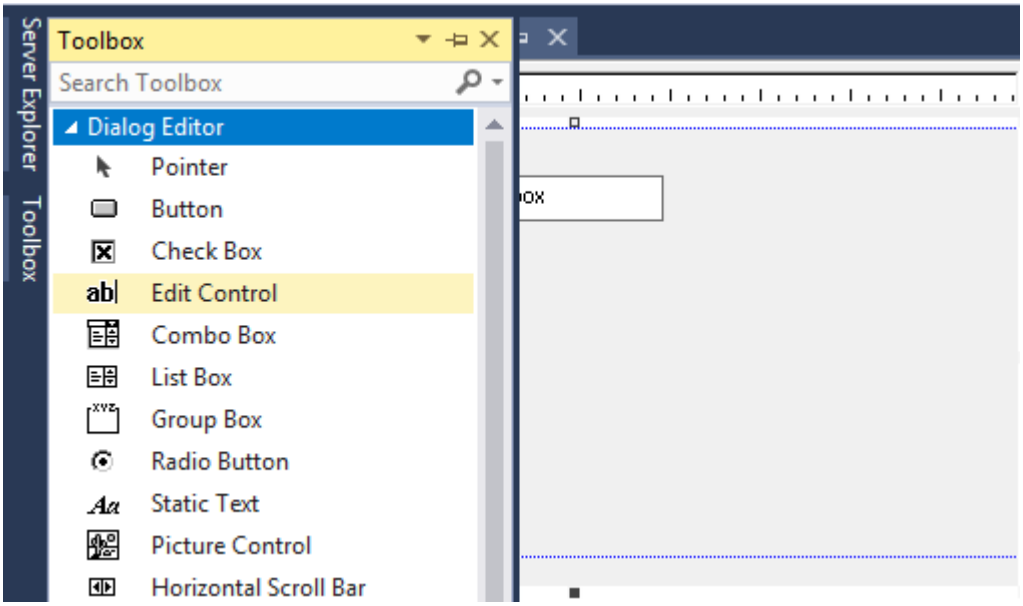
By default, Communication Drivers SDK provides IP Address and Port Number configuration. In order to add a new configuration parameter (for example, Reply Timeout), follow these steps:

1. Update <ServerName>.aaRUL file: Add Reply Timeout entry in aaRUL file under HIERARCHYNODE type PLC:

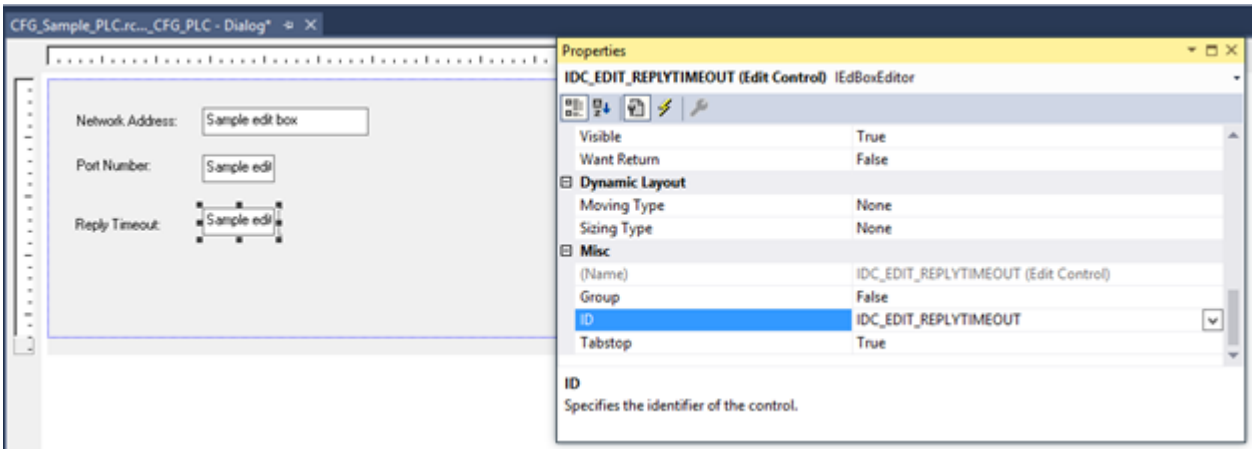
```
<ReplyTimeout>
<PROPERTYTYPE>VT_UI2</PROPERTYTYPE>
<DEFAULTVALUE>15</DEFAULTVALUE>
<PROPERTYMIN>1</PROPERTYMIN>
<PROPERTYMAX>300</PROPERTYMAX>
<PROPERTYEDITHEADER> ReplyTimeout </PROPERTYEDITHEADER>
<PROPERTYEDITUNIT/>
```

```
<PROPERTYEDITHELP> ReplyTimeout </PROPERTYEDITHELP>
</ReplyTimeout >
```

2. Update configuration editor:
   a. Open IDD_CFG_PLC dialog from CFG_Sample_PLC project
   b. Drag and drop Static and Edit controls to UI from ToolBox



   c. Rename Static control text to Replay Timeout and Edit control ID to IDC_EDIT_REPLYTIMEOUT



   d. Add the following attribute definitions in CCfgPLC class private section in CFG_PLC.h file

```
DWORD m_hReplyTimeout;
CComBSTR m_bstrReplyTimeout;
CComBSTR m_bstrReplyTimeout _Applied;
CComBSTR m_bstrReplyTimeout_Min;
CComBSTR m_bstrReplyTimeout_Max;
```

e. Add COMMAND_HANDLER entry in MessageMap section of CCfgPLC class in CFG_PLC.h file

```
COMMAND_HANDLER(IDC_EDIT_PORT, EN_CHANGE, OnChangeReplyTimeout)
```

f. Declare OnChange command handler in CCfgPLC class public section in CFG_PLC.h file

```
LRESULT OnChangeReplyTimeout (WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
```

g. Define OnChange command handler in CCfgPLC class public section in CFG_PLC.cpp file

```
LRESULT

CCfgPLC::OnChangeReplyTimeout(WORD /*wNotifyCode*/, WORD /*wID*/, HWND /*hWndCtl*/,
BOOL& /*bHandled*/)

{

SetHostDirty();

return 0;

}
```

h. Initial property values in OnInitialize method of CCfgPLC class in CFG_PLC.cpp file

```
hr = InitializeValues(m_hReplyTimeout, m_bstrReplyTimeout, m_bstrReplyTimeout_Min,
m_bstrReplyTimeout_Max);

if (hr != S_OK)

return hr;
```

i. Initialize applied variables in OnInitialize method of CCfgPLC class in CFG_PLC.cpp file

```
m_bstrReplyTimeout_Applied = m_bstrReplyTimeout;
```

j. Disable the edit control for read-only mode in OnInitialize method of CCfgPLC class in CFG_PLC.cpp file

```
if( m_bReadOnly )

{

//Disable the Reply Timeout edit box.

//TODO: Similar block of code for each property.

HWND hwndReplyTimeout = GetDlgItem(IDC_EDIT_REPLYTIMEOUT);

CWindow wReplyTimeout;

ReplyTimeout.Attach(hwndReplyTimeout);

wReplyTimeout.EnableWindow(FALSE);

}
```

k. Capture 'applied' version Reply Timeout property in OnRestore method of CCfgPLC class in CFG_PLC.cpp file

```
m_bstrReplyTimeout = m_bstrReplyTimeout_Applied;
```

l. Close handle for Reply Timeout property in OnClose method of CCfgPLC class in CFG_PLC.cpp file

```
CheckResult(m_PackageSite->CloseAttrHandle(m_hReplyTimeout));
```

m. Read from package and write value to dialog control in DDX method (bUpdate True case) of CCfgPLC class in CFG_PLC.cpp file

```
SetDlgItemText(IDC_EDIT_REPLYTIMEOUT, W2T(m_bstrReplyTimeout.m_str));
```

n. Read from dialog control for the package in DDX method (bUpdate False case) of CCfgPLC class in CFG_PLC.cpp file

```
GetDlgItemText(IDC_REMOTE_IPADDR, m_bstrReplyTimeout.m_str);
```

o. Save configured value into configuration file in OnApply method of CCfgPLC class in CFG_PLC.cpp file

```
// ReplyTimeout
AttrVar.Clear();
AttrVar.vt = VT_BSTR;
AttrVar.bstrVal = m_bstrReplyTimeout.Copy();
if (!SetAttribute(m_hReplyTimeout, AttrVar))
return E_FAIL;
m_bstrReplyTimeout_Applied = m_bstrReplyTimeout;
```

3. Update Communication Driver runtime: Use GetProperty method to read the configuration parameter from runtime project <ServerName>PLC constructor in SampleHierarchy.cpp as shown below for Modicon Communication Driver as an example

```
ModiconPLC::ModiconPLC(IOTHANDLE hBaseObject) :
SvIoHierarchy(hBaseObject)
{
m_HostName = GetProperty(L"PLC", L"NetworkAddress", wstring(TEXT("")));
m_usPortNumber = (u_short)GetProperty(L"PLC", L"Port", 1234);
m_ReplyTimeout = (u_short)GetProperty(L"PLC", L"ReplyTimeout", 30);
}
```

Converting a Legacy DAServer to a Communication Driver

Comparing Legacy DAServers with Modern Communication Drivers

In the initial releases, the I/O server was termed as DAServer. Later it was named OI Server, and now it is called a Communication Driver. The main difference between a DAServer and an OI Server is that an OI Server supports multi-instance. There are no significant functional differences between an OI Server and a Communication Driver.

To Convert DAServer to an OI Server

1. Upgrade DAServer projects to VS2022.
2. Change _WIN32_WINNT version to 0x0500 in all DAServer projects.
3. Update DASToolkit libraries/include files with the latest SDK files provided.
4. Add TagListUtil.lib dependency to Communication Driver runtime project.
5. Change GetModuleName() implementation in DAServer runtime.

   See Get Module name change .
6. Remove const for CLSID CLSID_CIosActivate = DASSim::CLSID_IosActivate;
7. Insert DAServer CLSID to new ServerExe.exe using ResMan.
8. Find and replaceDAServer ProgId to OI.<ServerName> (Do not use DAS in ProgId name, for example, ArchestrA.DASSim to OI.SIM).
9. Change aaCfg, aaRGul file installation path to Operations Integration Supervisory Servers location in RuntimeMSM as shown in below example:

   See Change aaRUL and aaCFG installation path (Example).
10. Comment <ServerName>.exe.reg5, <ServerName>.exe.reg4 and <ServerName>.exe.reg3 entries in RuntimeMSMReg file.
11. Go to **C:** > **Program Files(x86)** > **Wonderware** > **OI-Server** > **OI-SDK** > **SamplesAndVideos.zip** > **DAS2OIConversion**. Copy the **OIServerProductInstall.wsx** file to the ProductInstall folder. Rename "**OIServer**" in the file name with the actual name of the server. Open the file in VS2022 and update product name, version, and GUIDs appropriately.

Get Module name change

Copy and paste the following section:

```
STDAPI DllServerGetClassName(wchar_t *);
LPCSTR SvIoServer::GetModuleName(void)
{
        static char classname[100] = {0};
        if (classname[0] == 0)
        {
                wchar_t name[100];

                DllServerGetClassName(name);
                wcstombs(classname, name, 100);
        }
        return classname;
}
```

Change aaRUL and aaCFG installation path (Example)

```
    <Directory Id="CommonAppDataFolder" Name="CommonAppDataFolder">
          <Directory Id="VendorFolder" Name="Wonderware">
            <Directory Id="DAServerFolder" Name="OI-Server">
              <Directory Id="DAServerPAKFolder" Name="$Operations Integration
Supervisory Servers$">
```

```
              <Directory Id="DAServerNameFolder" Name="OI.SIM">
                <Directory Id="BaseInstanceFolder" Name="OI.SIM">
                  <Component Id="DASSim.AArul"
Guid="35742EC4-38F7-43CA-9DFE-002169FD8A2F">
                    <File Id=" DASSim.AArul" Name=" DASSim.AArul"
Source="..\..\Support\ DASSim.AArul" KeyPath="yes" />
                  </Component>

                  <Component Id=" DASSim.AAcfg" Guid="4F8C2DE3-EBAE-480A-
AA9A-19E3EC336DD5" Permanent="yes" NeverOverwrite="yes">
                    <File Id=" DASSim.AAcfg" Name=" DASSim.AAcfg"
Source="..\..\Support\Configuration\ DASSim.AAcfg" KeyPath="yes" />
                  </Component>

                </Directory>

                <Component Id="OI.SIM.1" Guid=" NEW GUID " Permanent="yes"
NeverOverwrite="yes">
                  <File Id="OI.SIM.1" Name="OI.SIM.1" Source="..\..\Support\OI.SIM.1"
KeyPath="yes" />
                </Component>
              </Directory>
              </Directory>
            </Directory>
          </Directory>
          </Directory>
```

To follow the step-by-step process of converting a DAServer to an OI Server, refer to the **Converting DAServer to OI Server.mp4** video in the **C:** > **Program Files(x86)** > **Wonderware** > **OI-Server** > **OI-SDK** > **SamplesAndVideos.zip** > **Videos** folder.

Note:
- The video was recorded in VS2017, but the procedure remains the same in VS2022.
- In the video, you may see the folder name as DAS to OI Conversion 30 instead of DAS2OI Conversion.
- The document for the conversion procedure has been added to the the "Converting a Legacy DAServer to a Communication Driver" section in the AVEVA™ Communication Drivers SDK Help.

You can view the output folder of a converted OI Server by clicking **C:** > **Program Files(x86)** > **Wonderware** > **OI-Server** > **OI-SDK** > **SamplesAndVideos.zip** > **DAS2OIConversion** > **SAMPLE-Server**.

Adding Custom License support

To add Custom License support:

1. Add DeviceEngine.lib dependency to Communication Driver runtime project
2. Include DeviceEngine.h, CustomLicenseMgr.h and CustomLicenseMgr.cpp in Communication Driver runtime project
3. Derive Communication Driver runtime from SvIoCustomLicenseServer class

   Ex: class SAMPLEServer : public SvIoCustomLicenseServer
4. Implement `IsLicensed()` and `GetDemoTimeout()` functions.

   See Add IsLicensed and GetDemoTimeout below.

Add IsLicensed and GetDemoTimeout (Example):

1. Declare IsLicensed() and GetDemoTimeout() methods in <ServerName>Class.h

```
virtual bool IsLicensed(bool* tolog, wstring* feature, wstring* version);

virtual DWORD GetDemoTimeout();
```

2. Define IsLicensed() and GetDemoTimeout() methods in <ServerName>Hierarchy.cpp

```
bool SAMPLEServer::IsLicensed(bool* tolog, wstring* feature, wstring* version)
   {
           bool bLicensed = true;
        // TODO - take server-specific action
           return bLicensed;
   }
   DWORD SAMPLEServer::GetDemoTimeout()
   {
           DWORD DemoTime = 120;   // Demo time = 120 mins

           // TODO - take server-specific action
           return DemoTime;
   }
```

To follow the step-by-step process of adding custom licensing, refer to the **OISDK_CustomLicense_Support.mp4** video in the **C:** > **Program Files(x86)** > **Wonderware** > **OI-Server** > **OI-SDK** > **SamplesAndVideos.zip** > **Videos** folder.

Note:
- The video was recorded in VS2017, but the procedure remains the same in VS2022.
- In the video, you may see the folder name as Toolkit instead of SDK.

Documenting the Help of the Communication Driver

It is recommended to document the Help of the Communication Driver for your reference. This section explains the general workflow to configure a Communication Driver in the OCMC, and provides guidelines on how to document controller-specific information of the Communication Driver.

General Workflow

This section outlines the general procedures required to configure a Communication Driver once it is installed in the OCMC.

1. Configure your Communication Driver for run-time. Refer to 'Configuring Your OI Server' in the AVEVA™ Communication Drivers Pack User Guide for more information on each of the following steps:
   a. Add a new PLC connection.
   b. Rename the PLC connection.
   c. Instantiate data sources (requires an AVEVA Communication Drivers Pack Professional License.)
   d. Enable/Disable a PLC connection.
   e. Configure parameters of the PLC (Network Address and Port Number).
   f. Archive configuration sets.

g. Reset a PLC connection.

h. Delete a PLC connection.

2. Configure the global parameters in the **Configuration** node.

   Refer to 'Configuring Global Parameters' in the AVEVA™ Communication Drivers Pack User Guide for more information on each of the global parameters.

3. Configure the device groups. Refer to 'Managing Device Groups' in the AVEVA™ Communication Drivers Pack User Guide more information on each of the following steps:

   a. Add a device group.

   b. Rename a device group.

   c. Modify the update interval.

   d. Delete a device group.

4. Configure the device items. Refer to 'Managing Device Items' in the AVEVA™ Communication Drivers Pack User Guide for more information on each of the following steps:

   a. Add a device item.

   b. Rename a device item.

   c. Set the item reference.

   d. Export and import CSV files.

   e. Delete a device item.

   f. Clear all device items.

5. Add a redundant device (optional).

   Refer to 'Configuring Device Redundancy' in the AVEVA™ Communication Drivers Pack User Guide for more information.

6. Activate/Deactivate your Communication Driver.

   Refer to 'Managing Your OI Server' in the AVEVA™ Communication Drivers Pack User Guide for more information.

7. Troubleshoot any issues using the **Diagnostics** node and Log Viewer.

   Refer to 'Troubleshooting' in the AVEVA™ Communication Drivers Pack User Guide for more information.

Guidelines for Documenting the Help of Your Communication Driver

The configuration parameters for each Communication Driver varies according to the PLC you are connected to. Following are some guidelines to help you document controller-specific information:

- Ensure that you have access to your device manufacturer's documentation in order to configure controller-specific information such as IP addresses, register numbers, and so on.
- Refer to the AVEVA Communication Drivers Pack Help, since the Communication Drivers Pack supports controllers from popular manufacturers such as Allen Bradley, Schneider Electric, Mitsubishi, Modicon, General Electric, and many others. In case your device manufacturer is supported by the Communication Drivers Pack, you may find the required controller-specific details in the Help.

To view the AVEVA Communication Drivers Pack Help:

a. Open the AVEVA Help Viewer by:

   - Searching for **AVEVA Help** in your desktop menu

   OR

   - Typing **https://localhost:5000/** in your desktop browser

b. Click **AVEVA Communication Drivers** to view the AVEVA Communication Drivers Pack Help .

   Note: The AVEVA Help Viewer and Communication Drivers Help is installed when you install the AVEVA Communication Drivers Pack.

- Save the document in your preferred output format (PDF/Word document/Text file).
- Save the document in an easily accessible location. If multiple users have access to your Communication Driver, it is recommended to save the document in a shared location.

AVEVA

Chapter 2

AVEVA™ TCP Communication Driver Sample Help

Getting Started with TCP Communication Driver

This document describes the technical specifications and configuration options for the TCP Communication Driver.

Important: TCP Communication Driver is a sample Communication Driver not meant for production or can not be used directly.

Introduction

The TCP Communication Driver implements a single hierarchy type that communicates to the PLCPanel using the TCP protocol. This server uses the DAS Toolkit library TCPIPEngine.lib. This server demonstrates the implementation for a simple protocol. Adapter and device hierarchies such as Discrete Input (DI), Discrete Output (DO), Analog Input (AI), and Analog Output (AO) can be configured using the OI Server Manager.

There are four types of I/O devices. Each of the four types inherently supports one or more items. Up to 128 each of DI, DO, AI, and AO devices may be configured. Each configured device has a unique name. Each of the four device types supports a specific set of items. Every device of a specific type supports the same item set, always with the same name, type, and access rights.

- Discrete Input (DI), devices support only one item:
  - Input, VT_BOOL, Read-Only, Offset+0.
- Discrete Output (DO), devices support only one item:
  - Output, VT_BOOL, Read/Write, Offset+0.
- Analog Input (AI), devices support three items:
  - Input, VT_R8, Read-Only, Offset+0.
  - HighLimit, VT_BOOL, Read-Only, Offset+8.
  - LowLimit, VT_BOOL, Read-Only, Offset+9.
- Analog Output (AO), devices support three items:
  - Output, Float, Offset+9.
  - HighLimit, Discrete, Offset+8.

- LowLimit, Discrete, Offset+9.

Configuring the TCP Communication Driver

The TCP Communication Driver can connect to PLCs. These connections are modeled in the hierarchy by means of DI, DO, AI, and AO objects, each of which models the end-point of the communications path.

From the ADAPTER_TCPIP branch of the Communication Driver hierarchy, create the new DI, DO, AI, and AO objects.

Adding and Configuring Adapter TCP/IP Connection

Adding Adapter TCP/IP Connection

The server-specific configuration portion of the TCP/IP Communication Driver hierarchy tree under the OI Server Manager starts at the Channel Selector object. This object lets you set server parameters for communication with agents (devices) in the hierarchy tree.

See Determining the Hierarchical Structure for more information about setting up your device hierarchy.

To add a ADAPTER_TCPIP connection to your TCP hierarchy

- In the console tree, right-click **Configuration** and then click **Add ADAPTER_TCPIP Connection**.

  The "New_ADAPTER_TCPIP_000" Parameters view is displayed.

  Edit the object name to appropriately describe components of your specific hardware environment. If you do not rename the object at this time, a numeric sequencing system is applied. You can rename the hierarchy entry later.

Configuring Adapter TCP/IP Connection

To configure the ADAPTER_TCPIP connection

1. In the **Host Name** field, enter the IP address or host name of the PLCPanel.
2. In the **Connection Timeout** field, enter a value, in milliseconds, beyond which a pending request to initiate a connection times out.

   Allowable range is 100 to 300,000 milliseconds.

   The default value is 10,000 milliseconds.
3. In the **Reply Timeou**t field, enter a value, in milliseconds, beyond which messages time out.

   Allowable range is 100 to 300,000 milliseconds.

   The default value is 2000 milliseconds. If you decrease this value, the TCP Communication Driver reacts faster to a communications failure.

Adding and Configuring DI Connection

Adding DI Connection

To add a DI connection to your TCP hierarchy

- In the console tree, right-click the **ADAPTER_TCPIP** object, and then click **Add DI Connection**. The **New_DI_000** object and associated **Parameters** configuration view appear.

Configuring DI Connection

A Discrete Input (DI) device has a name and an offset within the adapter. The only item allowed by the DI device is 'input' offset +0. The only item supported by the DI devices is:

- Input, VT_BOOL, Read-Only, Offset+0.

To configure a DI connection:

1. Enter the **DI Module Offset**.
2. The **'Input' Offset** field is disabled as only one item is supported.

Adding and Configuring DO Connection

Adding DO Connection

To add a DO connection to your TCP hierarchy

- In the console tree, right-click the **ADAPTER_TCPIP** object, and then click **Add DO Connection**. The **New_DO_000** object and associated **Parameters** configuration view appear.

Configuring DO Connection

A Discrete Output (DO) device has a name and an offset within the adapter. The only item allowed by the DO device is 'output' offset +0. The only item supported by the DO devices is:

- Output, VT_BOOL, Read/Write, Offset+0.

To configure a DO connection:

1. Enter the **DO Module Offset**.
2. The **'Output' Offset** field is disabled as only one item is supported.

Adding and Configuring AI Connection

Adding AI Connection

To add an AI connection to your TCP hierarchy

- In the console tree, right-click the **ADAPTER_TCPIP** object, and then click **Add AI Connection**. The **New_AI_000** object and associated **Parameters** configuration view appear.

Configuring AI Connection

An Analog Input (AI) device has a name and an offset within the adapter. The AI, devices support three items:

- Input, VT_R8, Read-Only, Offset+0.
- HighLimit, VT_BOOL, Read-Only, Offset+8.
- LowLimit, VT_BOOL, Read-Only, Offset+9.

To configure a AI connection:

1. Enter the **AI Module Offset**.
2. The **'Input' Offset**, **'HighLimit' Offset**, and **'LowLimit' Offset** fields are disabled as the default values are already entered.

Adding and Configuring AO Connection

Adding AO Connection

To add an AO connection to your TCP hierarchy

- In the console tree, right-click the **ADAPTER_TCPIP** object, and then click **Add AO Connection**. The **New_AO_000** object and associated **Parameters** configuration view appear.

Configuring AO Connection

An Analog Output (AO) device has a name and an offset within the adapter. The AO, devices support three items:

- Output, Float, Offset+9.
- HighLimit, Discrete, Offset+8.
- LowLimit, Discrete, Offset+9.

To configure a AO connection:

1. Enter the **AO Module Offset**.
2. The **'Input' Offset**, **'HighLimit' Offset**, and **'LowLimit' Offset** fields are disabled as the default values are already entered.

Device Group Definitions

Use the Device Groups dialog box, which appears by clicking the **Device Groups** tab in the Device Selector configuration editor to create, add, delete, and define device groups. You can also configure default update intervals for the objects and edit update intervals in this dialog box.

Note: When you select another part of the Communication Driver tree hierarchy, you are prompted to save the modifications to the configuration set.

To create or add device groups

1. Right-click in the **Device Groups** box and click **Add**.
2. Enter a unique name up to 32 characters long for the device group.

To delete device groups

- Right-click on the device group to be deleted from the list and select **Delete**.

To configure default update intervals

- To configure a default update interval for the object, right-click in the **Device Groups** box and then click **Config Default Update Interval**.

To edit update intervals

- To edit the update interval for an object, double-click its value in the **Update Interval** column and make the edits.

  or

- Right-click its value in the **Update Interval** column and then click **Modify Update Interval**.

  The update interval is the frequency, in milliseconds, that the TCP Communication Driver acquires data from the topics associated with that device group.

  Different topics can be polled at different rates from a PLC by defining multiple device group names for the same PLC and setting a different update interval for each device group.

Device Item Definitions

The device item name is an “alias” or a label for the data in the device. It is an alternative name for the item reference, and can be used instead of the item reference when you create the client application. Device item configuration is optional, but is strongly recommended.

To create or add device items

1. Right-click anywhere in the table, and then click **Add**.
2. In the **Name** column, type a unique item name. The maximum is 32 characters.
3. In the corresponding line, double-click the **Item Reference** column and enter the correlated item reference for the name you created.

To rename device items

Right-click the device item to be renamed and click **Rename**. Make the changes.

To delete device items

Right-click the device item to be deleted from the list and click **Delete**.

To clear all device items

Right-click in the **Device Items** box and click **Clear All**. All the device items listed are cleared after you confirm their deletion.

NOTE: You can import a .csv file containing your item definitions to help streamline configuration. See "Exporting and Importing CSV Files" in the Communication Drivers Pack Help.
