CONTINUAL PROGRESSION

AVEVA (12.1)

.NET Customisation

TM-1406

TRAINING GUIDE

www.aveva.com

Revision Log

Date	Revision	Description of Revision	Author	Reviewed	Approved
08/10/2013	0.1	Approved for Training 12.1.SP4	EJW		
31/10/2013	0.2	Reviewed	EJW	SP	
04/11/2013	1.0	Issued for Training 12.1.SP4	EJW	SP	SK

Updates

All headings containing updated or new material will be highlighted.

Suggestion / Problems

If you have a suggestion about this manual or the system to which it refers please report it to the AVEVA Training & Product Support at tps@aveva.com

This manual provides documentation relating to products to which you may not have access or which may not be licensed to you. For further information on which products are licensed to you please refer to your licence conditions.

Visit our website at **http://www.aveva.com**

Contents

1 Introduction 9
1.1 Aim 9
1.2 Objectives 9
1.3 Prerequisites 9
1.4 Course Structure 9
1.5 Using this Guide 9

Exercise 1 – Training setup 10

2 .NET Customisation Overview 11
2.1 PML .NET Classes 12
2.2 PML .NET User Controls 12
2.3 .NET Addins 13
2.4 Standalone Applications 13
2.5 Customising Application UI 14
2.6 Supporting documentation and examples 15

Exercise 2 – Investigating the supplied examples 17

3 Creating a PMLNetCallable Class 19
3.1 What is a PMLNetCallable C# Class? 19
3.2 An example PMLNetCallable C# Class 20
3.3 Using a PML .NET object 21
3.4 What information can be exchanged? 21
3.5 Loading the PMLNetCallable() 22
3.6 Exception handling within the Class 22

Worked Example - NetTimeZone PMLNetCallable Class 23
Exercise 3 – Providing PML with random numbers 26

4 Creating a PMLNetCallable User Control 29
4.1 Applying a user control to a PML form 29
4.2 Raising user control events 29

Worked Example – DatePickerControl user control 30
Exercise 4 – Providing PML with a check list box 34

5 Creating a .NET Addin for PDMS 35
5.1 Required C# code 35
5.1.1 The IAddin Interface 35
5.1.2 The Command Class 36
5.2 Update the Addin .XML 36
5.3 Add to a PDMS Toolbar 37
5.4 Using PMLNetCallable Controls 37

Worked Example – PDF Viewer .NET Addin 38

5.5 Accessing supporting files 44

Worked Example – PDF Viewer .NET Addin (continued) 45
Exercise 5 – .NET Addin to create circular panels 49

6 Database Interface ... 51
6.1 DbElement ... 51
6.1.1 Getting instance of DbElement ... 51
6.1.2 Current Element ... 51
6.1.3 Getting Element Type ... 52
6.1.4 Navigating ... 52
6.2 Element Attributes ... 53
6.2.1 Getting an Attribute ... 53
6.2.2 Attribute Qualifier ... 53
6.2.3 Setting an Attribute ... 54
6.3 Creating Elements ... 54
6.4 Deleting Elements ... 54
6.5 Moving Elements ... 54

Worked Example – Creating circular panel elements ... 55
Exercise 6 – Adding functionality to the .NET Addin ... 58

7 Collections and Filters ... 59
7.1 Collection ... 59
7.2 Filters ... 59
7.2.1 TypeFilter ... 59
7.2.2 AttributeFalse/TrueFilter ... 60
7.2.3 AttributeRefFilter ... 60
7.2.4 And/OrFilter ... 60
7.2.5 BelowFilter ... 60
7.3 Custom filters ... 61
Worked Example – Collecting circular panel elements ... 62
Exercise 7 – Adding collection functionality to the .NET Addin ... 64

8 PdmsStandalone Interface ... 65
8.1 Making use of PDMSStandalone ... 65
8.2 Setting the environment ... 65
8.3 Accessing specific modules ... 65
Worked Example – MDB Overview ... 66
Exercise 8 – Extending the standalone example ... 69

Appendix A – Source code 71
Appendix A.1 – Source code for Chapter 3 ..... **71**
NetTimeZone.cs 71
RandomNumber.cs 72
Appendix A.2 – Source code for Chapter 4 ..... **74**
NetCalendarControl.cs 74
DatePickerControl.cs 75
CheckListBoxControl.cs 76
cSharpCheckListBoxControl.pmlfrm 78
Appendix A.3 – Source code for Chapter 5 ..... **79**
PdfViewerCntrl.cs 79
PdfViewerCmd.cs 79
PdfViewerOpenCmd.cs 81
PdfViewerAddin.cs 82
CircularPanelCmd.cs 83
PdfViewerAddin.cs (Exercise 5 modifications) 85
Appendix A.4 – Source code for Chapter 6 ..... **86**
CreateCircularPanel.cs 86
CircularPanelCntrl.cs (Exercise 6 modifications) 88
Appendix A.5 – Source code for Chapter 7 ..... **89**
CollectCircularPanel.cs 89
CircularPanelFilter.cs 90
CircularPanelCntrl.cs (Exercise 7 modifications) 90
CircularPanelFilter.cs (Exercise 7 modifications) 91
Appendix A.6 – Source code for Chapter 8 ..... **92**
Form1.cs 92
Form1.cs (Exercise 8 modifications) 93

Appendix B – Additional information 95
Appendix B.1 - Database Expressions **95**
Appendix B.2 - MDB/DB Operations **95**
Appendix B.3 - Invoking PDMS Commands **96**
Appendix B.4 - Getting Information about Current Project and Session **97**
Appendix B.5 - Deploying an Addin on Network Location **98**

CHAPTER 1

1 Introduction

This training guide provides an introduction to.NET customisation of the AVEVA product suite. There is no intention to teach software programming within this guide, but it will provide examples detailing how to customise the AVEVA application using the .NET platform and the C# language.

This training guide is supported by the documentation available within the products installation folder. References will be made to these manuals throughout the guide.

1.1 Aim

To provide the participants with enough knowledge to efficiently understand the following:

- How the .NET platform can be used to customise the AVEVA application
- How to create a PML .NET object
- How to create a PML .NET user control
- How to create .NET addins for an AVEVA product
- How to access the database through the supplied API
- How to find and return data from the database
- How to customise the applications UI

1.2 Objectives

At the end of this training the participants will be able to:

- Describe the types of .NET customising available for AVEVA software.
- Demonstrate how to create a PML Callable .NET object and user control.
- Demonstrate how to create a .NET addin and a standalone application.
- Extract data from the database using the supplied API.
- Setup a custom environment with modified Toolbars and Menu bars.

1.3 Prerequisites

The participants must have completed an AVEVA Basic Foundations Course (or similar) and be familiar with the AVEVA applications. AVEVA PDMS and Visual Studio 2012 must be installed. It is expected the attendee will have knowledge of PML and object oriented programming with some experience of working with C# in a Visual Studio environment.

1.4 Course Structure

The training will consist of oral and visual presentations, demonstrations, worked examples and practical exercises. Each trainee will be provided with some example files to support this guide. Each workstation will have a training project, populated with model objects. This will be used by the trainees to practice their methods, and complete the set exercises.

1.5 Using this Guide

Certain text styles are used to indicate special situations throughout this document. Below is a summary;

Menu pull downs and button press actions are indicated by **bold dark turquoise text**.
Information the user has to Key-in **'will be red and BOLD'**

Additional information will be highlighted

Reference to other documentation will be separate

System prompts should be bold and italic in inverted commas i.e. ***'Choose function'***
Example files or inputs will be in the `courier new font` with colours and styles used as before.

Exercise 1 – Training setup

Install the supplied training files into **C:\AVEVA\Training**.

This includes:

1) **AVEVA.C.Sharp.Training** Visual Strudio 2012 Project
2) Example PML & icons
   a. Csharpcalendarcontrol (code shown in Chapter 4)
   b. Fileopen16.png
   c. Pdf.png
   d. Avevalogo.ico
3) DesignAddins.xml
4) DesignCustomization.xml
5) Training.bat
6) Training_vs.bat
7) Training_evars.bat

The supplied .bat files are modified versions taken from a standard installation of PDMS 12.1.SP4.

The following changes have been made:

```
rem ----------------------------------------------------------------
rem Set pdms_installed_dir to the folder this .bat is running in
rem This line is edited by the installer to point to your chosen folder
rem ----------------------------------------------------------------

set pdms_installed_dir=C:\AVEVA\Plant\PDMS12.1.SP4\
set training_folder=C:\AVEVA\Training\NET

rem ---------------------------------------------
rem Set evars for PDMS
rem this sets all the project variables
rem ---------------------------------------------

call "%training_folder%\training_evars" "%pdms_installed_dir%"

set pmllib=%training_folder%\pmllib %pmllib%
set CAF_ADDINS_PATH=%training_folder%
set CAF_UIC_PATH=%training_folder%
```

The file **Training_vs.bat** has been supplied to start Visual Studio so it can debug PDMS. Depending on the version installed, the called executable may need to be updated.

The supplied **DesignAddins.xml** and **DesignCustomization.xml** files are also from a standard installation. These have been provided as they will be used during the course to register the .NET Addins.

Double-click on **training.bat** and start PDMS.

CHAPTER 2

2 .NET Customisation Overview

It is now possible to integrate user-developed C# code within a PDMS environment, providing the benefits of the using .NET code over standard PML (AVEVAs text based customisation language).

AVEVA supplies a full .NET API to provide access to various parts of the product, including the graphical user interface (GUI), Database and Geometry.

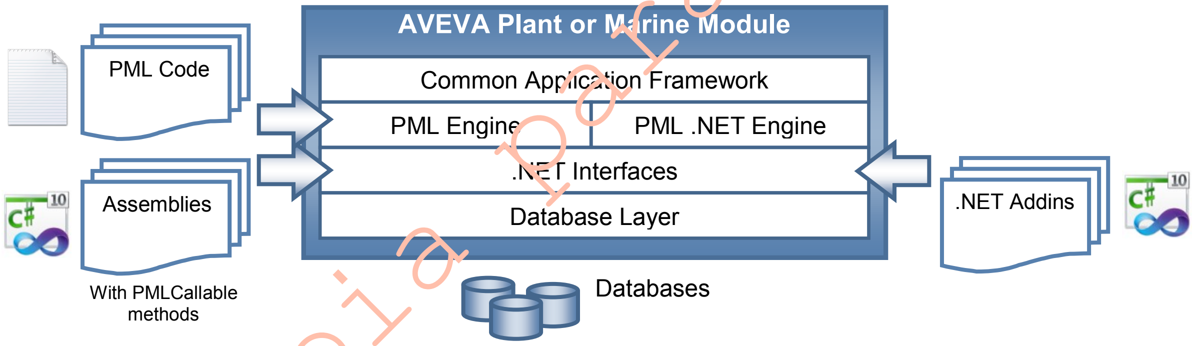
There are four main ways in which C# code can use the supplied API:

1) As a PML .NET object
   A PMLCallable class exposed to PML, providing the strengths of compiles code (PML required)

2) As a PML .NET User Control
   A PMLCallable user control class hosted within a container gadget on a PML form (PML required)

3) As an .NET Addin
   A complete control hosted by PDMS within a form loaded at runtime (No PML required)

4) As a Standalone Application
   Outside PDMS with access to project and databases (No PML required)



The Common Application Framework (CAF) provides developers access to various services within PDMS which support both application development and customisation. The two assemblies which provide access to the CAF are:

⇨ Aveva.ApplicationFramework.dll
⇨ Aveva.ApplicationFramework.Presentation.dll

The AVEVA supplied assemblies are located within the PDMS installation folder. The following assemblies also provide access to PDMS and are available for use:

⇨ Aveva.Pdms.Database.dll	provides access to database elements and attributes
⇨ PDMSFilters.dll	provides collection and filtering functionality
⇨ AVEVA.Pdms.Geometry.dll	provides equivalents to PML geometric objects
⇨ Aveva.Pdms.Shared.dll	provides access to the current element and selection events
⇨ Aveva.Pdms.Utilities.dll	provides messaging, undo, tracing and access to the command line
⇨ Aveva.Pdms.Graphics.dll	provides access to the drawlist and colours
⇨ PMLNet.dll	provides to means to load classes as PML objects

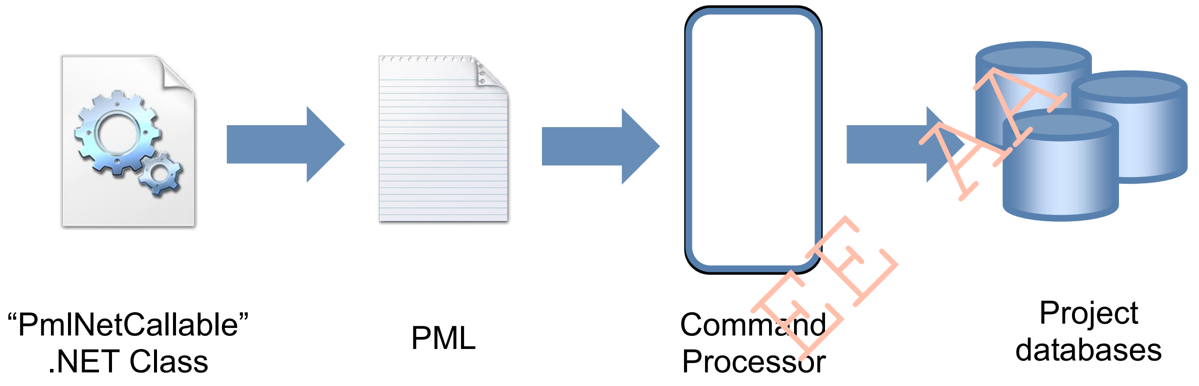
ⓘ *Any other assemblies located within the PDMS installation folder should be considered private and their use is unsupported.*

2.1 PML .NET Classes

The .NET API allows the user to write a class using C#, which can be used directly by PML.

A class becomes available to PML when identified as “PMLNetCallable” and can be used as a normal PML object. This can add extra functionality to PML, including:

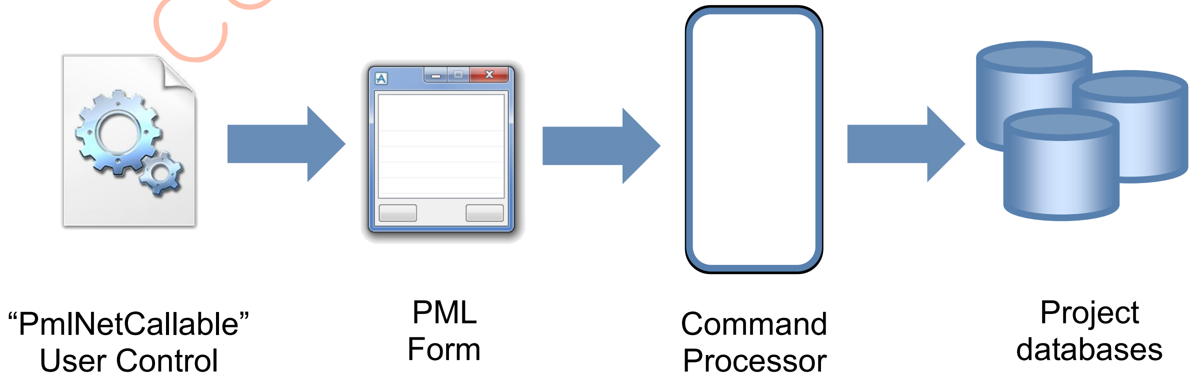
- Communicating with external systems/processes
- Generating random numbers
- Interacting with the operating system
- Reading information from files
- Using a timer to regularly repeat a task (e.g. SAVEWORK)



2.2 PML .NET User Controls

If a PMLNetCallable class contains a user control, it can be loaded then placed onto a PML form. This allows for a greater range of form gadgets to be made available to PML, increasing the user experience. The available possibilities include:

- Deploying the standard windows date picker
- Give a form a web browser, giving specific access to web portal
- Check list boxes, with custom graphics
- Customizable tree views
- Custom control bringing specific functionality to the users



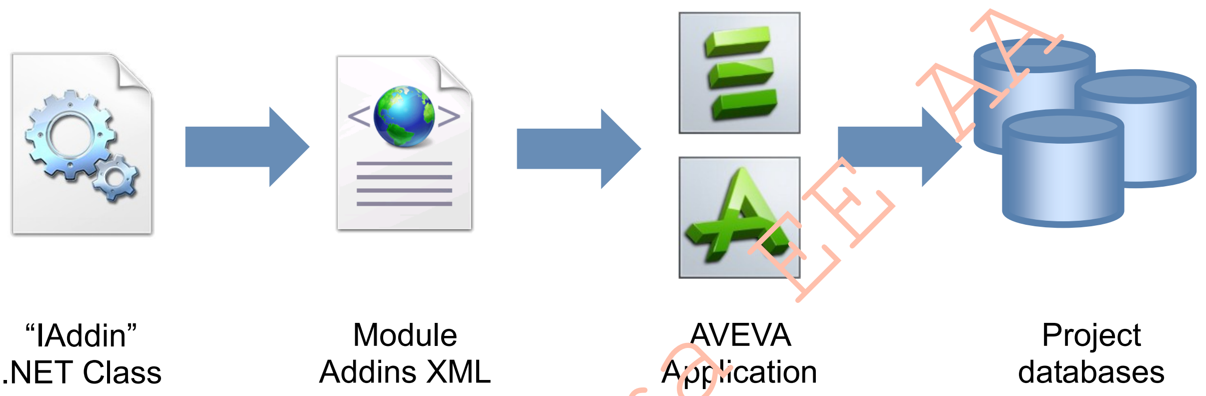
2.3 .NET Addins

A .NET addin is an assembly with a specific implementation added to an AVEVA application. The assembly must contain a single class which implements the IAddin interface. This accesses the AVEVA application to register the addin.

Only .NET addins referenced in the module addin XML file are loaded automatically.

Once loaded, the addin can then use the .NET Interfaces to communicate with the application databases and user interactions. This could include:

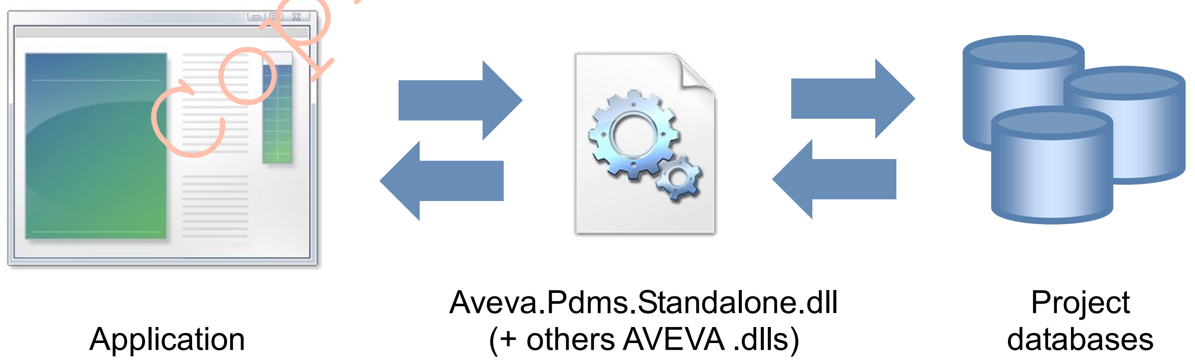
- Element selection
- Changing the current element
- Dealing with geometry
- Accessing database information



2.4 Standalone Applications

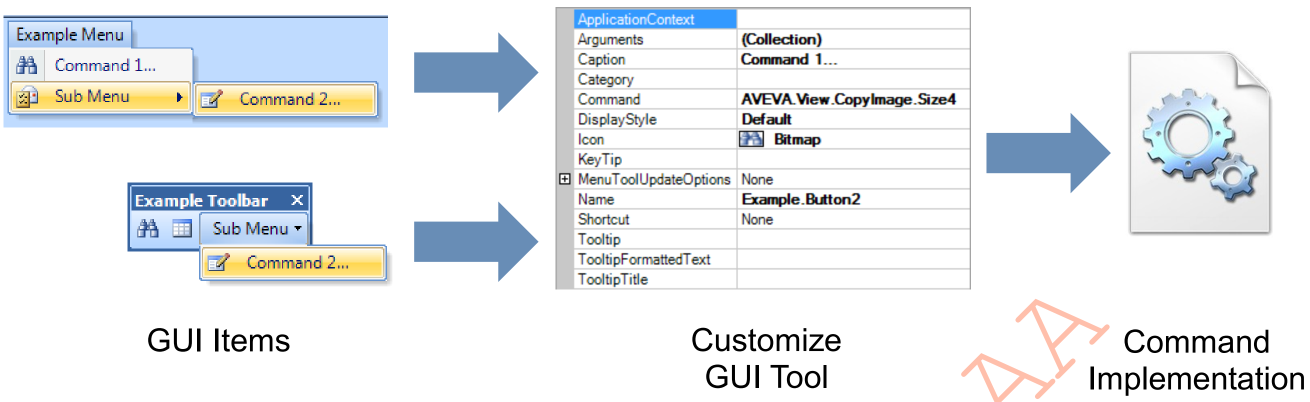
A standalone application is able to access the AVEVA API without being hosted by an AVEVA application.

To do this, it establishes a link by supplying the PdmsStandalone class with the usual PDMS login credentials, which requires the standard PDMS environment variables to locate the project files.



2.5 Customising Application UI

The PDMS UI can be customised using reusable tools like menu items, buttons and separate commands. These tools are associated with .NET commands and the commands are executed when the tool is clicked. Multiple tools can use execute the same command.

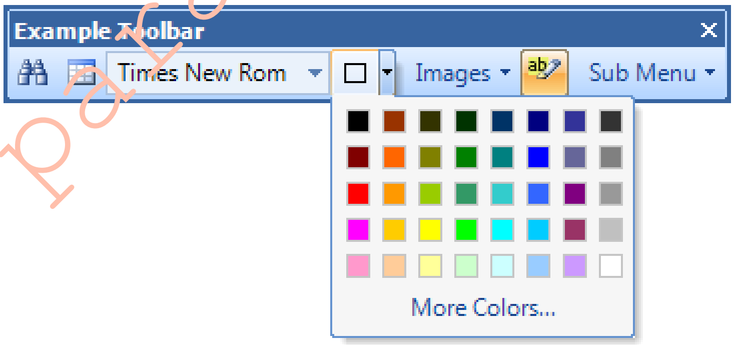


ApplicationContext	
Arguments	(Collection)
Caption	Command 1...
Category	
Command	AVEVA.View.CopyImage.Size4
DisplayStyle	Default
Icon	Bitmap
KeyTip	
MenuToolUpdateOptions	None
Name	Example.Button2
Shortcut	None
Tooltip	
TooltipFormattedText	
TooltipTitle	

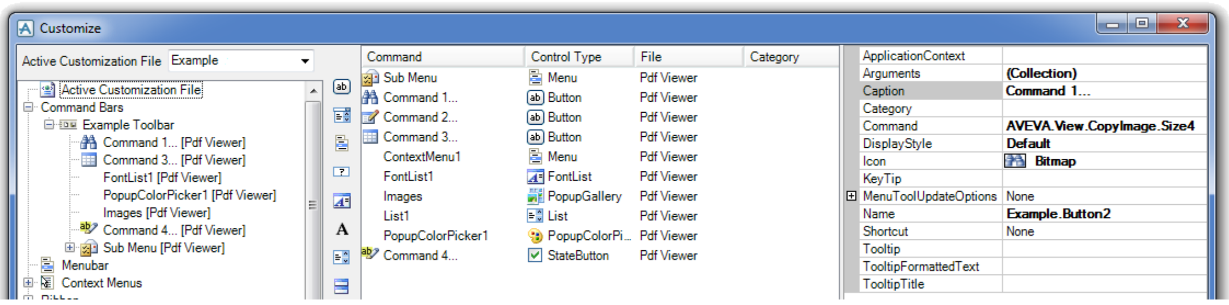
Customised commands are defined and registered by an addin. The command can be associated with any available tool type. They can also control the tools state, visibility or value.

The available tool types are:

- ButtonTool
- ComboBoxTool
- ContainerTool
- FontListTool
- LabelTool
- ListTool
- MdiWindowListTool
- MenuTool
- PopupColorPickerTool
- PopupContainerTool
- StateButtonTool
- TaskPaneTool
- TextBoxTool



Association of a given command with a given tool can be done programmatically via the CAF or by using the below Customisation tool. The information defined using the tool is stored within User Interface files (.uic).


Customize
Active Customization File Example
Active Customization File
Command Bars
Example Toolbar
Command 1... [Pdf Viewer]
Command 3... [Pdf Viewer]
FontList1 [Pdf Viewer]
PopupColorPicker1 [Pdf Viewer]
Images [Pdf Viewer]
Command 4... [Pdf Viewer]
Sub Menu [Pdf Viewer]
Menubar
Context Menus

Command	Control Type	File	Category
Sub Menu	Menu	Pdf Viewer	
Command 1...	Button	Pdf Viewer	
Command 2...	Button	Pdf Viewer	
Command 3...	Button	Pdf Viewer	
ContextMenu1	Menu	Pdf Viewer	
FontList1	FontList	Pdf Viewer	
Images	PopupGallery	Pdf Viewer	
List1	List	Pdf Viewer	
PopupColorPicker1	PopupColorPi...	Pdf Viewer	
Command 4...	StateButton	Pdf Viewer	

ApplicationContext	
Arguments	(Collection)
Caption	Command 1...
Category	
Command	AVEVA.View.CopyImage.Size4
DisplayStyle	Default
Icon	Bitmap
KeyTip	
MenuToolUpdateOptions	None
Name	Example.Button2
Shortcut	None
Tooltip	
TooltipFormattedText	
TooltipTitle	

2.6 Supporting documentation and examples

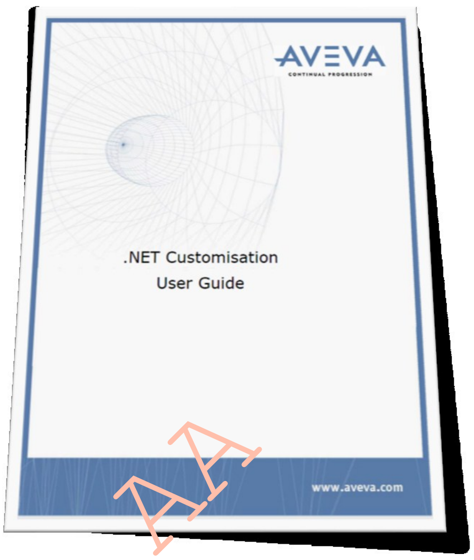
AVEVA PDMS is supplied with supporting documentation and examples to help .NET customisation.

The main document is the **.NET Customisation User Guide**, available through the PDMS help files.

This document describes:

- How to write a CAF Addin
- Customising the menu bar
- Interfacing with the database
- Using PML .NET
- Making use of the supplied Grid Control

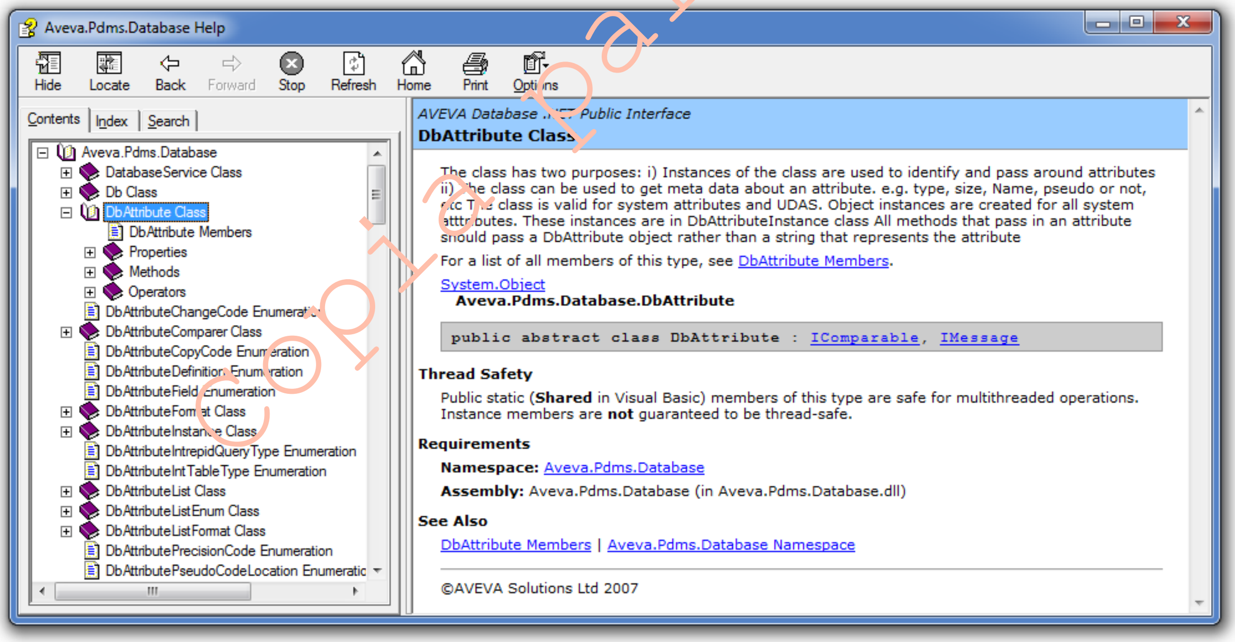
It describes the workflow, but does not expand on the detail of the API. For this, detailed API documentation is available.



API documentation is supplied as a series of windows help files (.chm) available in the following location:

- %PDMSEXE%\Documentation\NetInterfaceReferenceFiles.zip

The same information would also be available through the Object Browser within Visual Studio when working with the classes.


Aveva.Pdms.Database Help
Hide
Locate
Back
Forward
Stop
Refresh
Home
Print
Options
Contents
Index
Search
Aveva.Pdms.Database
DatabaseService Class
Db Class
DbAttribute Class
DbAttribute Members
Properties
Methods
Operators
DbAttributeChangeCode Enumeration
DbAttributeComparer Class
DbAttributeCopyCode Enumeration
DbAttributeDefinition Enumeration
DbAttributeField Enumeration
DbAttributeFormat Class
DbAttributeInstance Class
DbAttributeIntrepidQueryType Enumeration
DbAttributeIntTableType Enumeration
DbAttributeList Class
DbAttributeListEnum Class
DbAttributeListFormat Class
DbAttributePrecisionCode Enumeration
DbAttributePseudoCodeLocation Enumeratic

AVEVA Database .NET Public Interface
DbAttribute Class

The class has two purposes: i) Instances of the class are used to identify and pass around attributes ii) The class can be used to get meta data about an attribute. e.g. type, size, Name, pseudo or not, etc The class is valid for system attributes and UDAS. Object instances are created for all system atttributes. These instances are in DbAttributeInstance class All methods that pass in an attribute should pass a DbAttribute object rather than a string that represents the attribute

For a list of all members of this type, see DbAttribute Members.

System.Object
Aveva.Pdms.Database.DbAttribute

```
public abstract class DbAttribute : IComparable, IMessage
```

Thread Safety

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Requirements

Namespace: Aveva.Pdms.Database

Assembly: Aveva.Pdms.Database (in Aveva.Pdms.Database.dll)

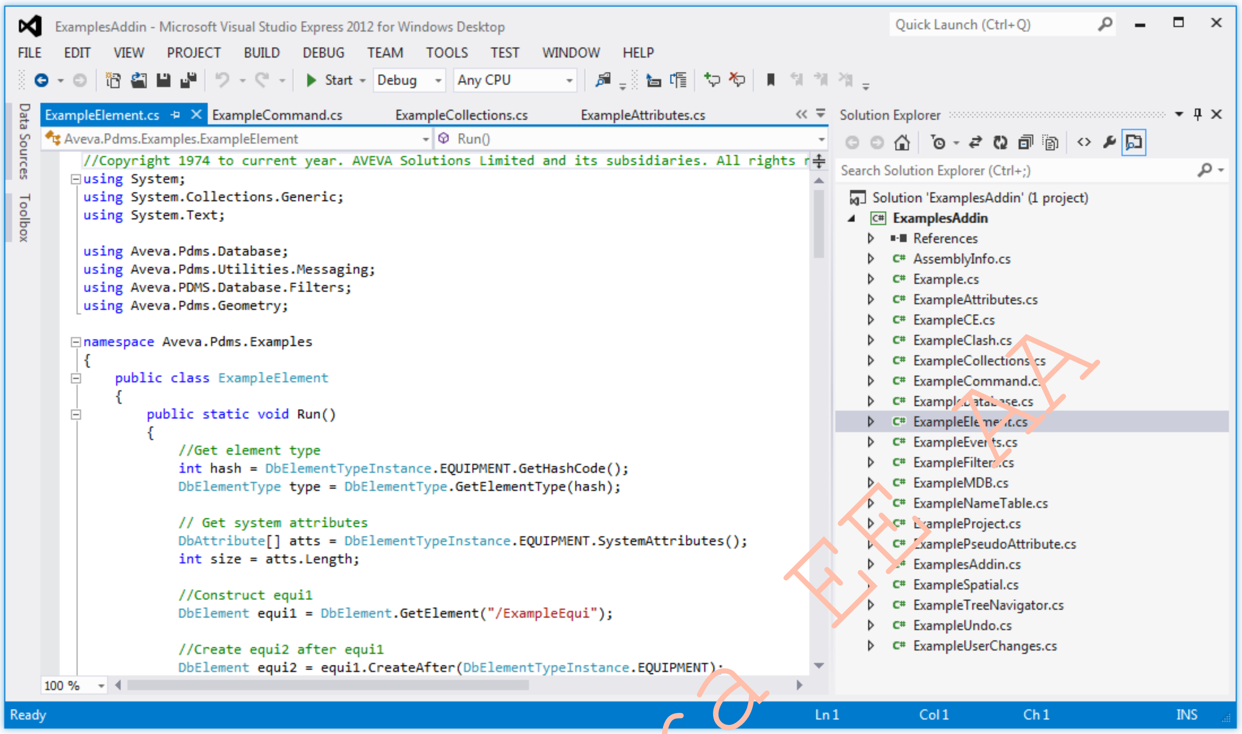
See Also

DbAttribute Members | Aveva.Pdms.Database Namespace

©AVEVA Solutions Ltd 2007

A series of example classes are supplied to demonstrate the use of the API. These can be found within the ExamplesAddin project available in the following location:

- %PDMSEXE%\Samples.zip

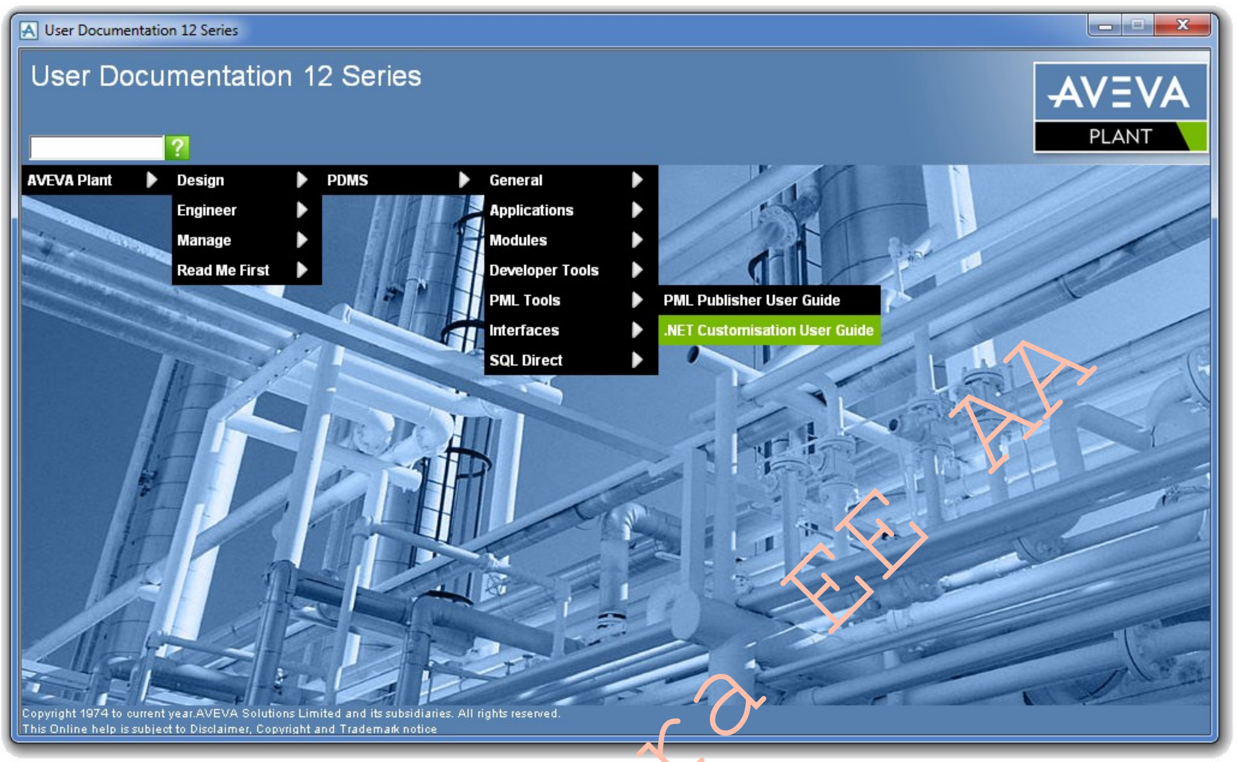

ExamplesAddin - Microsoft Visual Studio Express 2012 for Windows Desktop
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Start
Debug
Any CPU
ExampleElement.cs
ExampleCommand.cs
ExampleCollections.cs
ExampleAttributes.cs
Aveva.Pdms.Examples.ExampleElement
Run()
Data Sources
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
100 %
Ready
Ln 1
Col 1
Ch 1
INS

```
//Copyright 1974 to current year. AVEVA Solutions Limited and its subsidiaries. All rights r
using System;
using System.Collections.Generic;
using System.Text;

using Aveva.Pdms.Database;
using Aveva.Pdms.Utilities.Messaging;
using Aveva.PDMS.Database.Filters;
using Aveva.Pdms.Geometry;

namespace Aveva.Pdms.Examples
{
    public class ExampleElement
    {
        public static void Run()
        {
            //Get element type
            int hash = DbElementTypeInstance.EQUIPMENT.GetHashCode();
            DbElementType type = DbElementType.GetElementType(hash);

            // Get system attributes
            DbAttribute[] atts = DbElementTypeInstance.EQUIPMENT.SystemAttributes();
            int size = atts.Length;

            //Construct equi1
            DbElement equi1 = DbElement.GetElement("/ExampleEqui");

            //Create equi2 after equi1
            DbElement equi2 = equi1.CreateAfter(DbElementTypeInstance.EQUIPMENT);
```

```
Solution 'ExamplesAddin' (1 project)
  ExamplesAddin
    References
    AssemblyInfo.cs
    Example.cs
    ExampleAttributes.cs
    ExampleCE.cs
    ExampleClash.cs
    ExampleCollections.cs
    ExampleCommand.cs
    ExampleDatabase.cs
    ExampleElement.cs
    ExampleEvents.cs
    ExampleFilter.cs
    ExampleMDB.cs
    ExampleNameTable.cs
    ExampleProject.cs
    ExamplePseudoAttribute.cs
    ExamplesAddin.cs
    ExampleSpatial.cs
    ExampleTreeNavigator.cs
    ExampleUndo.cs
    ExampleUserChanges.cs
```

Exercise 2 – Investigating the supplied examples

From the start menu, open the **AVEVA Plant Help Suite 12.1.SP4** (located within the AVEVA Plant folder)

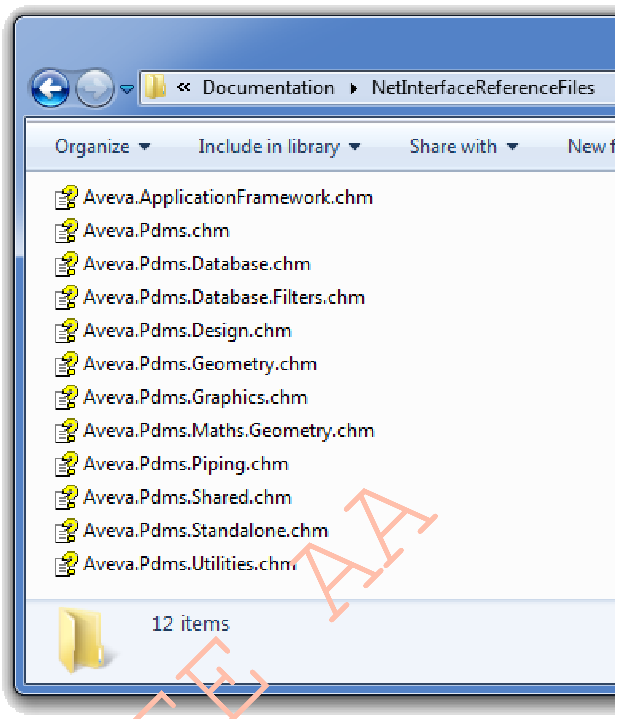

User Documentation 12 Series
User Documentation 12 Series
AVEVA
PLANT
AVEVA Plant
Design
Engineer
Manage
Read Me First
PDMS
General
Applications
Modules
Developer Tools
PML Tools
Interfaces
SQL Direct
PML Publisher User Guide
.NET Customisation User Guide
Copyright 1974 to current year.AVEVA Solutions Limited and its subsidiaries. All rights reserved.
This Online help is subject to Disclaimer, Copyright and Trademark notice

This utility provides access to the available help files. Open the **.NET Customisation User Guide** and review the information to answer the following questions:

Questions:

1. What is the "CAF"?
2. Which assembly provides interfaces related to the database?
3. What is the advice on thread safety?
4. Which database events can be captured by a .NET addin?
5. How should .NET code access the current element?

Browse to find **%PDMSEXE%\Documentation\ NetInterfaceReferenceFiles.zip**. Unzip the file and review the contents. There is a help file provided for each of the public assemblies available for use



Aveva.ApplicationFramework.chm
Aveva.Pdms.chm
Aveva.Pdms.Database.chm
Aveva.Pdms.Database.Filters.chm
Aveva.Pdms.Design.chm
Aveva.Pdms.Geometry.chm
Aveva.Pdms.Graphics.chm
Aveva.Pdms.Maths.Geometry.chm
Aveva.Pdms.Piping.chm
Aveva.Pdms.Shared.chm
Aveva.Pdms.Standalone.chm
Aveva.Pdms.Utilities.chm

12 items

Questions:

1. What is the purpose of the DbAttribute class?
2. Which class and method would give you access to the owner of an element?
3. How many filter classes are available within Aveva.PDMS.Database.Filters?
4. Which filter can be used to ignore a specific element type?
5. How can a .NET addin access which elements are in the drawlist?

Browse to find **%PDMSEXE%\Samples.zip**. Unzip the file and open the project within the **ExamplesAddin** folder. This project contains a series of example classes. Use them to answer the following questions:

Questions:

1. How can a .NET addin access the first available design database world element?
2. What is the process for saving work?
3. How many overloads are available on the **SetAttribute** method of a **DbElement** class?
4. Can an element of a specific name be accessed?
5. Which class can help when collecting elements within a specific volume?

CHAPTER 3

3 Creating a PMLNetCallable Class

The purpose of this chapter is to understand how PML can load and use .NET assemblies to expand the available functionality.

3.1 What is a PMLNetCallable C# Class?

A PMLNetCallable class is a standard C# class that can be used within PDMS by PML. Once loaded, it behaves as a normal PML object.

First, the assembly containing the class must be loaded into PDMS:

```
import |GridControl|
```

This will load **GridControl.dll** from within the **%PDMSEXE%** directory. It is also possible to specify the full file pathname to import assemblies from other directories. For example:

```
import |C:\AVEVA\Plant\PDMS12.1.SP4\GridControl|
```

If the assembly has already been loaded, this command may cause an error. Therefore the command is typically followed by a handle statement. For example:

```
import |GridControl|
handle ANY
endhandle
```

Once imported, the namespace of the class must be specified. This ensures the correct class is located and used. For example:

```
using namespace 'Aveva.Pdms.Presentation'
```

The PML .NET object needs to be assigned to a PML variable before it can be used:

```
!grid = object NETGRIDCONTROL()
```

In this example, **NETGRIDCONTROL** is the name of the class being assigned to the variable **!grid**. If there is an overloaded constructor available, arguments could be supplied to the object within the parenthesis.

Once declared as a variable, it can be queried. The returned information indicates the namespace and name of the class.

```
q var !grid
<AVEVA.PDMS.PRESENTATION.NETGRIDCONTROL> Aveva.Pdms.Presentation.NetGridControl
```

A PMLNetCallable class is identified with the attribute **[PmlNetCallable()]**. It must also have a Constructor and Assign method, both public and identified with the same attribute.

To get the available NetCallable methods on a PML .NET object, type the following:

```
q var !grid.methods()
```

3.2 An example PMLNetCallable C# Class

The following is a C# class (NetString) that can be loaded as a PML .NET object.

```
// NETSTRING Class
// ==========================================================
// Example class, exposed for use with PML
// © AVEVA Solutions Ltd, www.aveva.com

using System;

using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class NetString
    {
        private string mval;

        [PMLNetCallable()]
        public NetString()
        {
            mval = "";
        }

        [PMLNetCallable()]
        public NetString(string val)
        {
            mval = val;
        }

        [PMLNetCallable()]
        public void Assign(NetString that)
        {
            mval = that.mval;
        }

        [PMLNetCallable()]
        public string Val
        {
            get { return mval; }
            set { mval = value; }
        }

        [PMLNetCallable()]
        public string Append(string val)
        {
            mval = mval + val;
          return mval;
        }

        [PMLNetCallable()]
        public double Length()
        {
            return mval.Length;
        }

        public void Reset()
        {
            mval = "";
        }
    }
}
```

PMLNet.dll assembly supplied with PDMS. Allows methods to become "PMLNetCallable"

Field mval
NOT exposed to PML
Can be accessed through the Property Val

Constructor
Exposed to PML (required)
Called when the object is declared

Overloaded Constructor
Exposed to PML
Called when the object is declared

Method Assign
Exposed to PML (required)
One argument: An instance of the object. Used when a PML assignment is made i.e. !a = !b

Property Val
Exposed to PML
Use to get/set the stored string

Method Append
Exposed to PML
A method to append to the stored string and return it

Method Length
Exposed to PML
Used to return the string length

Method Reset
NOT exposed to PML
Used to clear the stored string

3.3 Using a PML .NET object

Once assigned to a variable, all of the exposed class methods are available and can be used. NetCallable properties appear as two methods; a get and set.

The following is an example of using the **NetString** class in the PDMS command window:

```
import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
using namespace |Aveva.C.Sharp.Training|
!string = object NETSTRING(|abcde|)
q var !string
<AVEVA.C.SHARP.EXAMPLES.NETSTRING> Aveva.C.Sharp.Examples.NetString

q var !string.methods()
<ARRAY>
   [1]  <STRING> 'ADDEVENTHANDLER(STRING, ANY, STRING)'
   [2]  <STRING> 'APPEND(STRING)'
   [3]  <STRING> 'ASSIGN(AVEVA.C.SHARP.EXAMPLES.NETSTRING)'
   [4]  <STRING> 'HANDLE()'
   [5]  <STRING> 'LENGTH()'
   [6]  <STRING> 'NETSTRING()'
   [7]  <STRING> 'NETSTRING(STRING)'
   [8]  <STRING> 'REMOVEEVENTHANDLER(STRING, REAL)'
   [9]  <STRING> 'STRING()'
   [10]  <STRING> 'VAL()'
   [11]  <STRING> 'VAL(STRING)'
   [12]  <STRING> 'VALID()'
q var !string.val()
<STRING> 'abcde’
q var !string.length()
<REAL> 5
q var !string.append(|fgh|)
<STRING> 'abcdefgh'
q var !string.length()
<REAL> 8
!string.val(|finished|)
q var !string.val()
<STRING> ‘finished’
```

Available methods
(Notice, no reset method)

Getting the stored string from the property Val

Getting the length of the stored string from the function Length

Adding to the stored string with the sub Append

ⓘ *Once an assembly has been imported, it becomes locked by the application. This means that if it needs to be updated, the application must be restarted. This can be achieved by typing the module name onto the command line.*

3.4 What information can be exchanged?

The following object types can be passed between PML and .NET. All other types must be converted to one of the below before they can be exchanged.

PML Variable Type	.NET Variable Type
REAL	Double
STRING	String
BOOLEAN	Boolean
ARRAY	Hashtable

For Hashtables, the key is of type double and represents the PML Array index.

3.5 Loading the PMLNetCallable()

For the custom attribute **[PMLNetCallable()]** to work within the assembly, it has to be referenced in **AssemblyInfo.cs**. The following is taken from an assembly which works with PML (notice the extra line):

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using Aveva.PDMS.PMLNet;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Aveva.C.Sharp.Training")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("AVEVA Solutions Ltd")]
[assembly: AssemblyProduct("Aveva.C.Sharp.Training")]
[assembly: AssemblyCopyright("Copyright © AVEVA Solutions Ltd 2013")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: PMLNetCallable()]
```

Line added. This assembly is the source of NetCallable()

Extra line added for the Assembly

3.6 Exception handling within the Class

Errors can be returned to PDMS from the .NET code by using a **PMLNetException**. This provides the error handles for PML to intercept and respond to. The following method is an example of a PMLNetExcepton:

```
[PMLNetCallable()]
public double Real()
{
    try
    {
        return Convert.ToDouble(mval);
    }
    catch
    {
        throw new PMLNetException(1000, 1, "String cannot be converted to number");
    }
}
```

If this method was added to the **NetString** class and the object was used within a PML function or macro, the exception can be handled using the standard PML handle syntax:

```
!input = |abc|
!string = object NETSTRING(!input)
!real = !string.real()
handle(1000, 1)
    $p !!<error.text>
endhandle
```

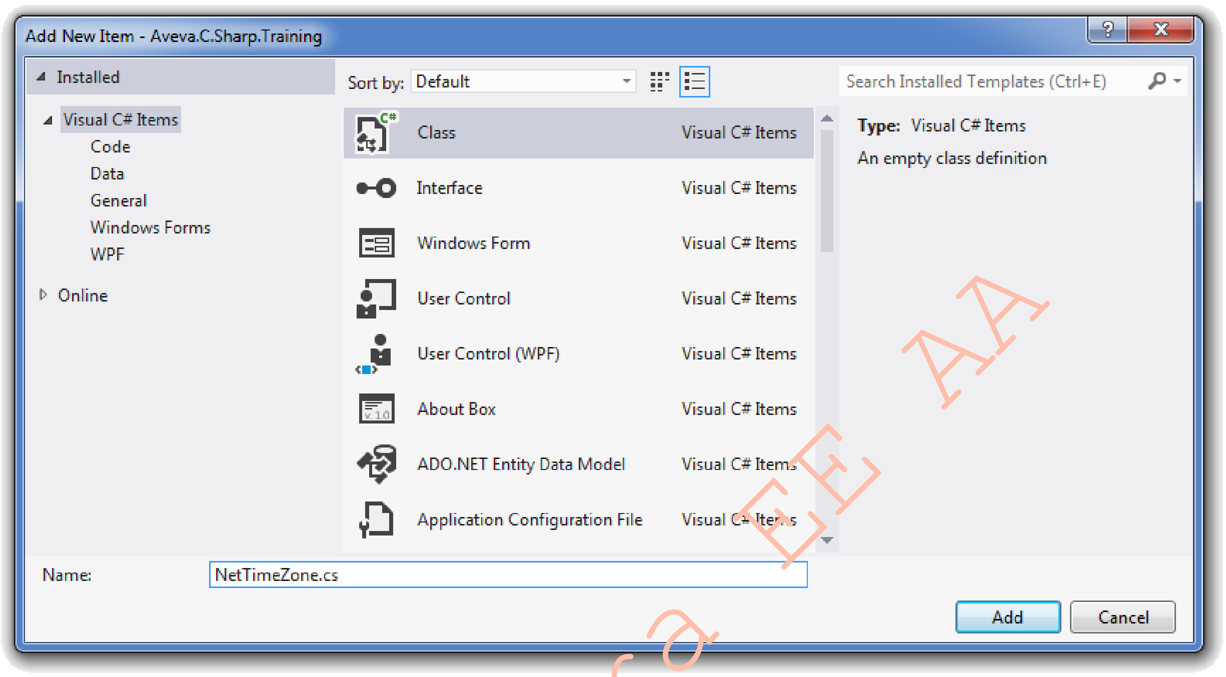
The global error object contains the defined error string.

ⓘ *Notice how the Module number and Message number can be used to handle the specific error (instead of relying on handle ANY)*

Worked Example - NetTimeZone PMLNetCallable Class

1 Startup Visual Studio and open the supplied project.

Add a new class to the project and call it **NetTimeZone.cs**


Add New Item - Aveva.C.Sharp.Training
Installed
Visual C# Items
Code
Data
General
Windows Forms
WPF
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
Application Configuration File Visual C# Items
Add
Cancel

Type: Visual C# Items
An empty class definition

Name: NetTimeZone.cs

2 Update the assemblies used by the class to the following:

```
using System;
using System.Collections;
using System.Collections.ObjectModel;

using Aveva.PDMS.PMLNet;
```

3 Add the **[PMLNetCallable()]** attribute to the class name.

Add the constructor method and the compulsory Assign method (both with the attribute)

```
[PMLNetCallable()]
class NetTimeZone
{

    [PMLNetCallable()]
    public NetTimeZone()
    {
    }

    [PMLNetCallable()]
    public void Assign(NetTimeZone that)
    {
    }
```

4 Add a property to represent the current time zone. Only provide a get method as the value should not be set by the user.

```
[PMLNetCallable()]
public string CurrentTimeZone
{
    get{return TimeZone.CurrentTimeZone.StandardName;}
}
```

5 To test the class, first build the solution and browse to find the .dll. This can be found within the Visual Studio project folders, under:

..\Aveva.C.Sharp.Training\Aveva.C.Sharp.Training\bin\x86\Release

Check it is available and copy it into %PDMSEXE% folder

Alternatively, add a post-build event to the project, e.g.:

```
copy /y "$(TargetPath)" "C:\AVEVA\Training\NET\$(TargetFileName)"
```

ⓘ *This post event will error if the assembly in currently loaded into an open session of PDMS*

6 Within a session of PDMS, type the following into the command window:

```
import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
using namespace |Aveva.C.Sharp.Training|
!timeZone = object NETTIMEZONE()

q var !timeZone
<AVEVA.C.SHARP.TRAINING.NETTIMEZONE> Aveva.C.Sharp.Training.NetTimeZone

q var !timeZone.methods()
<ARRAY>
   [1]  <STRING> 'ADDEVENTHANDLER(STRING, ANY, STRING)'
   [2]  <STRING> 'ASSIGN(AVEVA.C.SHARP.TRAINING.NETTIMEZONEA)'
   [3]  <STRING> 'CURRENTTIMEZONE()'
   [4]  <STRING> 'HANDLE()'
   [5]  <STRING> 'NETTIMEZONEA()'
   [6]  <STRING> 'REMOVEEVENTHANDLER(STRING, REAL)'
   [7]  <STRING> 'STRING()'
   [8]  <STRING> 'VALID()'

q var !timeZone.currentTimeZone()
<STRING> 'GMT Standard Time'
```

If there are any problems with the class, double check the source code and re-compile the .dll.

Using Visual Studios “Attach to process” can help with the debugging.

ⓘ *As the assembly has been imported by the PDMS session, it needs to be restarted before it can be updated. Type the name of the module onto the command line.*

7 Add a new method to get all the available time zone identities.

```
[PMLNetCallable()]
public Hashtable GetSystemTimeZones()
{
    ReadOnlyCollection<TimeZoneInfo> timeZones =
        TimeZoneInfo.GetSystemTimeZones();
    Hashtable TimeZoneArray = new Hashtable();
    int i = 0;
    foreach (TimeZoneInfo timeZoneInfo in timeZones)
    {
        i++;
        TimeZoneArray.Add(i, timeZoneInfo.Id);
    }

    return TimeZoneArray;
}
```

8 Add a new method to return the time in another time zone

```
[PMLNetCallable()]
public string GetTimeZoneInfo(string ZoneName)
{
    try
    {
        DateTimeOffset nowTime = DateTimeOffset.Now;
        TimeZoneInfo UserTimeZone =
            TimeZoneInfo.FindSystemTimeZoneById(ZoneName);
        DateTimeOffset newTime = TimeZoneInfo.ConvertTime(nowTime, UserTimeZone);

        return newTime.ToString();
    }
    catch (Exception ex)
    {
        throw new PMLNetException(1000, 1, ex.Message);
    }

}
```

9 Test the object within PDMS to confirm it is working.

```
import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
using namespace |Aveva.C.Sharp.Training|
!timeZone = object NETTIMEZONE()

q var !timeZone.currentTimeZone()
<STRING> 'GMT Standard Time'

!timeZones = !timezone.getSystemTimeZones()
q var !timeZones[86]
<STRING> 'US Eastern Standard Time'

q var !timeZone.getTimeZoneInfo(!timeZones[86])
<STRING> '01/01/2012 09:00:00 -04:00'

q var !timeZone.getTimeZoneInfo(|United Kingdom|)
(1000,1)The time zone ID 'United Kingdom' was not found on the local computer.
```

Exercise 3 – Providing PML with random numbers

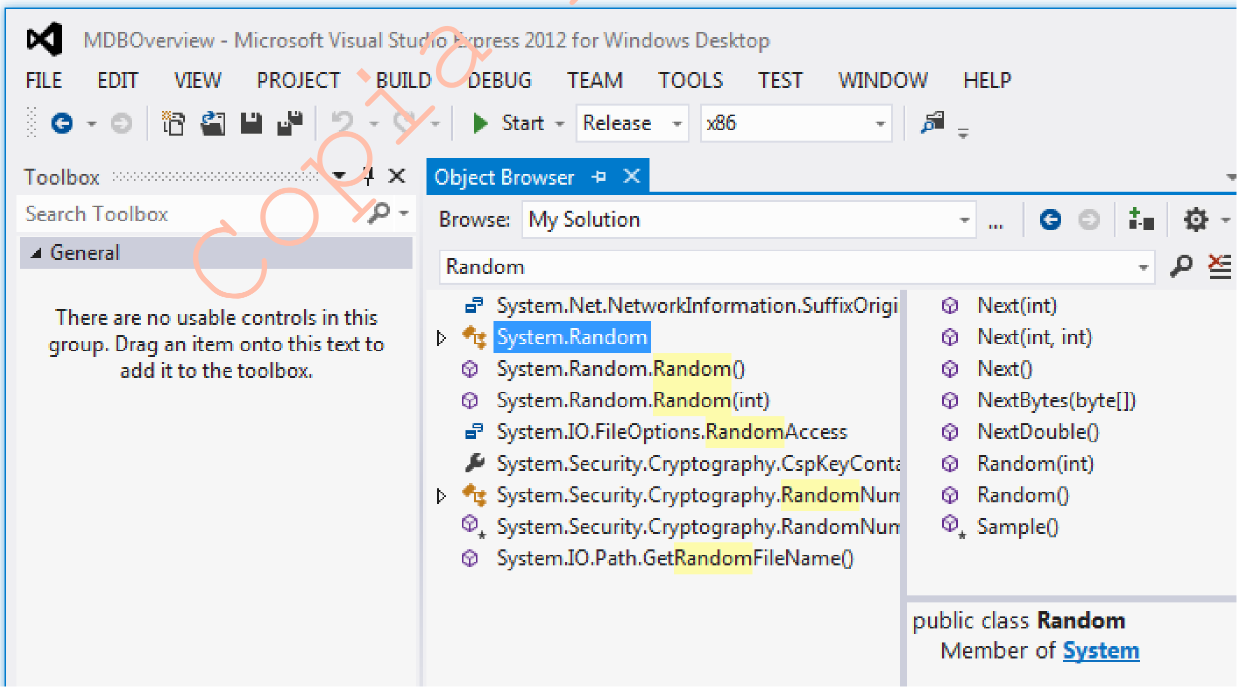
Using the **System.Random** class, create a PMLNetCallable() class that can generate random numbers.

Once created, a user should be able to type the following to generate random numbers:

```
import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
using namespace |Aveva.C.Sharp.Training|
!random = object RANDOMNUMBER()
q var !random
<AVEVA.C.SHARP.Training.RANDOMNUMBER> Aveva.C.Sharp.Training.RandomNumber
q var !random.methods()
<ARRAY>
   [1]  <STRING> 'ADDEVENTHANDLER(STRING, ANY, STRING)'
   [2]  <STRING> 'ASSIGN(AVEVA.C.SHARP.EXAMPLES.RANDOMNUMBER)'
   [3]  <STRING> 'GENERATE()'
   [4]  <STRING> 'HANDLE()'
   [5]  <STRING> 'RANDOMNUMBER()'
   [6]  <STRING> 'REMOVEEVENTHANDLER(STRING, REAL)'
   [7]  <STRING> 'STRING()'
   [8]  <STRING> 'VALID()'
q var !random.generate()
<REAL> 0.885699015057506
q var !random.generate()
<REAL> 0.23129615012151
q var !random.generate()
<REAL> 0.246350737868972
q var !random.generate()
<REAL> 0.979015379668686
```

Open the **Object Brower** in Visual Studio (available from the View menu)

Search for “**Random**” in “**My Solution**” and review the available methods.



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

System.Net.NetworkInformation.SuffixOrigi
System.Random
System.Random.Random()
System.Random.Random(int)
System.IO.FileOptions.RandomAccess
System.Security.Cryptography.CspKeyConta
System.Security.Cryptography.RandomNum
System.Security.Cryptography.RandomNum
System.IO.Path.GetRandomFileName()

Next(int)
Next(int, int)
Next()
NextBytes(byte[])
NextDouble()
Random(int)
Random()
Sample()

public class **Random**
Member of System

Consider how best to use the **System.Random** class and which method can return a **double** to PML. When should it be defined within the code?

Create a new **PMLNetCallable** class called **RandomNumber**, making sure it has the required constructor and assign method.
Add a **.generate()** method and using an instance of a **System.Random** class

Build and test the class by rapidly calling the **.generate()** method from the command line.

Are unique random numbers created?

Extra:

Extend the functionality by providing an additional method that will allow a random number to be generated between two values, e.g.

```
[PMLNetCallable()]
public double Generate(double minimum, double maximum, bool rounded)
```

What other methods are available on **System.Random** that can be exposed to PML?

Consider adding a private field to store if a rounded value is required. Add an overloaded constructor to set the value and add another Generate method to use it:

```
[PMLNetCallable()]
public double Generate(double minimum, double maximum)
```

The object should now behave as follows:

```
q var !random.methods()
<ARRAY>
   [1]  <STRING> 'ADDEVENTHANDLER(STRING, ANY, STRING)'
   [2]  <STRING> 'ASSIGN(AVEVA.C.SHARP.TRAINING.RANDOMNUMBER)'
   [3]  <STRING> 'GENERATE()'
   [4]  <STRING> 'GENERATE(REAL, REAL)'
   [5]  <STRING> 'GENERATE(REAL, REAL, BOOLEAN)'
   [6]  <STRING> 'HANDLE()'
   [7]  <STRING> 'RANDOMNUMBER()'
   [8]  <STRING> 'RANDOMNUMBER(BOOLEAN)'
   [9]  <STRING> 'REMOVEEVENTHANDLER(STRING, REAL)'
   [10]  <STRING> 'ROUNDED()'
   [11]  <STRING> 'ROUNDED(BOOLEAN)'
   [12]  <STRING> 'STRING()'
   [13]  <STRING> 'VALID()'

q var !random.generate()
<REAL> 0.876277111878748

q var !random.rounded()
<BOOLEAN> FALSE

q var !random.generate(10,20)
<REAL> 15.2821603954221

!random.rounded(true)

q var !random.generate(10,20)
<REAL> 13
```

An example of the completed RANDOMNUMBER class can be found in Appendix A

CHAPTER 4

4 Creating a PMLNetCallable User Control

It is possible to create customer user controls, defined within a PMLNetCallable assembly. The C# class defining the control must be derived from UserControl and exposed to PML using the [PMLNetCallable()] attribute.

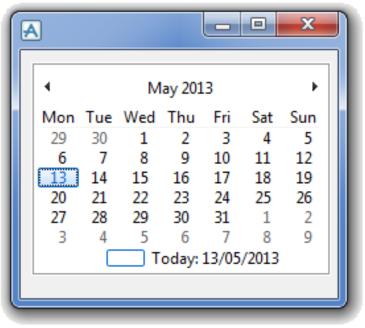
4.1 Applying a user control to a PML form

If an assembly contains a User Control, it can be assigned to a PML **CONTAINER** gadget and hosted onto a PML form. The container gadget acts like a placeholder for the user control. The following is an example form definition that will host a custom user control:

```
setup form !!cSharpCalendarControl

  import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
  handle ANY
  endhandle
  using namespace |Aveva.C.Sharp.Training|

  container .calendarFrame NOBOX PMLNETCONTROL |date| anchor all width 29 height 7.5

  member .calendar is NETCALENDARCONTROL

exit
```

During the constructor method, the two are linked by assigning the return of the **.handle()** method to the **control** member of the container gadget. For example:

```
define method .cSharpCalendarControl()

  using namespace |Aveva.C.Sharp.Training|

  !this.calendar = object NETCALENDARCONTROL()
  !this.calendarFrame.control = !this.calendar.handle()

endmethod
```



The PML form is shown by typing **SHOW !!cSharpCalendarControl** into the command window.

4.2 Raising user control events

PMLNetCallable components can expose user control events to PML. There is one type of event declared in PMLNet assembly under **Aveva.PDMS.PMLNet** namespace.

Event declaration:

```
public event PMLNetDelegate.PMLNetEventHandler OnDateSelected;
```

Where **OnDateSelected** is the name of the event being created.

ⓘ *As PMLNetEventHandler type is defined by the PMLNet component there is no need to expose it to PML with [PMLNetCallable()] attribute.*

To raise an event it is necessary to declare an **ArrayList** (System.Collections namespace) of data that is to be supplied to PDMS.

```
if (OnDateSelected!= null)
{
        ArrayList args = new ArrayList();
        args.Add(this.monthCalendar1.SelectionStart.ToLongDateString());
        args.Add(this.monthCalendar1.SelectionEnd.ToLongDateString());
        EventName(args);
}
```

In this example, the event will return the start and end dates selected by the user. It runs within the **DateChanged** event on the MonthCalendar control within the user control.

If an event is exposed, it must be associated with a PML method for the values to be returned.   The following method links the event with a PML method . This is typically done during the constructor method:

```
!this.calendar.addeventhandler(|OnDateSelected|, !this, |dateChanged|)
```

The first argument is the name of the event, the second is the location of the PML method to be run and the last is the name of the PML method.

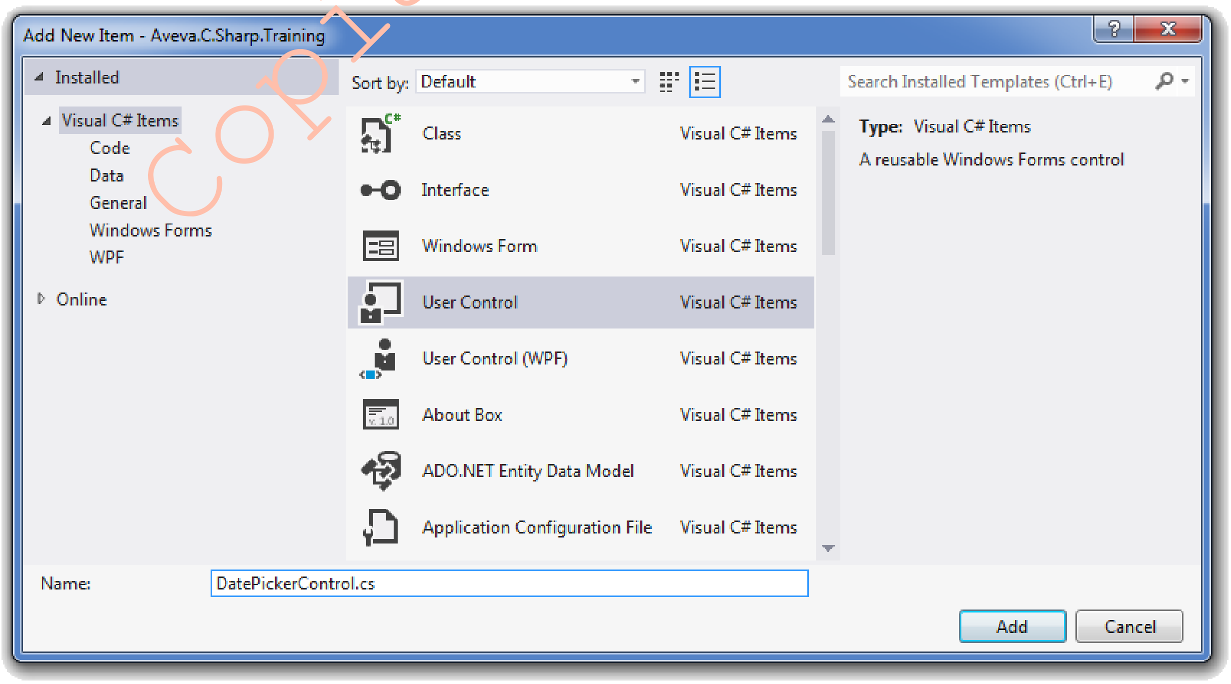
With the above event declared, there should be the following form method available. It must have a single ARRAY argument available.:

```
define method .dateChanged(!data is ARRAY)
  q var !data[0] !data[1]
endmethod
```

As the user interacts with the calendar control, the selected values will be printed to the command window,

Worked Example – DatePickerControl user control

1 Add a new user control to the project and call it DatePickerControl.cs


Add New Item - Aveva.C.Sharp.Training

Installed
- Visual C# Items
  - Code
  - Data
  - General
  - Windows Forms
  - WPF
- Online

Sort by: Default

Search Installed Templates (Ctrl+E)

Class	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items

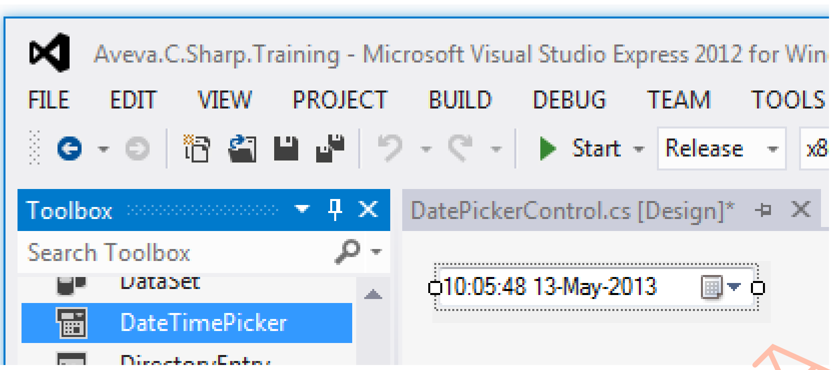
Type: Visual C# Items
A reusable Windows Forms control

Name: DatePickerControl.cs

Add   Cancel

2 Add a **DateTimePicker** control to the user control. Resize the user control to fit the DateTimePicker.

Set the **Format** to **Custom**, **Dock** to **Top** and the **CustomFormat** to **hh:mm:ss dd-MMM-yyyy**


Aveva.C.Sharp.Training - Microsoft Visual Studio Express 2012 for Win
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS
Start
Release
x8
Toolbox
Search Toolbox
DataSet
DateTimePicker
DirectoryEntry
DatePickerControl.cs [Design]*
10:05:48 13-May-2013

3 Switch the code and add:

```
using Aveva.PDMS.PMLNet;
using System.Collections;
```

Add the **[PMLNetCallable()]** attribute to the class name and constructor method. Add the compulsory Assign method (both with the attribute)

```
[PMLNetCallable()]
public partial class DatePickerControl : UserControl
{
        [PMLNetCallable()]
        public DatePickerControl()
        {
                InitializeComponent();
        }

        [PMLNetCallable()]
        public void Assign(DatePickerControl that)
        {
                this.dateTimePicker1.Value = that.dateTimePicker1.Value;
        }
}
```

ⓘ *Notice how the assign method has been used to transfer to the selected date between instances.*

4 Add a method to allow the date to by picked through PML

```
[PMLNetCallable()]
public void SetDate(string DateString)
{
    try
    {
        this.dateTimePicker1.Value = DateTime.Parse(DateString);
    }
    catch
    {
        throw new PMLNetException(1000, 1,
            "String cannot be converted into a date");
    }
}
```

5 Add the date change event:

```
public event PMLNetDelegate.PMLNetEventHandler OnDatePicked;
```

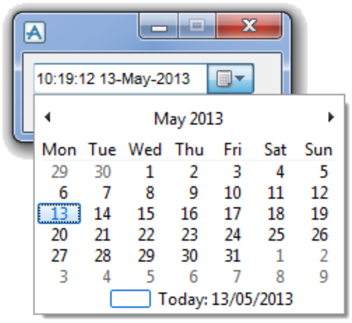
6 On the user control, double click on the DateTimePicker control. This will add a default **ValueChanged** event to the code. Update the provide code to return the selected time and date:

```
private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
{
        if (OnDatePicked != null)
        {
                ArrayList args = new ArrayList();
                args.Add(this.dateTimePicker1.Value.ToLongTimeString());
                args.Add(this.dateTimePicker1.Value.ToLongDateString());
                OnDatePicked(args);
        }
}
```

7 Build the user control and debug any build errors. Confirm the .dll has been updated.

8 Create a new PML form to display the user control. Save the following as **csharpdatepicker.pmlfrm** and save it with the training PMLLIB directory.

```
        setup form !!cSharpDatePicker dialog resiz

          import | C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
          handle ANY
          endhandle
          using namespace |Aveva.C.Sharp.Training|

          container .datePickerFrame NOBOX PMLNETCONTROL |date| width 22 height 2
          member .datePicker is DATEPICKERCONTROL

        exit

        define method .cSharpDatePicker()

          using namespace |Aveva.C.Sharp.Training|

          !this.datePicker = object DATEPICKERCONTROL()
          !this.datePickerFrame.control = !this.datePicker.handle()

          !this.datePicker.addeventhandler(|OnDatePicked|, !this, |datePicked|)

        endmethod

        define method .datePicked(!data is ARRAY)
          q var !data[0] !data[1]
        endmethod
```



Before the form can be shown, type `PML REHASH ALL` into the command window. This will rebuild the PML index form, allowing the form to be located.

Type `SHOW !!cSharpDatePicker` to display the form.

If there is an error in the form definition, it can be reloaded by typing:
`PML RELOAD FORM !!cSharpDatePicker`

9 Add a new text gadget and button to the form. These gadgets will allow the form to test the .setDate() method. The form definition will become:

```
text      .date width 15 is STRING
button    .update |Update| at xmax ymin
container .datePickerFrame NOBOX PMLNETCONTROL |date| width 22 height 2
```

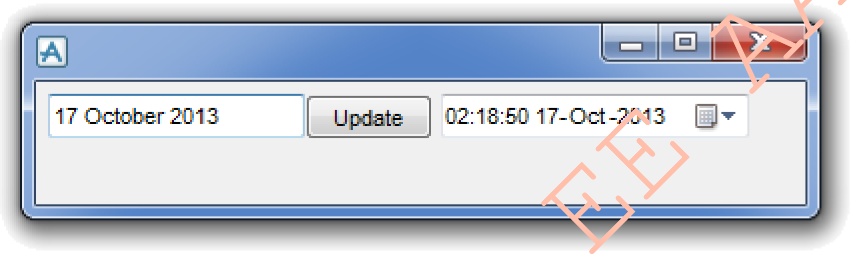
The following line should be added to the constructor:

```
!this.update.callback = |!this.datePicker.setDate(!this.date.val)|
```

The following line should be added to the datePicked method

```
!this.date.val = !data[1]
```

Writing a test form is a great way to test a PML .NET control as it would be used on a form.



Exercise 4 – Providing PML with a check list box

Sometimes on a form there is a need to display a list of checkboxes to capture a series of choices. In PML, this could be constructed as toggle gadgets. However, if the number of check boxes needs to changed, this would require a new form.

To address this problem, using a **ListView**, develop a control that can be hosted onto a PML form and allow a user make selections from a list of choices. On the ListView, set **CheckBoxes** = **True** and **View** = **List**

Ensure the control fills the available space and resizes with the owning PML container gadget.

Add a method that will allow users to add a new item to the list. Overload the method by allowing the check state to be set when the item is created. A ListView owns **ListViewItems** below its **Items** member.

```
[PMLNetCallable()]
public void AddValue(string Value)

[PMLNetCallable()]
public void AddValue(string Value, bool State)
```

Add a method that will get the list items the user has checked. This could return the position of the item in the list, or its value:

```
[PMLNetCallable()]
public Hashtable GetSelectedItems()
```

Add a method that will clear all the items from the list:

```
[PMLNetCallable()]
public void Clear()
```

Add a method that will set the state of all the items in the list:

```
[PMLNetCallable()]
public void SetAllChecked(bool State)
```

Add an event called **OnChecked**. When the user clicks on the list, return to PML information about the item selected:

```
public event PMLNetDelegate.PMLNetEventHandler OnChecked;
```

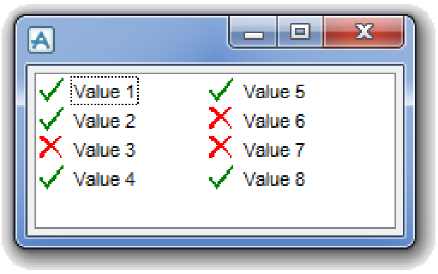
Add two methods to the control:

1) To set checked and unchecked icons (where the arguments are the full paths of the .png files)

```
[PMLNetCallable()]
public void SetCheckImages(string CheckedName, string UnCheckedName)
```

2) Reset back to StandardCheckBoxes

```
[PMLNetCallable()]
public void StandardCheckBoxes()
```



For these methods, research the **StateImageList** property of the ListView in the Object Browser.

Write a PML form to test the control. Add additional gadgets to form to test the functionality (e.g. a toggle to run the **SetAllChecked** method.

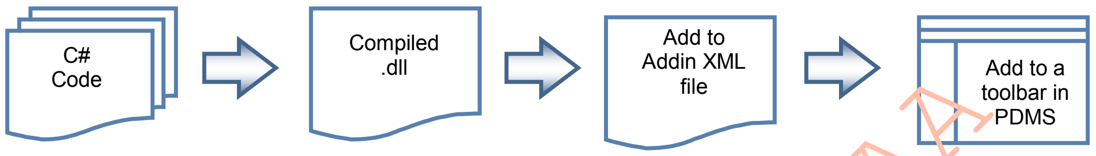
An example of the completed user control can be found in Appendix A

CHAPTER 5

5 Creating a .NET Addin for PDMS

A .NET Addin differs from the previous examples as it does not use any PML and is loaded into PDMS at runtime. When a compiled Addin is loaded, its commands become available to the toolbars and menus.

When writing an Addin, there are a number of stages which need to be followed:



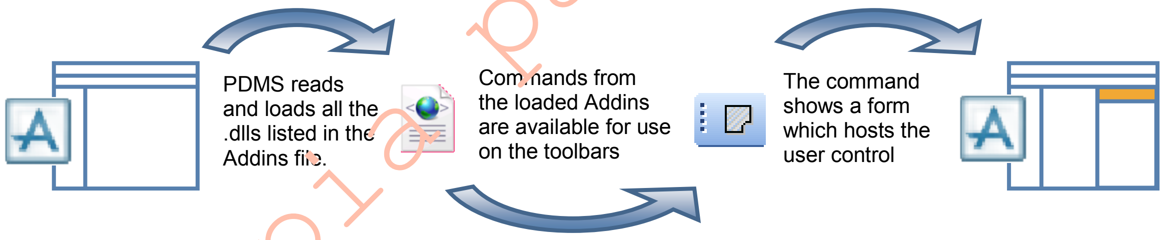
5.1 Required C# code

There are three parts to a .NET Addin:

1) <u>IAddin Interface</u>
Started by PDMS as it loads, it can make the user commands available.

2) <u>Command Class</u>
Available for use on a PDMS toolbar and can build the forms that host custom user controls

3) <u>User Control</u>
The control that will be hosted by the form created by the command


PDMS reads and loads all the .dlls listed in the Addins file.
Commands from the loaded Addins are available for use on the toolbars
The command shows a form which hosts the user control

ⓘ *It is worth noting that the Command Class does not need to generate a form to host a user control. It could just execute code, without any UI. Also an iAddin class does not need to register a command.*

5.1.1 The IAddin Interface

The IAddin interface is loaded when PDMS loads and reads the modules Addin .XML file

⇨ Required assemblies
- Aveva.ApplicationFramework.dll
- Aveva.ApplicationFramework.Presentation.dll

⇨ Required methods:

Start(ServiceManager)	Loads and registers the command
Stop()	No requirement to edit, but must be declared

⇨ Required properties:

String Description	Return the description of the Addin
String Name	Returns the name of the Addin

5.1.2 The Command Class

The command class is can be run from within an AVEVA application and can be used to load and display a custom user control on a form. Once loaded, the command is available for use on a PDMS toolbar and has methods available to show and hide the control.

- Required assemblies:
  - Aveva.ApplicationFramework.Presentation.dll
  - System.Windows.Forms.dll

- Required private members:

DockedWindow MForm	the form created by the class
Boolean BSkip	to stop unnecessary executions

- Required methods:

New(WindowManager)	sets up and registers the form
OnWindowLayoutLoaded()	sets the Checked member based on visibility
OnFormClosed()	sets the Checked member to false
Execute()	run when the command is called

- Required property:

Boolean Checked	stores a Boolean representing form visibility

- Optional property:

Boolean Enabled	can control the state of the calling toolbar gadget
Object Value	can be displayed on an associated toolbar gadget.

5.2 Update the Addin .XML

Each PDMS module has a corresponding .xml file to register the addins. This file takes the naming format of <Module Name>Addins.xml. For example, the design module has a file called **DesignAddins.xml**.

To add the addin to this file, an additional <string> tag can be added:
<string>Aveva.C.Sharp.Training</string>

Referencing the file by only its name assuming the .dll is located within the %CAF_ADDINS_PATH% environment variable. It is possible to refer the full file path of the .dll:
<string>C:\AVEVA\Training\NET\Aveva.C.Sharp.Training</string>

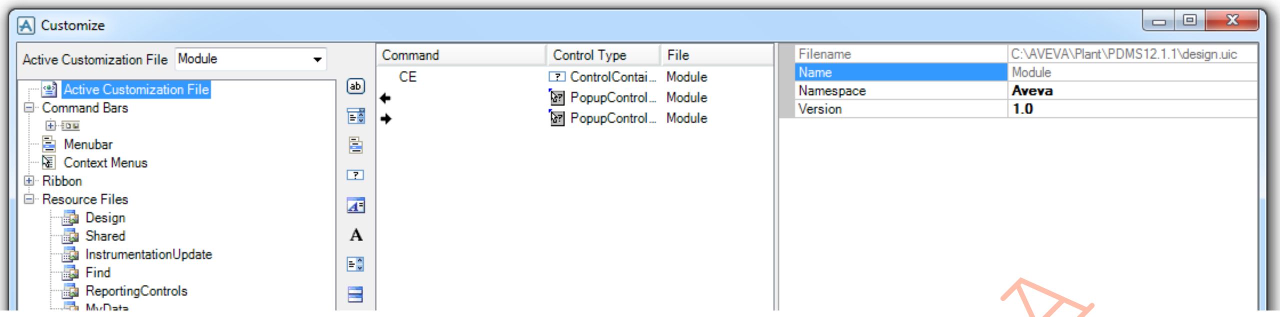
ⓘ *If changes are made to the standard addins .xml file, it should first be copied to a local %CAF_ADDINS_PATH% folder. This will ensure that any changes are kept separate from AVEVA standard product.*

ⓘ *If %CAF_ADDINS_PATH% is not defined, then %PDMSEXE% is used*

5.3 Add to a PDMS Toolbar

A command can be added to a PDMS toolbar from within PDMS. When logged into PDMS Design, right click anyway on a free part of the toolbar and choose **Customize...**

The following form will be displayed:



Customize

Active Customization File: Module

- Active Customization File
- Command Bars
  - Menubar
  - Context Menus
- Ribbon
- Resource Files
  - Design
  - Shared
  - InstrumentationUpdate
  - Find
  - ReportingControls
  - MyData

Command	Control Type	File
CE	ControlContai...	Module
	PopupControl...	Module
	PopupControl...	Module

Filename	C:\AVEVA\Plant\PDMS12.1.1\design.uic
Name	Module
Namespace	Aveva
Version	1.0

The workflow through the form would be as follows:

- Choose the required “Active Customization File”
- Create a new command bar for the new gadgets (right click menu in the LHS pane)
- Update the properties of the command bar (in the RHS pane)
- Create a new button for the command (right click menu in the middle pane)
- Update the properties of the button, including the icon and the command

To register the new .uic file, it needs to be reference in the file <ModuleName>Customization.xml. For example, the default **DesignCustomization.xml** file could be modified (in red) as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<UICustomizationSet xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <UICustomizationFiles>
    <CustomizationFile Name="Module" Path="design.uic" />
    <CustomizationFile Name="SchematicExplorerAddin" Path="CoreSchematicMenu.uic" />
    <CustomizationFile Name="Project" Path="$1.uic" />
    <CustomizationFile Name="SVGCompare" Path="SVGCompare.uic" />
    <CustomizationFile Name=”Training” Path=”C:\AVEVA\Training\NET\training.uic” />
  </UICustomizationFiles>
</UICustomizationSet>
```

This modified file should be saved into the %CAF_UIC_PATH% folder. The .uic file can be referenced with its full file path, or in the above example should be saved local to the DesignCustomization.xml file.

5.4 Using PMLNetCallable Controls

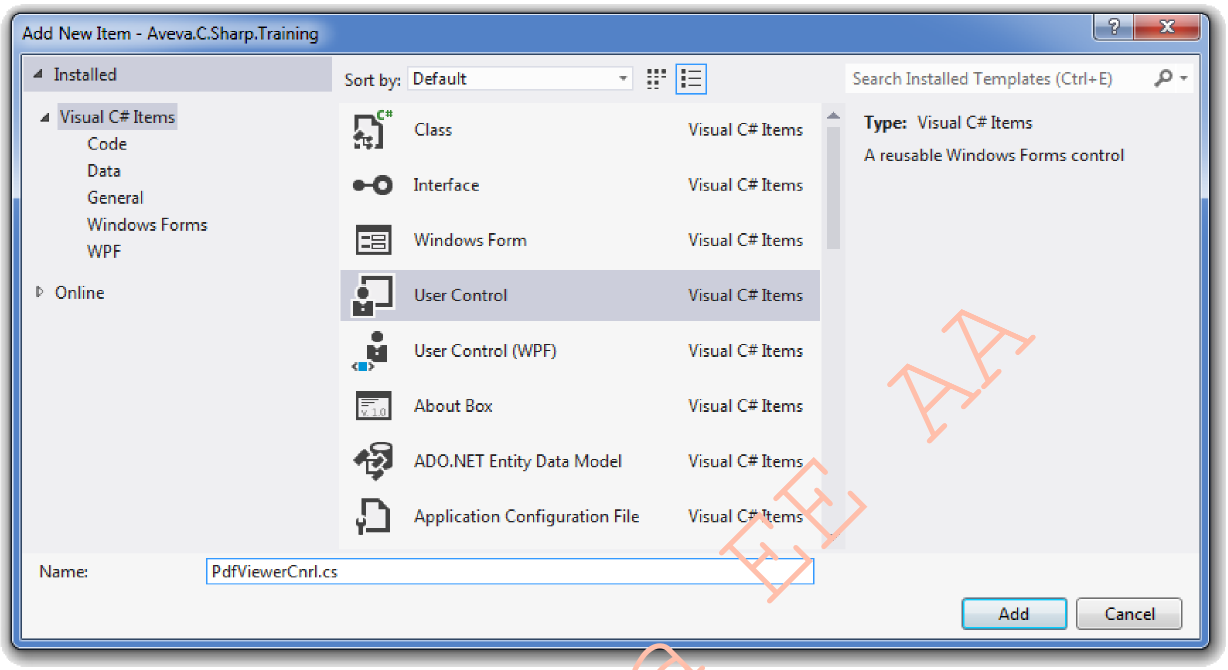
As mentioned before, PMLNetCallable objects can be shared between PML and .NET. It is therefore possible to use controls like the AVEVA NetGridControl or PMLFileBrowser from .NET.

For example, the AVEVA NetGridControl (**GridControl.dll**) user control can be referenced into a custom user control during its constructor method:

```
public partial class PanelsCtrl : UserControl
{
        private NetGridControl mGrid;

        public PanelsCtrl()
        {
                InitializeComponent();

                mGrid = new NetGridControl();
                mResultsPanel.Controls.Add(mGrid);
```

Worked Example – PDF Viewer .NET Addin

1 Create a new user control called **PdfViewerCntrl.cs**. This represents the user control that will be hosted on a form within PDMS once the addin is loaded.

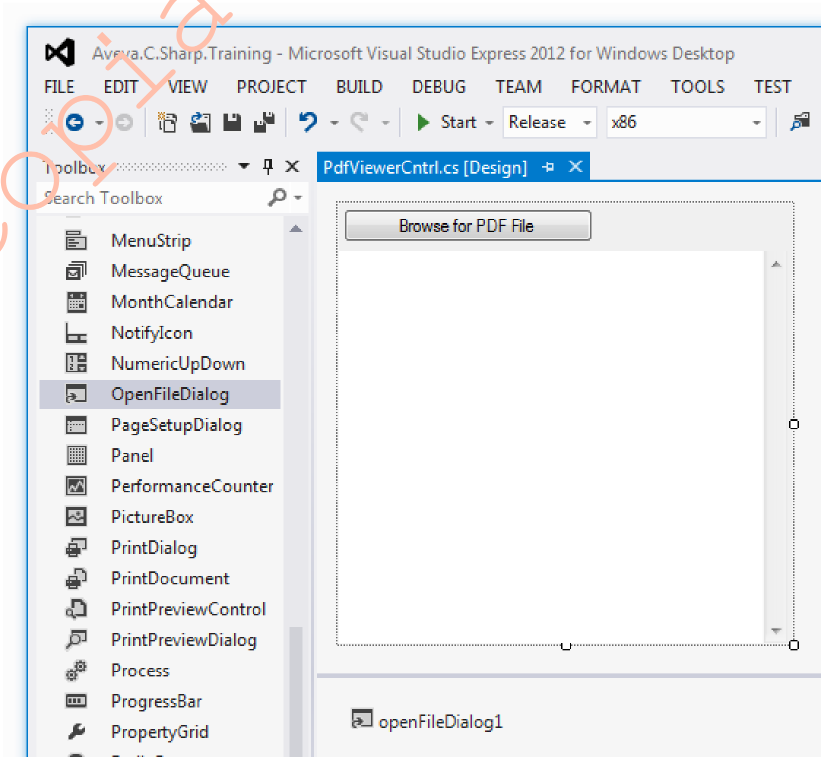


Type: Visual C# Items
A reusable Windows Forms control
Name: PdfViewerCnrl.cs

2 Add a **WebBrowser**, ensuring it is anchored correctly to resize with the user control.

Add a **Button** with the text **Browse for PDF File**.

Add an **OpenFileDialog** control, setting the **Filter** to **PDF Files|*.pdf** and **Title** to **Select PDF File**.



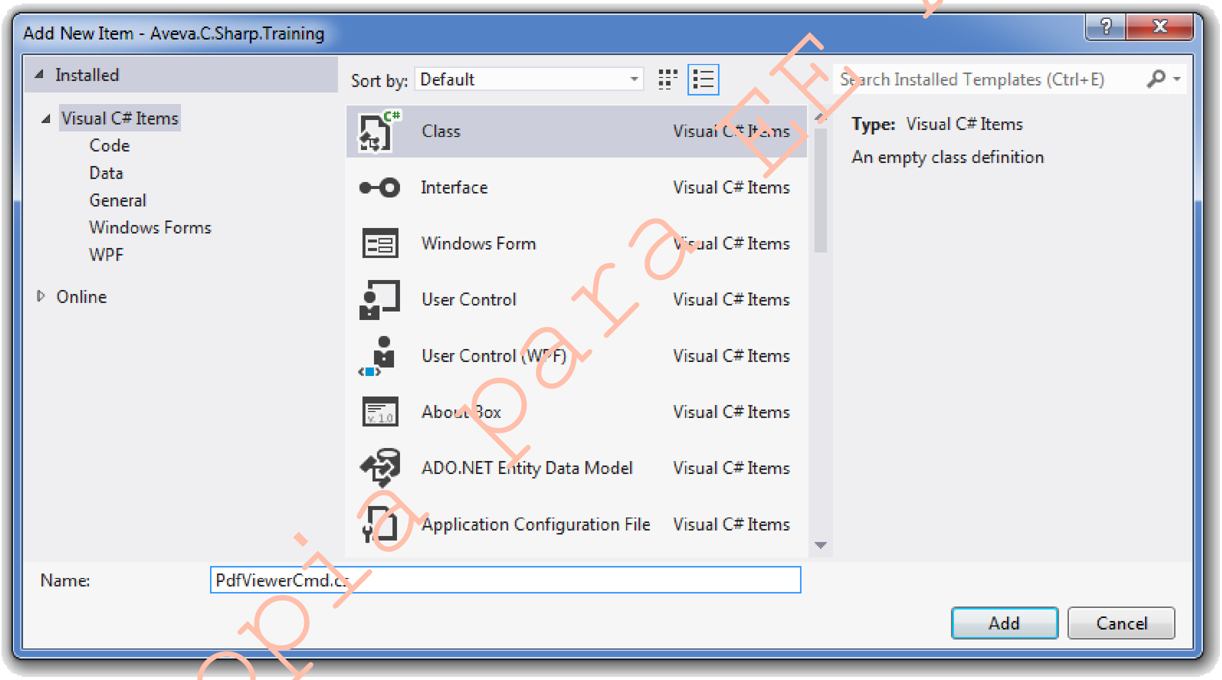
3 Double click on the **Browse for PDF File** button to create a default click event method.

Add the following code to it:

```
private void button1_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;

    if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
        this.webBrowser1.Url = new Uri(this.openFileDialog1.FileName);

    Cursor.Current = Cursors.Default;
}
```

Notice how the file path is passed as the URL, the web browser does the rest of the work!

4 Create a new class called **PdfViewerCmd.cs**. This represents the command that will run the addin. The command will be available to be assigned to a button on the interface.



Type: Visual C# Items
An empty class definition

Name: PdfViewerCmd.cs

5 Update the top of the class to read:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerCmd : Command
    {
        // Store the private members of the command
        private DockedWindow mForm;
        private bool bSkip = false;
```

6 Add a constructor method that receives an argument of a **WindowManager**. This method registers events, but also creates the form that will host the user control.

```
// Constructor method, supplied with an instance of the WINDOWMANAGER
public PdfViewerCmd(WindowManager wndManager)
{
    // Update the KEY of the COMMAND (must be unique for all loaded commands
    base.Key = "Aveva.C.Sharp.Training.PdfViewerCmd";

    // Add an EVENT to look out for when the form is shown
    wndManager.WindowLayoutLoaded += new EventHandler(OnWindowLayoutLoaded);

    // Create a DOCKED window in the WINDOWMANAGER
    mForm = wndManager.CreateDockedWindow("PdfViewerAddin.PdfViewerCmd",
        "PDF Viewer", new PdfViewerCntrl(), DockedPosition.Right);

    mForm.SaveLayout = true;

    // Add an EVENT to look out for when the window is closed
    mForm.Closed += new EventHandler(OnFormClosed);
}
```

7 Add the event methods. These are the two events registered during the constructor method:

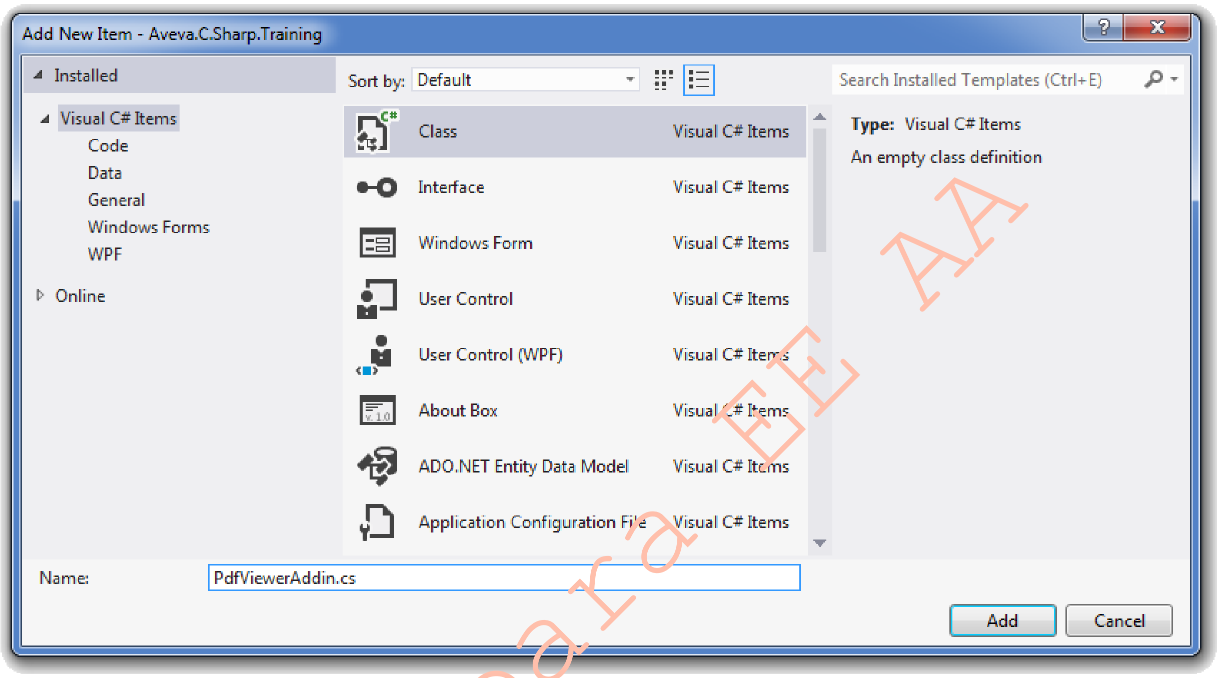
```
// When the control is loaded, ensure the CHECKED property is TRUE
void OnWindowLayoutLoaded(object sender, EventArgs e)
{
    this.Checked = mForm.Visible;
}

// When the CONTROL is closed, ensure the CHECKED property is FALSE
void OnFormClosed(object sender, EventArgs e)
{
    // To override a problem with 12.0.SP5
    this.bSkip = true;
    this.Checked = false;
    this.bSkip = false;
}
```

8 Override the Check property and the Execute method

```
// When the COMMAND is executed, with hide or show the form
public override void Execute()
{
    // If BSKIP, then stop...
    if (bSkip)
        return;

    // If the FORM is visible, hide it and vice versa
    if (mForm.Visible)
        mForm.Hide();
    else
        mForm.Show();

    // Execute the standard method on the base class
    base.Execute();
}
```

```
        // The CHECKED property is used by toggle gadgets on PDMS toolbars
        public override bool Checked
        {
            get { return mForm.Visible;}
            set { base.Checked = value;}
        }
```

9 Create a new class called **PdfViewerAddin.cs**. This represents the addin that PDMS will start.


Add New Item - Aveva.C.Sharp.Training

Installed
- Visual C# Items
  - Code
  - Data
  - General
  - Windows Forms
  - WPF
- Online

Sort by: Default

Search Installed Templates (Ctrl+E)

Class	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items

Type: Visual C# Items
An empty class definition

Name: PdfViewerAddin.cs

10 Update the top of the class to read:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Aveva.ApplicationFramework;
using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerAddin : IAddin
    {
```

11 Add properties to represent the name and description of the addin.

```
        // Get the description of the class
        public string Description
        {
            get { return "PDF Viewer C# Addin"; }
        }
        // Get the name of the class
        public string Name
        {
            get { return "PdfViewerAddin";}
        }
```

12 Add the **Start** and **Stop** methods. The Start method registers the command with the command manager.

```
// When the ADDIN is started, register the command with the commandmanager
public void Start(ServiceManager serviceManager)
{
   // Indicate that the ADDIN has been started in the console window as PDMS loads.
   Console.WriteLine("Example PDF Viewer C# Addin loaded");

   // Get instances of the COMMANDMANAGER and WINDOWMANAGER
   CommandManager sCommandManager =
       (CommandManager)serviceManager.GetService(typeof(CommandManager));
   WindowManager sWindowManager =
       (WindowManager)serviceManager.GetService(typeof(WindowManager));

   // Declare the PDFVIEWERCMD and register it with the COMMANDMANAGER
   sCommandManager.Commands.Add(new PdfViewerCmd(sWindowManager));

}

// For completeness, implement the STOP method, but leave it blank
public void Stop()
{
}
```

13 Add the following line to the **DesignAddins.xml** file located within **C:\AVEVA\Training\NET**

<string>C:\AVEVA\Training\NET\Aveva.C.Sharp.Training</string>

This should be added within the ArrayOfString tags.

Save the file.

14 Add the following line to the **DesignCustomization.xml** file located with **C:\AVEVA\Training\NET**

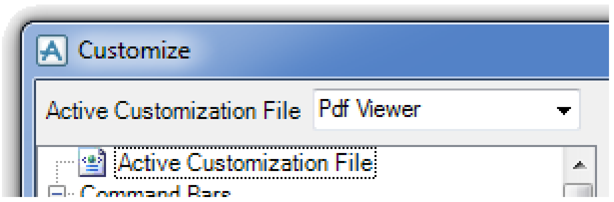
<CustomizationFile Name="PDF Viewer" Path="C:\AVEVA\Training\NET\PdfViewer.uic" />

This should be added within the UICustomizationFiles tags.

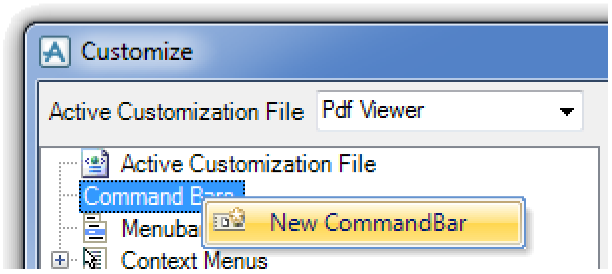
Save the file.

14 Start PDMS. A print to console window will show if the addin has been loaded. Once PDMS has loaded, show the **Customize…** form. This can be found by right clicking on an empty part of the toolbar.

Select **PDF Viewer** as the **Active Customization File**.


Customize
Active Customization File: Pdf Viewer

15 Right click on the **Command Bars** node on the left of the form. Select **New CommandBar**.



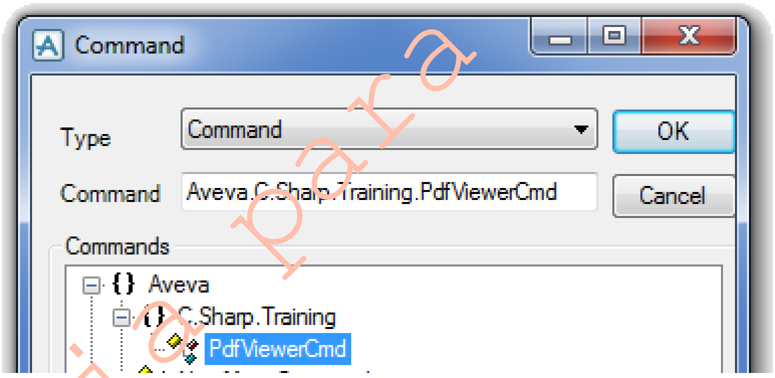
With the new command bar selected, update the Caption of the new command bar to **PDF Viewer**

16 Add a new **StateButton** by clicking the icon. This will add a new control to the end of the list in the middle of the form.

Select the new StateButton and update the **Caption** to **PDF Viewer** and the **DisplayStyle** to **ImageAndText.**

Select the **Command** property and click the button that appears to the right. This will show a form that allows a command to be associated with the button.

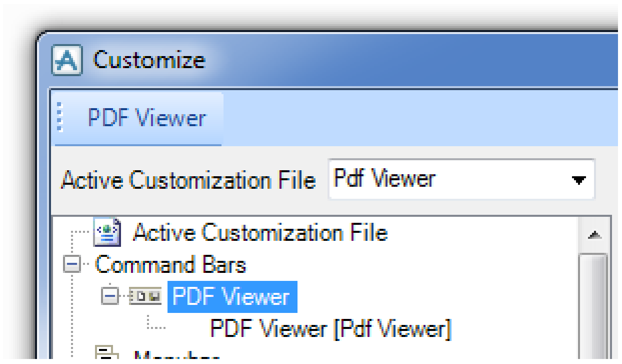
Choose **Aveva.C.Sharp.Training.PdfViewerCmd** and click **OK**.



Type: Command

Command: Aveva.C.Sharp.Training.PdfViewerCmd

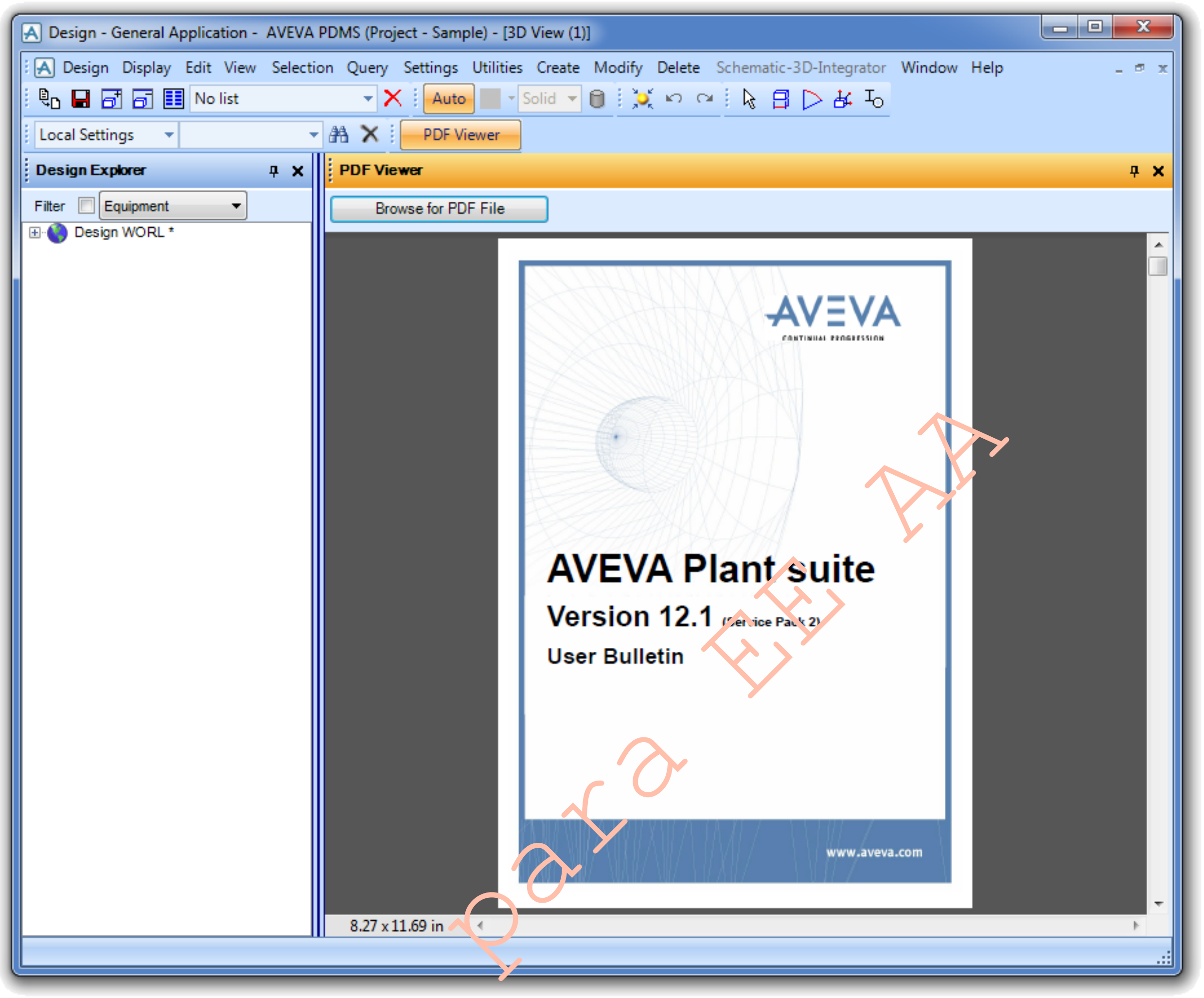
17 Drag the **StateButton** and drop it onto the PDF Viewer command bar. Selecting the command bar will show a preview at the top of the form.



Click **Apply** and then **OK**. This action will close the form and add the new toolbar to the interface.

ⓘ *A new file called PdfViewer.uic has been created in C:\AVEVA\Training.*

18 Click the **PDF Viewer** button and test the addin.


Design - General Application - AVEVA PDMS (Project - Sample) - [3D View (1)]
Design Display Edit View Selection Query Settings Utilities Create Modify Delete Schematic-3D-Integrator Window Help
No list
Auto
Solid
Local Settings
PDF Viewer
Design Explorer
Filter
Equipment
Design WORL *
PDF Viewer
Browse for PDF File
AVEVA
AVEVA Plant suite
Version 12.1 (Service Pack 2)
User Bulletin
www.aveva.com
8.27 x 11.69 in

ⓘ *Notice how the state button indicates when the addin is visible. This is possible because of the way the Checked property of the command has been used.*

5.5 Accessing supporting files

During the Start method of the addin, other files can be registered. This can include resource files and addin specific toolbars. For example, the following code would make available a resource file called ExampleAddin.resources.

```
// Get the ResourceManager service.
ResourceManager resourceManager =
      (ResourceManager)serviceManager.GetService(typeof(ResourceManager));
resourceManager.LoadResourceFile("ExampleAddin");
```

Once loaded, the resource manager can be used to obtain files. For example

```
resourceManager.GetImage("ID_EXAMPLE_ICON");
```

Addin specific toolbars can be registered with the following code:

```
// Load a UIC file for the AttributeBrowser.
CommandBarManager commandBarManager =
    (CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));
commandBarManager.AddUICustomizationFile("ExampleAddin.uic", "ExampleAddin");
```

The window manager also provides access to the Status bar. This can be useful for adding additional prompt information, or to access a progress bar.

Worked Example – PDF Viewer .NET Addin (continued)

1 Update the user control **PdfViewerCntrl.cs** so it only contains a single WebBrowser Control (i.e. remove the button). Update the WebBrowser so it fills the available space.

Modify the code of the user control so it contains these two methods:

```
public void OpenPDfFile()
{
    Cursor.Current = Cursors.WaitCursor;

    if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
        this.webBrowser1.Url = new Uri(this.openFileDialog1.FileName);

    Cursor.Current = Cursors.Default;
}

public string GetPdfName()
{
    return Path.GetFileName(this.openFileDialog1.FileName);
}
```

Using `Path` requires `using System.IO;` to be added to top of the file.

2 Two buttons will be added to the toolbar:

1) Show the addin form
2) Open a PDF file

The **Open** button should only be active when the form is visible. For this to work, both methods need to reference the same control.

Add a new class called **PdfViewerOpenCmd.cs**.

Add the following code at the top of the file:

```
using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerOpenCmd : Command
    {
        private PdfViewerCntrl _pdfViewerCntrl;
```

3 Add a constructor method. This method sets the key and stores the supplied pdfViewerCntrl. It also registers a method against the **CommandExecuted** event of the command **PdfViewerCmd**.

This means whenever the **PdfViewerCmd** command is executed, the **showCommandExecuted** method on the class will execute.

```
public PdfViewerOpenCmd(CommandManager cmdManager, PdfViewerCntrl pdfViewerCntrl)
{
    // Update the KEY of the COMMAND (must be unique for all loaded commands
    base.Key = "Aveva.C.Sharp.Training.PdfViewerOpenCmd";

    Command showCommand =
        cmdManager.Commands["Aveva.C.Sharp.Training.PdfViewerCmd"];
    showCommand.CommandExecuted += new EventHandler(showCommandExecuted);

    _pdfViewerCntrl = pdfViewerCntrl;
}
```

4 Add these methods to the class. They ensure the state of any gadgets associated with the command reflect the visibility state of the PdfViewerCntrl:

```
public void showCommandExecuted(object sender, EventArgs e)
{
    this.Enabled = _pdfViewerCntrl.Visible;
}

public override bool Enabled
{
    get { return _pdfViewerCntrl.Visible; }
    set { base.Enabled = value; }
}
```

5 Add an override for the Execute method. When the PdfViewerCntrl is visible this method will run the **OpenPdfFile** method on the control, set the **Value** as the name of the PDF and refresh any associated gadgets.

```
public override void Execute()
{
    if (_pdfViewerCntrl.Visible)
    {
        _pdfViewerCntrl.OpenPDfFile();
        this.Value = _pdfViewerCntrl.GetPdfName();
        this.Refresh();
    }

    // Execute the standard method on the base class
    base.Execute();
}
```

6 Update the constructor method of class **PdfViewerCmd** to accept a second argument:

```
public PdfViewerCmd(WindowManager wndManager, PdfViewerCntrl pdfViewerCntrl)
```

Add a private member to the class:

```
private PdfViewerCntrl _pdfViewerCntrl;
```

Update the stored value during the constructor method by using `_pdfViewerCntrl = pdfViewerCntrl;` and use `_pdfViewerCntrl` when defining `mForm`.

7 Open the class **PdfViewerAddin.cs** and add the following after the class name:

```
private PdfViewerCntrl _pdfViewerCntrl;
```

Update the code that registers the commands during the constructor method to the following:

```
_pdfViewerCntrl = new PdfViewerCntrl();

// Declare the PDFVIEWERCMD and register it with the COMMANDMANAGER
sCommandManager.Commands.Add(new PdfViewerCmd(sWindowManager, _pdfViewerCntrl));
sCommandManager.Commands.Add(new PdfViewerOpenCmd(sCommandManager,
    _pdfViewerCntrl));
```

At the end of the constructor method, add the following code:

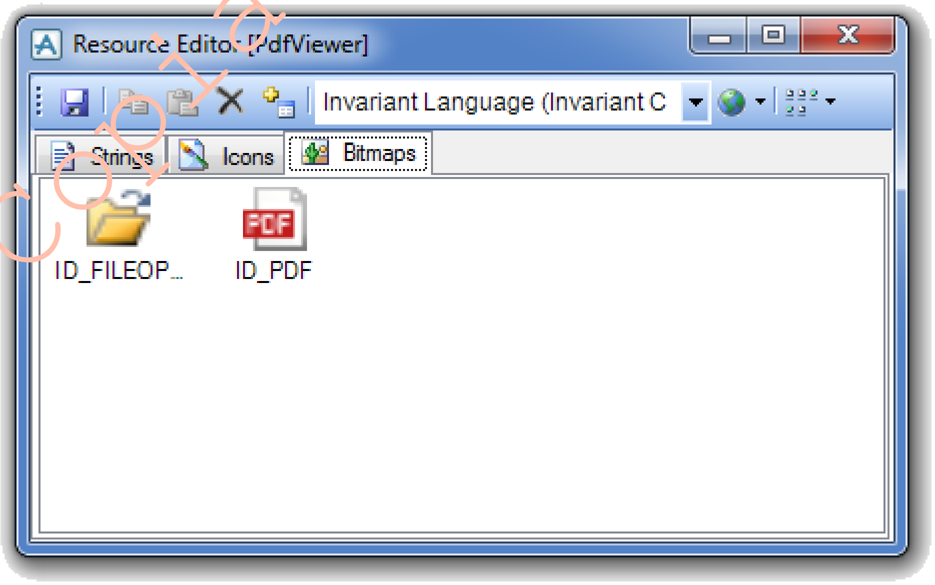
```
ResourceManager rManager =
(ResourceManager)serviceManager.GetService(typeof(ResourceManager));
rManager.LoadResourceFile("PdfViewer");

// Load a UIC file for the Pdf Viewer.
CommandBarManager commandBarManager =
(CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));
commandBarManager.AddUICustomizationFile("PdfViewer.uic", "Pdf Viewer");
```

This code will load into PDMS the UIC file and supporting resource file.

8 To create the resource file for the addin, start **ResourceEditor.exe** (located within the PDMS installation folder).

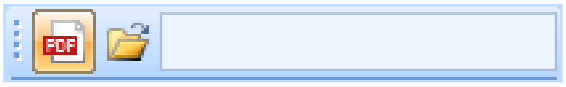
An **Open Resource File** dialog will be displayed first. Browse to **C:\AVEVA\Training\NET**, enter the name **PdfViewer** and click **Open**.

On the form that opens, click the button and add the two icons supplied in the training pmllib.



With the files added, click and close the utility.

9 Within PDMS, modify the customization file **PdfViewer.uic** to provide the following toolbar:



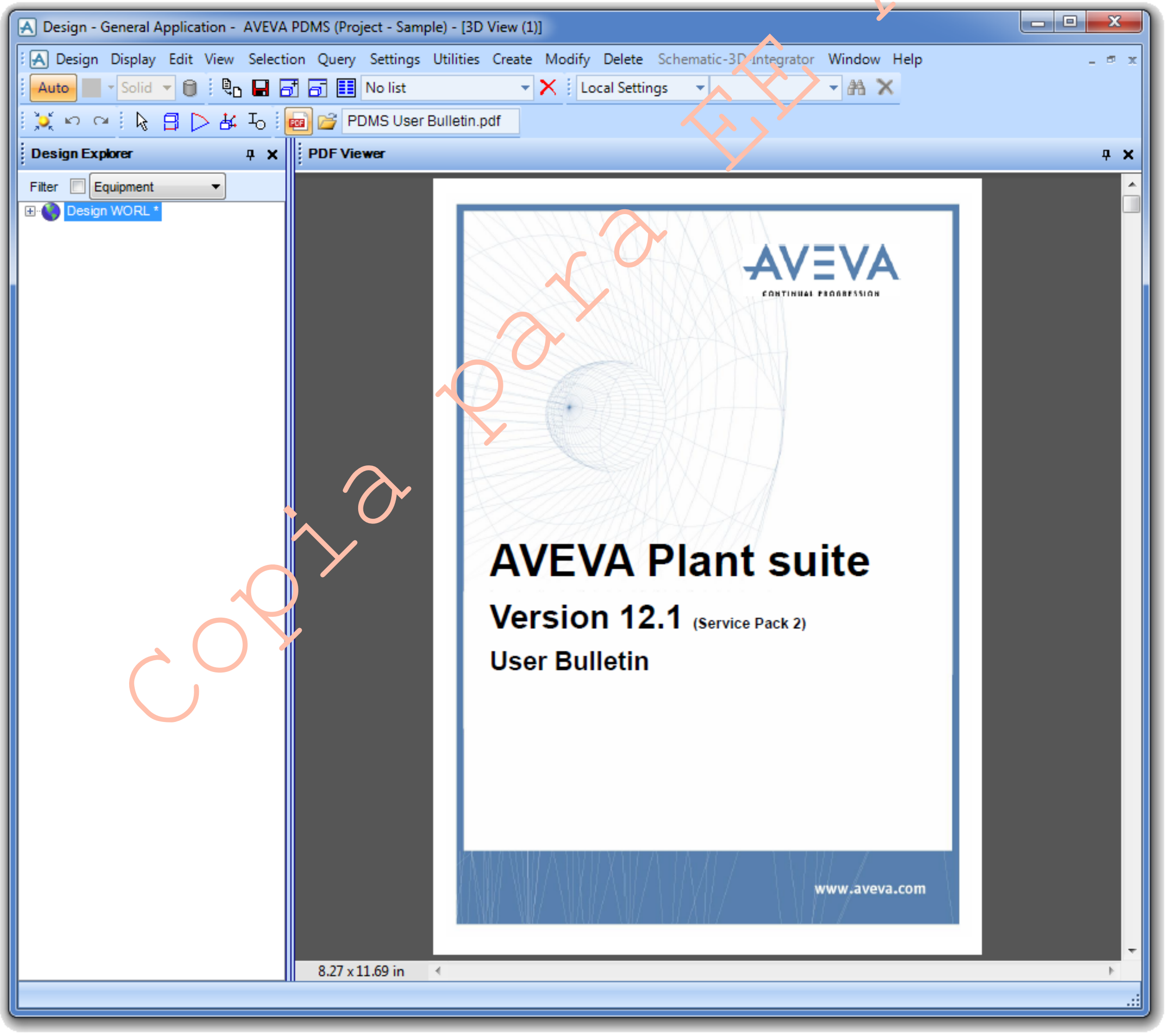
Featuring the following three controls:

1)	StateButton	- command PdfViewerCmd
2)	Button	- command PdfViewerOpenCmd
3)	TextBox	- command PdfViewerOpenCmd

Set suitable captions and tooltips.

Add the icons from the resource file **PdfViewer**.

10 Test the addin within PDMS, ensuring the state of the gadgets is correct.


Design - General Application - AVEVA PDMS (Project - Sample) - [3D View (1)]
Design Display Edit View Selection Query Settings Utilities Create Modify Delete Schematic-3D Integrator Window Help
Auto Solid No list Local Settings
PDMS User Bulletin.pdf
Design Explorer
PDF Viewer
Filter Equipment
Design WORL *
AVEVA
AVEVA Plant suite
Version 12.1 (Service Pack 2)
User Bulletin
www.aveva.com
8.27 x 11.69 in

As PdfViewer.uic is being directly loaded by the addin, it no longer needs to be referenced within DesignCustomization.xml

Exercise 5 – .NET Addin to create circular panels

Create a new user control called **CreateCircularPanelCntrl.cs**

Using a mixture of standard controls, setup the control to look like the image to the right. This will include:

- PictureBox
- Label
- Button
- NumericUpDown
- ListView
- Text

The main picture is available within **C:\AVEVA\Plant\PDMS12.1.SP4\PMLLIB\icons\aslgeneral**

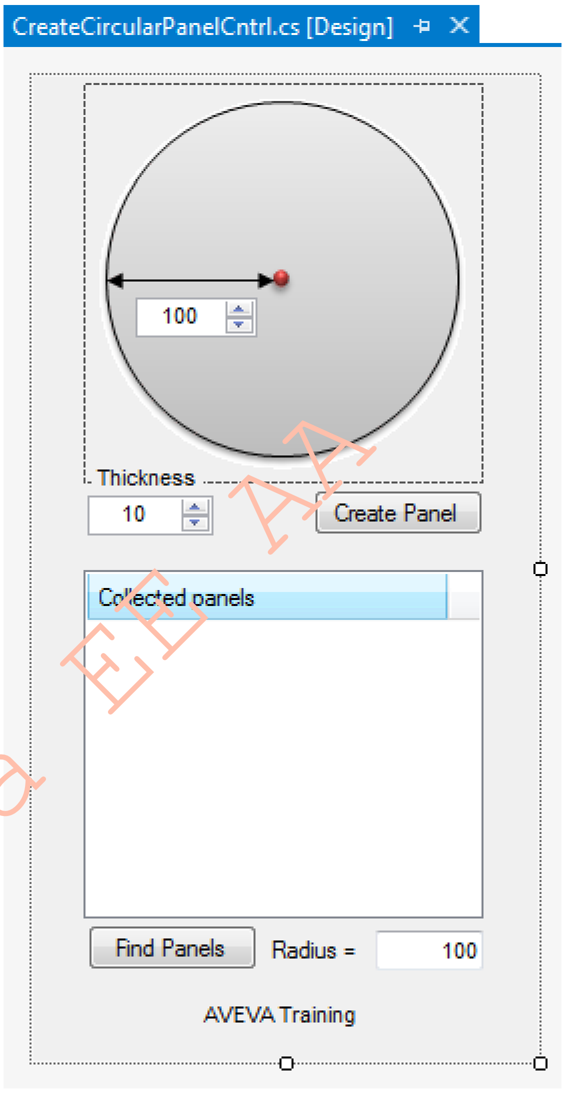
The functionality of the user control will be added during subsequent exercises.

Changing the ListView to **View** = **Detail** will allow the heading to be displayed.

Create a new command to load the user control, call it **CreateCircularPanelCmd.cs**

As an assembly can only have one addin, update **PdfViewerAddin.cs** to make it more generic and register this new command.

Test the new command is registered correctly. Create a new .uic file and a new button to display the form.



CHAPTER 6

6 Database Interface

The database interface provides one class which represents all types of dabacon elements. The class name is **DbElement** and it is implemented in **Aveva.Pdms.Database** assembly under namespace of the same name.

ⓘ *Most of AVEVA assemblies depend on Aveva.Pdms.Utilities assembly. Visual Studio will prompt for adding also this assembly to your project references if not yet added.*

6.1 DbElement

The **DbElement** class is the most widely used class and it covers a large proportion of the database functionality that will be used in practice.

The methods fall into the following groups:

- Navigation
- Querying of attributes
- Database modifications
- Storage of rules and expressions
- Comparison across sessions

DbElement is a generic object that represents all database elements regardless of their type.

6.1.1 Getting instance of DbElement

To get an instance of any database element use the static method **GetElement(…)** on the DbElement.

This method provides two overrides to get an element from its name or reference number and type. The second override is used in the following example where we know the name of the element

Example:

```
DbElement element = DbElement.GetElement("/Site1");
```

If the element cannot be found then the method returns a **null** DbElement. To check if the element has been found, it can be tested using the **IsNull** property.

An element may exist but may not be valid (e.g. a deleted element). There is an **IsValid** property to test if an element is valid.

Getting the element **/*** will return the world element of the current default database

6.1.2 Current Element

To get an instance of the current element (CE) the Element property of the **CurrentElement** class can be used. The **CurrentElement** class is implemented in **Aveva.Pdms.Shared** assembly under namespace of the same name.

Example:

```
DbElement element = Aveva.Pdms.Shared.CurrentElement.Element
```

The **CurrentElement** class also provides a **CurrentElementChanged** event which can be used to handle CE changed events.

6.1.3 Getting Element Type

As all elements are represented by the same class, it is necessary to query the element for its type.

There is a method **GetElementType()** on the **DbElement** class that provides the type of element as an instance of **DbElementType**.

A specific **DbElementType** can be obtained in several ways:

- Use the globally defined instances in **DbElementTypeInstance**. This is the recommended and easiest way to obtain a DbElementType. Example: **DbElementTypeInstance.EQUIPMENT**.
- Use DbElementType.GetElementType(…) method to get type via name. Useful in case of UDETs – must be given with colon.
- Use DbElementType.GetElementType(…) method to get type via hash value. Useful when stored outside the AVEVA application.

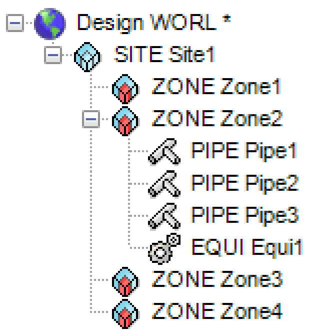
Example:

```
if (elem.GetElementType() == DbElementTypeInstance.EQUIPMENT)
```

6.1.4 Navigating

PDMS data is stored within the database in a hierarchy structure. Starting at the **World** of the database, each element has an owning element and potentially member elements.

There are basic methods to navigate the primary hierarchy provided. For example, consider the following hierarchy structure:



Starting at **/Zone2**, the following methods can be used to navigate:

```
DbElement zone2 = DbElement.GetElement("/Zone2")
DbElement temp;
temp = zone2.Next();             // temp is now /Zone3
temp = zone2.Previous;           // temp is now /Zone1
temp = zone2.Owner;              // temp is now /Site1
temp = zone2.FirstMember();      // temp is now /Pipe1
DbElement pipe1=temp;            // pipe1 is /Pipe1
temp = zone2.FirstMember(DbElementTypeInstance.EQUIPMENT); // temp is /Equi1
temp = zone2.LastMember();       // temp is /Equi1
temp = pipe1.Next(DbElementTypeInstance.EQUIPMENT);  // temp is /Equi1
temp = pipe1.Previous;           // temp is 'null'
```

6.2 Element Attributes

The attributes available on a given **DbElement** will depend on its type.

For example, a SITE will have different attributes to a BRAN.

For this reason, attributes are accessed through generic methods rather than specific methods. These generic methods pass in the identity of the attribute being queried (**DbAttribute**). There are separate methods for each attribute type (int, double etc), plus separate methods for single values or arrays.

6.2.1 Getting an Attribute

There is a general method **GetAttribute(…)** on the **DbAttribute** class that returns a **DbAttribute** instance. This class provides information about an attribute: Name, Category, Description, possible values, etc.

For example, to get specific attribute values from **DbElement Element:**

```
Element.GetString(DbAttributeInstance.NAMN);
Element.GetBool(DbAttributeInstance.LOCK);
Element.GetPosition(DbAttributeInstance.POS);
```

Supported types of attributes:

- int
- double
- bool
- string
- DbElement
- DbElementType
- DbAttribute
- Position
- Direction
- Orientation
- Expresion

If an attribute is not valid for a given object the system will raise an exception.

To avoid exceptions there are a set of methods that return false if an attribute cannot be accessed:

```
Element.GetValidString(DbAttributeInstance.NAMN, ref name);
Element.GetValidBool(DbAttributeInstance.LOCK, ref locked);
```

Also a general method for validating attributes can be used:

```
Element.IsAttributeValid(DbAttributeInstance.TDIR)
```

6.2.2 Attribute Qualifier

Many attributes are able to take a qualifier, describing extra information about the attribute. Some examples of when qualifiers used are:

- Querying a **ppoint** position (**PPOS**) requires the ppoint number.
- A direction/position may be queried with respect to another element.

For example :

```
DbQualifier q = new DbQualifier();
q.wrtQualifier = DbElement.GetElement("/*");
Position pos = CE.GetPosition(DbAttributeInstance.POS, q);
```

The **DbQualifier** class represents the qualifier.

This can hold any type of qualifier, i.e. int, double, string, DbElementType, Attribute, position, direction, orientation. It can hold multiple qualifier values if required by the attribute being queried.

A set of overloaded **GetAttribute** methods all can take a qualifier as an extra argument.

ⓘ *There is a separate method, **wrtQualifier()**, on **DbQualifier** to set the wrt (with respect to) qualifier.*

For more information about pseudo attributes and qualifiers refer to the Data Model Reference Manual.

6.2.3 Setting an Attribute

As with getting attributes, there is a family of overloaded methods for setting attributes depending on the attribute type:

```
Element.SetAttribute(DbAttributeInstance.DESC, "Example description");
Element.SetAttribute(DbAttributeInstance.HEIGHT, 12.0);
```

There is also a boolean method available **IsAttributeSetable** to test if a given attribute may be set.

6.3 Creating Elements

Creation of new elements is straightforward. It is possible to create elements:

- Below given element
- After given element
- Before given element

When creating element, the position must be given. If the position is beyond the end of the current members list, it will create it at the end.

```
Element.Create(1, DbElementTypeInstance.PLOOP);
Element.CreateAfter(DbElementTypeIntance.PLOOP);
Element.CreateBefore(DbElementTypeInstance.PLOOP);
```

If a required element type cannot be created at the required point an exception will be raised. The method **IsCreatable** can be used to test if an element of given type can be created at a given location.

```
if (Element.IsCreatable(DbElementTypeInstance.PLOOP))
      Element.Create(1, DbElementTypeInstance.PLOOP)
```

6.4 Deleting Elements

There is a **Delete()** method provided by the **DbElement** class that deletes an element. All descendants in the primary hierarchy will be deleted. The **IsHierarchyDeleteable** property can be used to determine if an element and all its descendants can be deleted.

6.5 Moving Elements

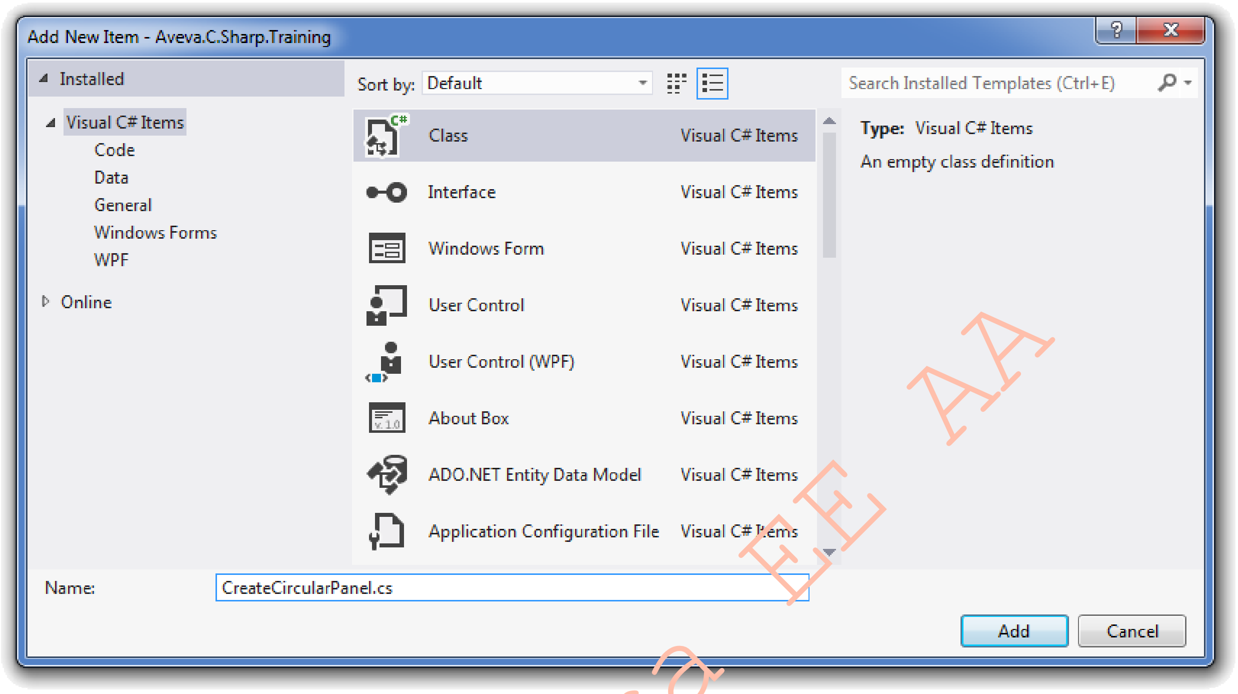
An element can be moved to a different location in the primary hierarchy. There are methods to:

- Insert before a given element
- Insert after a given element
- Insert into members list at the last position

ⓘ *Currently, an element may only be moved within the same database. Across databases it must be copied and then deleted.*

Worked Example – Creating circular panel elements

1 Create a new class called **CreateCircularPanel.cs**. This will be a PML .NET object that will allow PML to create circular panel elements directly.


Add New Item - Aveva.C.Sharp.Training
Installed
Visual C# Items
Code
Data
General
Windows Forms
WPF
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class | Visual C# Items
Interface | Visual C# Items
Windows Form | Visual C# Items
User Control | Visual C# Items
User Control (WPF) | Visual C# Items
About Box | Visual C# Items
ADO.NET Entity Data Model | Visual C# Items
Application Configuration File | Visual C# Items
Type: Visual C# Items
An empty class definition
Add
Cancel

Name: CreateCircularPanel.cs

2 Update the class to the following:

```
using System;
using Aveva.Pdms.Database;
using Aveva.Pdms.Geometry;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class CreateCircularPanel
    {
        [PMLNetCallable()]
        public CreateCircularPanel()
        {
        }

        [PMLNetCallable()]
        public void Assign(CreateCircularPanel that)
        {
        }
    }
}
```

*References to **Aveva.Pdms.Database** and **Aveva.Pdms.Geometry** will need to be added.*

3 Add private properties to represent the radius and thickness of the panel.

```
private double _radius;
private double _thickness;
```

Add NetCallable properties to get and set the stored values:

```
[PMLNetCallable()]
public double Radius
{
    get { return _radius; }
    set { _radius = value; }
}

[PMLNetCallable()]
public double Thickness
{
    get { return _thickness; }
    set { _thickness = value; }
}
```

Ensure the Assign method is updated to transfer the values of the private properties.

4 Add a NetCallable method that will check whether a panel can be created below a given element.

```
[PMLNetCallable()]
public bool CanBeCreated(string ElementName)
{
    // Get the supplied name as a DbElement
    DbElement Element = DbElement.GetElement(ElementName);
    return Element.IsCreatable(DbElementTypeInstance.PANEL);
}
```

5 Add a method that will create a circular panel below the supplied element:

```
[PMLNetCallable()]
public bool CreateBelowElement(string ElementName)
```

The first part of the method needs to be a DbElement instance of the supplied element name. It is worth checking that the element is both valid and not null.

```
// Get the supplied name as a DbElement
DbElement Element = DbElement.GetElement(ElementName);

// Check the element is valid
if (Element.IsNull || !Element.IsValid)
    return false;
```

Before trying to create a new element, check a Panel element can be created. If it can, create it:

```
// Check a PANEL can be created
if (!CanBeCreated(ElementName))
    return false;

// Create the PANEL element
DbElement panel = Element.Create(1, DbElementTypeInstance.PANEL);
panel.SetAttribute(DbAttributeInstance.DESC, "CIRCULAR");
```

Add the following code to finish the method. This will create and size the remaining elements that will define the circular panel:

```
        // If the PANEL is not valid, stop...
        if (!panel.IsValid)
            return false;

        // Create the PLOO as the first member of the PANEL
        DbElement ploop = panel.Create(1, DbElementTypeInstance.PLOOP);

        // Set the thickness of the PLOO, based on the user value
        ploop.SetAttribute(DbAttributeInstance.HEIG, _thickness);

        // Create the 4 PAVEs required to define the shape of the PANEL (with FRAD set)
        DbElement Pave1 = ploop.Create(1, DbElementTypeInstance.PAVERT);
        Pave1.SetAttribute(DbAttributeInstance.POS, Position.Create(-_radius, -_radius, 0));
        Pave1.SetAttribute(DbAttributeInstance.FRAD, _radius);

        DbElement Pave2 = Pave1.CreateAfter(DbElementTypeInstance.PAVERT);
        Pave2.SetAttribute(DbAttributeInstance.POS, Position.Create(-_radius, _radius, 0));
        Pave2.SetAttribute(DbAttributeInstance.FRAD, _radius);

        DbElement Pave3 = Pave2.CreateAfter(DbElementTypeInstance.PAVERT);
        Pave3.SetAttribute(DbAttributeInstance.POS, Position.Create(_radius, _radius, 0));
        Pave3.SetAttribute(DbAttributeInstance.FRAD, _radius);

        DbElement Pave4 = Pave3.CreateAfter(DbElementTypeInstance.PAVERT);
        Pave4.SetAttribute(DbAttributeInstance.POS, Position.Create(_radius, -_radius, 0));
        Pave4.SetAttribute(DbAttributeInstance.FRAD, _radius);

        return true;
    }
```

6 Add a method that will create a circular panel below the current element:

```
        [PMLNetCallable()]
        public bool CreateBelowCe()
        {
            DbElement Element = CurrentElement.Element;
            bool result = CreateBelowElement(Element.GetString(DbAttributeInstance.NAME));
            return result;
        }
```

ⓘ *Reference to **Aveva.Pdms.Shared** will need to be added along with a "using" statement.*

7 Build and test the class within PDMS:

```
import |C:\AVEVA\Training\NET\Aveva.C.Sharp.Training|
using namespace |Aveva.C.Sharp.Training|
!circularPanel = object CREATECIRCULARPANEL()

!circularPanel.radius(1000)
!circularPanel.thickness(10)

q var !circularPanel.canBeCreated(!!ce.name)
q var !circularPanel.createBelowCe()
```

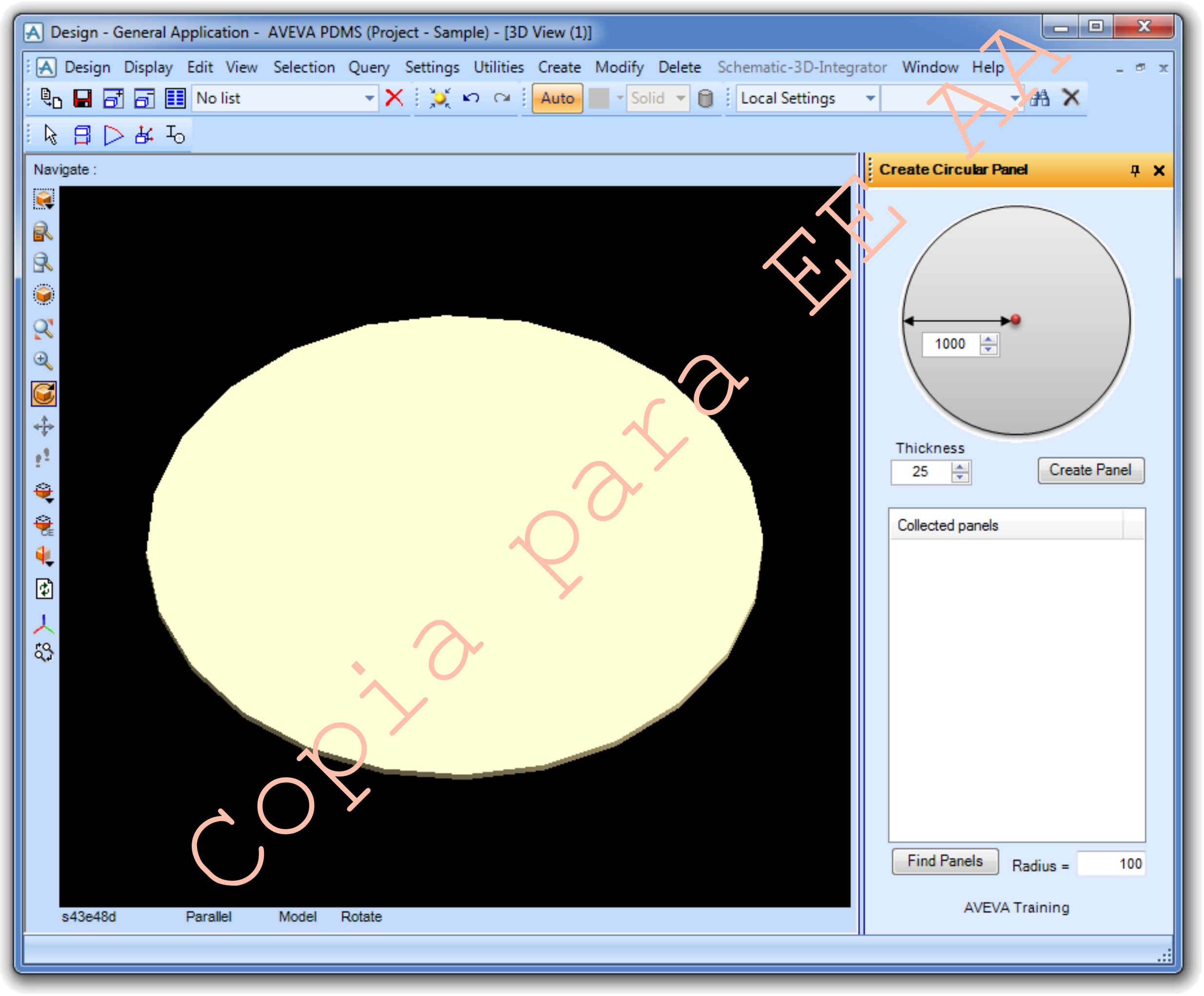
Exercise 6 – Adding functionality to the .NET Addin

Update the user control **CreateCircularPanelCntrl.cs** to make use of the class created during the worked example.

Add a click event method to the "**Create Panel**" button on the control.

The method should:

1) Create an instance of the **CreateCircularPanel** class
2) Check that a panel can be created below the current element
3) If it can, update the size information on the instance (taken from the form)
4) Create a new circular panel below the current element


Design - General Application - AVEVA PDMS (Project - Sample) - [3D View (1)]
Design Display Edit View Selection Query Settings Utilities Create Modify Delete Schematic-3D-Integrator Window Help
No list
Auto
Solid
Local Settings
Navigate :
Create Circular Panel
1000
Thickness
25
Create Panel
Collected panels
Find Panels
Radius =
100
AVEVA Training
s43e48d
Parallel
Model
Rotate

Build and test the addin within PDMS.

Consider what happens when a panel cannot be created. What feedback would a user expect?

CHAPTER 7

7 Collections and Filters

Navigating the database is one technique of accessing data, but is limited when trying to locate specific information. Using collections is an efficient way of finding and returning specific database information.

7.1 Collection

DBElementCollection is an object implemented in the **PDMSFilters** assembly under the **Aveva.PDMS.Database.Filters** namespace. It is specially designed for efficient searching and working with many DbElements.

DBElementCollection constructor will take:

- No parameters
- Root element (type DbElement)
- Root element and Filter (type BaseFilter)

For example:

```
DbElement Scope = DbElement.GetElement("/Site1");
DBElementCollection Collect = new DBElementCollection(Scope);
foreach (DbElement Element in Collect)
{
    string name = Element.GetString(DbAttributeInstance.NAMN);
}
```

The resulting collection will contain all the elements under /Site1 element (i.e. 9 elements).

7.2 Filters

Collecting all the elements for a given level in the hierarchy is useful, but sometimes a more considered search is required. Database filter is an object used to filtering elements in a database hierarchy. The filter object is used together with collections.

For example:

```
DbElement Scope = DbElement.GetElement("/Site1");
EquipmentFilter Filter = new EquipmentFilter();
DBElementCollection Collect = new DBElementCollection(Scope, Filter);
```

EquipmentFilter is a predefined filter that will only collect equipment elements. A collection will collect all the elements that match the filters criteria for a given scope (if provided).

The following provides some examples of the predefined filters in use.

ⓘ For information on all the provided predefined filters in ***Aveva.PDMS.Database.Filters***. *use the Object Browser within Visual Studio.*

7.2.1 TypeFilter

This is used to collect elements by type. The constructor takes **DbElemetType** or **DbElemetType[]** of types that the collection should contain. For example, to filter only PIPE elements:

```
TypeFilter PipeFilter = new TypeFilter(DbElementTypeInstance.PIPE);
```

7.2.2 AttributeFalse/TrueFilter

AttributeFalseFilter and **AttributeTrueFilter** are used to querying elements by attribute logical value.

The constructor takes a **DbAttribute** object as a parameter. The collection will contain all DbElements where the value of the given attribute is false or true depending of filter type. For example, a filter to find all the locked elements:

```
DbAttribute LockAtt = DbAttributeInstance.LOCK;
AttributeTrueFilter LockFilter = new AttributeTrueFilter(LockAtt);
```

7.2.3 AttributeRefFilter

AttributeRefFilter can be used to test if a reference attribute of DbElement matches the given value.

The constructor takes a **DbAttribute** and a **DbElement** test value. The collection will contain all elements where the reference attribute refers to the given DbElement. For example, this filter would collect all elements that are owned by **/Zone2**:

```
DbElement Owner = DbElement.GetElement("/Zone2");
DbAttribute OwnerAtt = DbAttributeInstance.OWNER;
AttributeRefFilter Ownerfilter = new AttributeRefFilter(OwnerAtt, Owner);
```

7.2.4 And/OrFilter

These filters are used to build a more complex query by logical constructions of defined filters. The constructor can be used to combine two filters together, or supplied with an array of filters. More can also be added by using the **Add(BaseFilter)** method:

For example, to collect all the locked pipes within /Zone2:

```
AndFilter LockedPipes = new AndFilter(PipeFilter, LockFilter);
LockedPipes.Add(OwnerFilter);
```

For example, to collect all the locked elements or pipes within /Zone2:

```
OrFilter LockedOrPipes = new OrFilter(PipeFilter, LockFilter);
AndFilter Filter = new AndFilter(LockedOrPipes, OwnerFilter);
```

7.2.5 BelowFilter

This filter can be used to test if an element is below an element for which the given filter is true. Constructor takes BaseFilter parameter. For example, to collect all elements that are below all locked pipes:

```
BelowFilter belowLockedPipes = new BelowFilter(lockedPipes);
```

7.3 Custom filters

Sometimes a specific collection is required that cannot be achieved with the predefined filters. In this case, a custom filter could be created.

For this AVEVA provides a **BaseFilter** class that is an interface for all database filters. To create a custom database filter, define a public class derived from the **BaseFilter** and override methods: **Clone()**, **Valid(DbElement element)** and **ScanBelow(DbElement element).**

The class would take the following form:

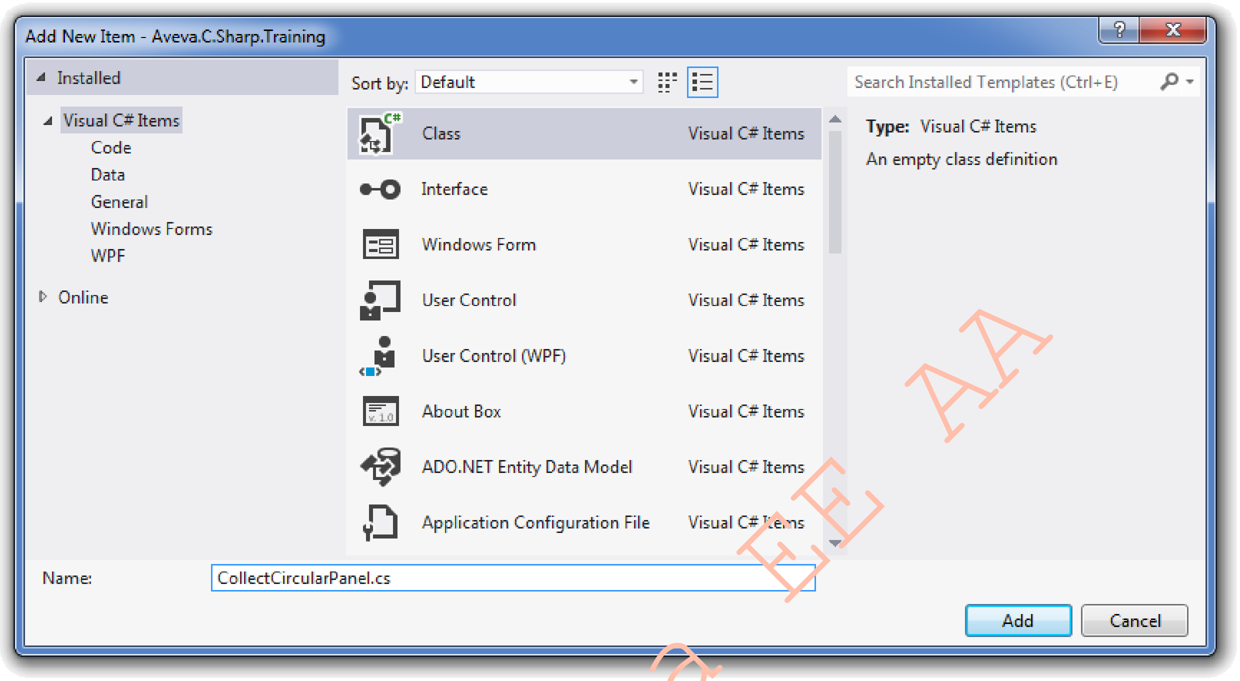
```
public class CustomFilter : BaseFilter
{
    public override object Clone()
    {
        return new CustomFilter();
    }

    public override bool ScanBelow(DbElement element)
    {
        return true;
    }

    public override bool Valid(DbElement element)
    {
        // check element against criteria and
        // return true or false
        return false;
    }
}
```

The methods provide the following:

- **Clone()** returns an exact copy of a filter object
- **ScanBelow()** decides if search under this element should be performed.
- **Valid()** decides which elements will match the filter criteria.

Worked Example – Collecting circular panel elements

1 Create a new class called **CollectCircularPanel.cs**. This will be a PML .NET object that will allow PML to create circular panel elements directly.


Add New Item - Aveva.C.Sharp.Training

Class	Visual C# Items	
Interface	Visual C# Items	
Windows Form	Visual C# Items	
User Control	Visual C# Items	
User Control (WPF)	Visual C# Items	
About Box	Visual C# Items	
ADO.NET Entity Data Model	Visual C# Items	
Application Configuration File	Visual C# Items	

Type: Visual C# Items
An empty class definition

Name: CollectCircularPanel.cs

2 Update the class to the following:

```
using System;
using System.Collections;
using Aveva.Pdms.Database;
using Aveva.PDMS.Database.Filters;
using Aveva.Pdms.Shared;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class CollectCircularPanel
    {
        [PMLNetCallable()]
        public CollectCircularPanel()
        {
        }

        [PMLNetCallable()]
        public void Assign(CollectCircularPanel that)
        {
        }
    }
}
```

*A reference to **PDMSFilters** will need to be added. This will make available Aveva.PDMS.Database.Filters. Notice how the namespace is different to the assembly name.*

3 Add a method to collect PANEL elements below the current element

```
        [PMLNetCallable()]
        public Hashtable Collect()
        {
            DbElement Element = CurrentElement.Element;
            TypeFilter PanelFilter = new TypeFilter(DbElementTypeInstance.PANEL);
            DBElementCollection Collect = new DBElementCollection(Element, PanelFilter);

            Hashtable Collected = new Hashtable();
            int i = 0;
            foreach (DbElement Panel in Collect)
            {
                i++;
                Collected.Add(i, Panel.GetString(DbAttributeInstance.NAME));
            }

            return Collected;
        }
```

4 Add a new class called **CircularPanelFilter.cs**. This will hold the custom filter that will ensure only circular panels are picked.

Add the following code to the class:

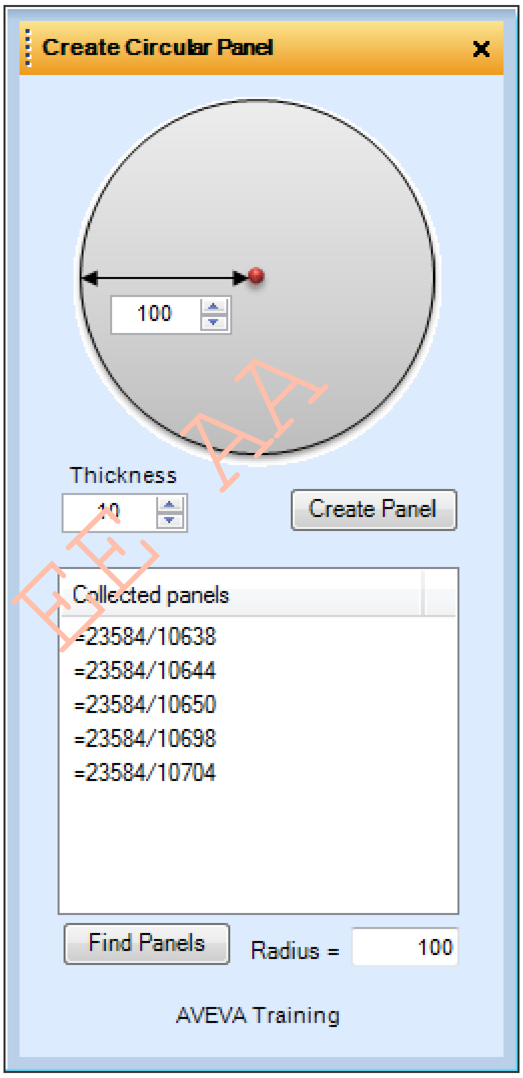
```
using System;
using Aveva.Pdms.Database;
using Aveva.PDMS.Database.Filters;

namespace Aveva.C.Sharp.Training
{
    public class CircularPanelFilter : BaseFilter
    {
        public override object Clone()
        {
            return new CircularPanelFilter();
        }

        public override bool ScanBelow(DbElement Element)
        {
            return true;
        }

        public override bool Valid(DbElement Element)
        {
            string description = Element.GetString(DbAttributeInstance.DESC);

            if (description == "CIRCULAR")
                return true;
            else
                return false;
        }
    }
}
```

5 Update the Collect method to include and use an **AndFilter()**

```
AndFilter CircularPanelFilter = new AndFilter(PanelFilter, new CircularPanelFilter());
DBElementCollection Collect = new DBElementCollection(Element, CircularPanelFilter);
```

Build and test in PDMS by collecting elements created by **CreateCircularPanel.cs**

Exercise 7 – Adding collection functionality to the .NET Addin



Update CircularPanelFilter.cs so it can filter based on the radius of a panel.

Add a private member to the class to store the radius value and update the **Clone** and **Valid** methods to suit. Also add a constructor method that accepts an argument of the required radius value.

Consider how the **Valid** method should be updated to filter panels of the required radius.

Update the method to:

1) Get the PLOO element from below the PANE
2) Get all the PAVE elements from below the PLOO
3) Check that there are four PAVE elements
4) Check that each of the PAVE elements have the FRAD attribute equal to the required radius

Update the **Collect** method in CollectCircularPanel.cs to accept a radius argument.

Add a click event method to the "**Find Panels**" button on CreateCircularPanelCntrl.cs

This method should:

1) Use an instance of CollectCircularPanel to collect the panels of the required size.
2) Loop through the collected elements and add them to the list.

Consider how a **SelectIndexChanged** event can be added to the list view so a user can navigate to the collected panels.

8 PdmsStandalone Interface

Aveva.PDMS.Standalone allows an application that sits outside PDMS to have access to PDMS databases. This interface loads a standalone version of PDMS with access to loaded projects.

8.1 Making use of PDMSStandalone

After importing the **Aveva.Pdms.Standalone** assembly, the **PdmsStandalone** class will be available for use. The first stage is to try and start it using the **Start** method:

```
if (!PdmsStandalone.Start())
{
    PdmsStandalone.ExitError("Failed to start");
}
```

If a false is returned, then the class cannot be used. Calling the **ExitError** method closes the program down. If it could be started, the **Open** method (supplied with the project code, user name, password and MDB) can be used:

```
if (!PdmsStandalone.Open("SAM", "SYSTEM", "XXXXXX", "SAMPLE"))
{
    PdmsStandalone.ExitError("Failed to login");
}
```

If this returns a false, then the log in details are incorrect.

ⓘ *It is typical for these two blocks to be run within a Try..Catch statement to cover any unforeseen errors.*

8.2 Setting the environment

The standalone application needs the same environment variables to be set before it can run. The easiest way to achieve this is to use the **pdms.bat** file from the PDMS installation, with the following update:

Replace:	With

`echo running: %monexe%\%executable% %args%`
`cmd/c "%monexe%\%executable%" %args%`
`goto end`
	`echo running Example Standalone Application`
	`cmd/c "Standalone.Example.exe"`
	`goto end`

ⓘ *Where Standalone.Example.exe is the name of the compiled executable, saved into the same directory as the modified .bat file*

It is also possible to supply the **Start** method of **PdmsStandalone** with a **Hashtable** argument of environment variables. This technique would remove the need to use a .bat file.

8.3 Accessing specific modules

The ability to read from and write to a database is controlled by which PDMS module is being used.

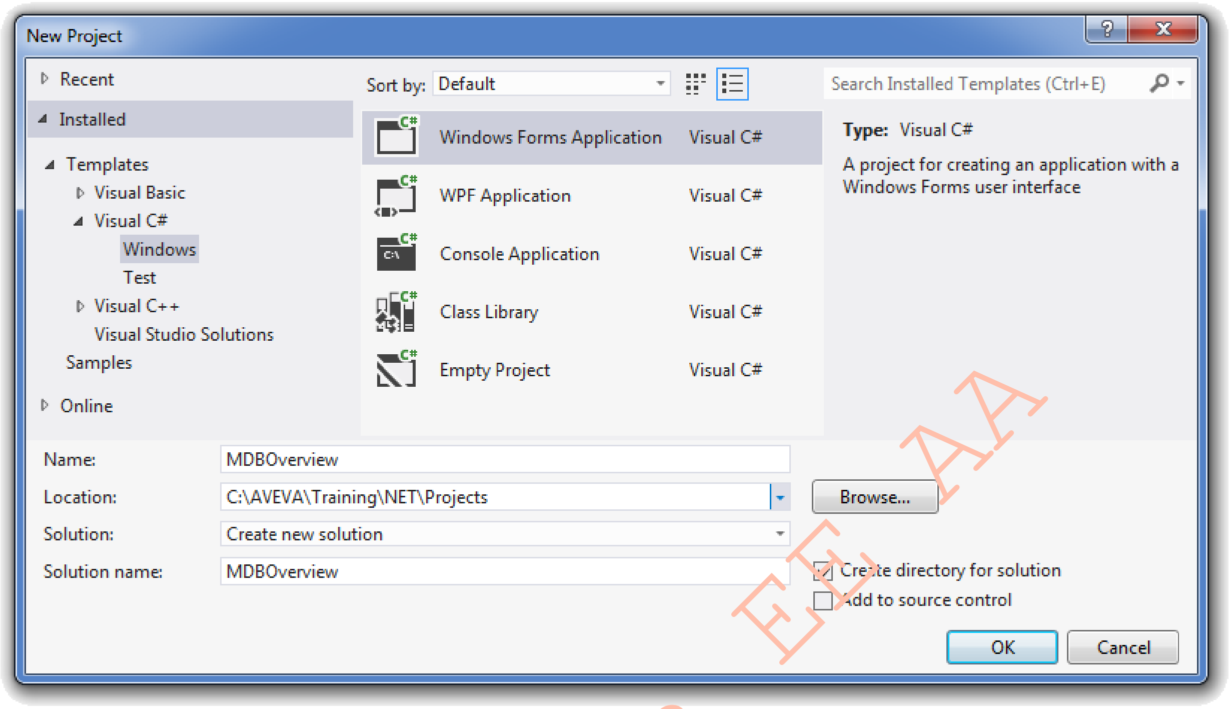
For example, PDMS Design can write 3D data, read catalogue data but cannot read Draft data.

The module definitions in set in the admin module using **Project > Module Definitions…**

The module number can be supplied as an argument to the **Start** method of **PdmsStandalone**.

Worked Example – MDB Overview

1 Create a new project called **MDBOverview**. Select **Visual C#** and **Windows Forms Application**.


New Project

- Recent
- Installed
  - Templates
    - Visual Basic
    - Visual C#
      - Windows
      - Test
    - Visual C++
    - Visual Studio Solutions
  - Samples
- Online

Sort by: Default

Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Class Library	Visual C#
Empty Project	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: MDBOverview
Location: C:\AVEVA\Training\NET\Projects
Solution: Create new solution
Solution name: MDBOverview

[✓] Create directory for solution
[ ] Add to source control

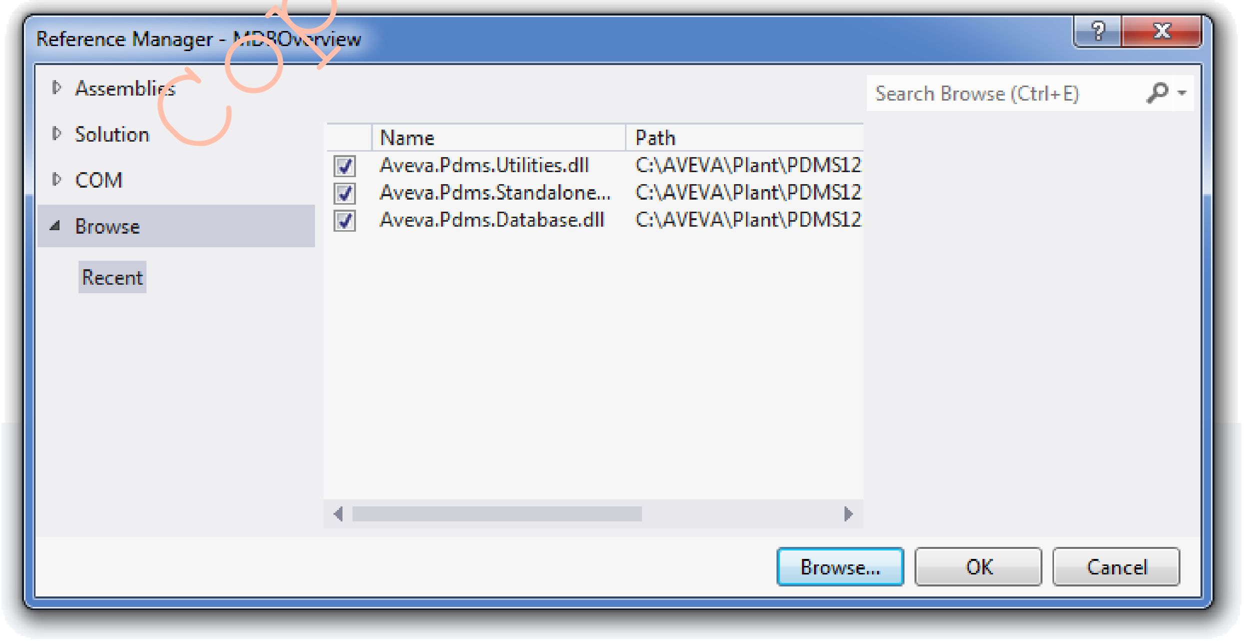
2 Set the Target Framework as **.NET Framework 3.5**

Ensure the build CPU for **Release** is **x86**. This may require a new setting to be created.

Add References to:

- Aveva.Pdms.Database
- Aveva.Pdms.Standalone
- Aveva.Pdms.Utilities

Set a **Reference Path** to **C:\AVEVA\Plant\PDMS12.1.SP4** in the application properties.


Reference Manager - MDBOverview

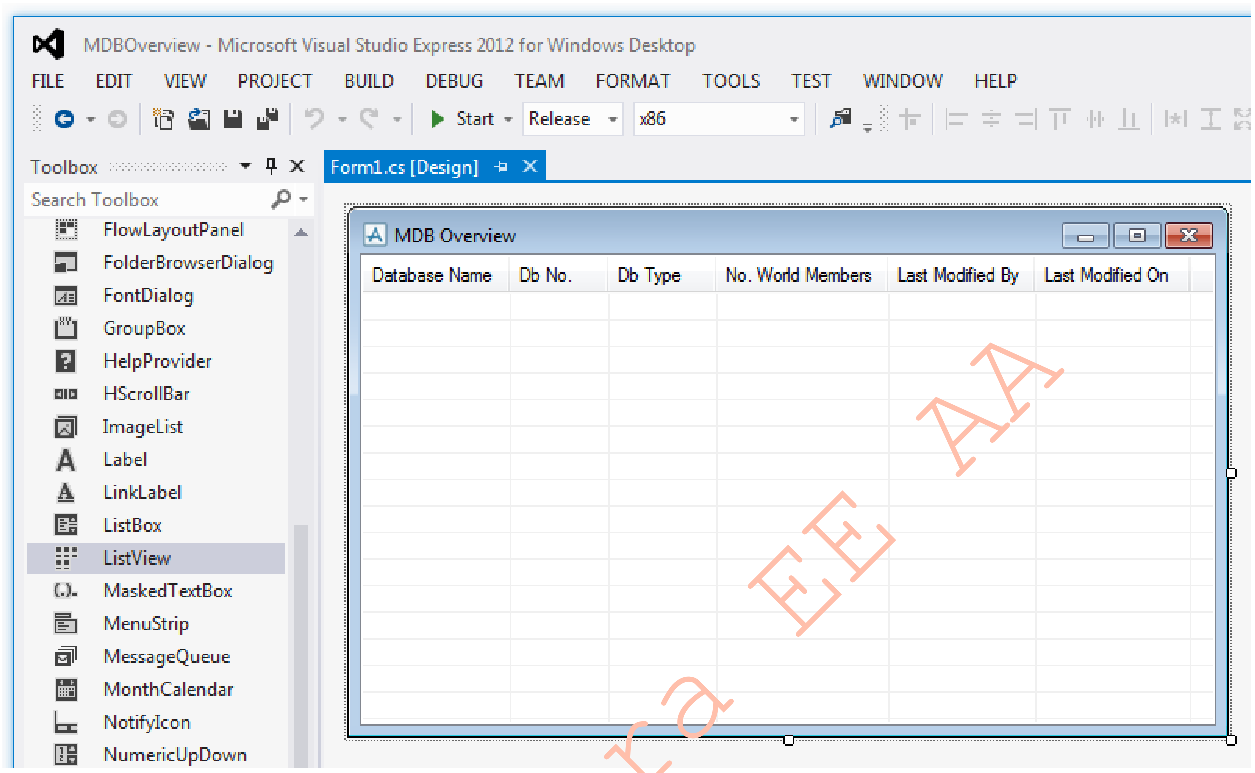
- Assemblies
- Solution
- COM
- Browse
  - Recent

Search Browse (Ctrl+E)

	Name	Path
✓	Aveva.Pdms.Utilities.dll	C:\AVEVA\Plant\PDMS12
✓	Aveva.Pdms.Standalone...	C:\AVEVA\Plant\PDMS12
✓	Aveva.Pdms.Database.dll	C:\AVEVA\Plant\PDMS12

3 Add **ListView** control to Form1.

Set **Gridlines** to **True**, **View** to **Detail**, **Dock** to **Fill** and create the required columns shown in the screenshot:



On Form1, set the **Text** as **MDB Overview** and the **Icon** to the supplied one. The same icon can be Application icon.

4 Add the following code to the constructor method of the form (after `InitializeComponent();`)

```
try
{
    // Try starting the PdmsStandalone
    if (!PdmsStandalone.Start())
    {
        PdmsStandalone.ExitError("Failed to start");
    }
    // Try opening the SAM project using the PDMSStandalone
    if (!PdmsStandalone.Open("SAM", "SYSTEM", "XXXXXX", "SAMPLE"))
    {
        PdmsStandalone.ExitError("Failed to login");
    }
}
catch
{
    PdmsStandalone.ExitError("Failed to start");
}
```

This code will establish the link to PDMS. Once established, the code will have access to databases through the API (as the previous examples).

ⓘ *Add the correct "using" statements to the class as required*

5 Add the following code after the previous:

```
// Get the DB array from the current MDB
 Db[] dbs = MDB.CurrentMDB.GetDBArray();

 // Loop through the databases and compile the information
 foreach (Db d in dbs)
 {
     ListViewItem Litem = this.listView1.Items.Add(d.Name);
     Litem.SubItems.Add(d.Number.ToString());
     Litem.SubItems.Add(d.Type.ToString());
     Litem.SubItems.Add(d.WorldMembers().Length.ToString());
     Litem.SubItems.Add(d.LatestSessionOnDb.User);
     Litem.SubItems.Add(d.LatestSessionOnDb.Date.ToLongDateString());
 }
```

This code uses the current MDB to access the available databases. It then loops through and extracts information to present on the form.

6 Add a post-build event to the project:

copy /y "$(TargetPath)" "C:\AVEVA\Plant\PDMS12.1.SP4\$(TargetFileName)"

Build the project and debug any build issues.

Confirm that **MDBOverview.exe** is now available within the PDMS installation folder.

7 Copy **training.bat** and rename the file to be **MDBOverview.bat**.

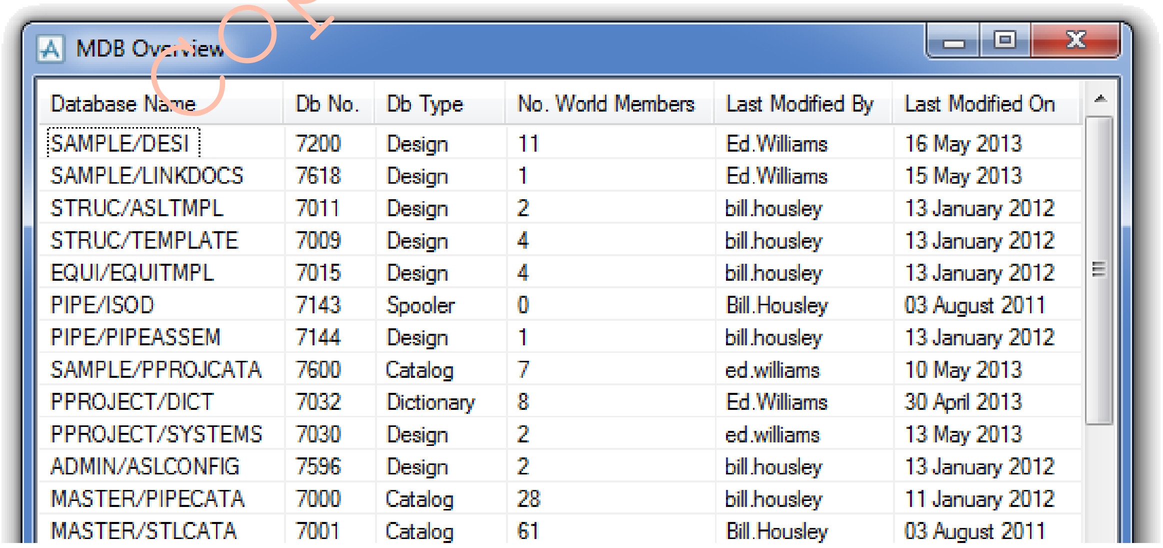
Edit the file and change this line:

cmd/c "%monexe%\%executable%" %args%

To the following:

cmd/c “C:\AVEVA\Plant\PDMS12.1.SP4\MDBOverview.exe”

8 Save and run the new .bat file. Test the **MDBOverview** Application.



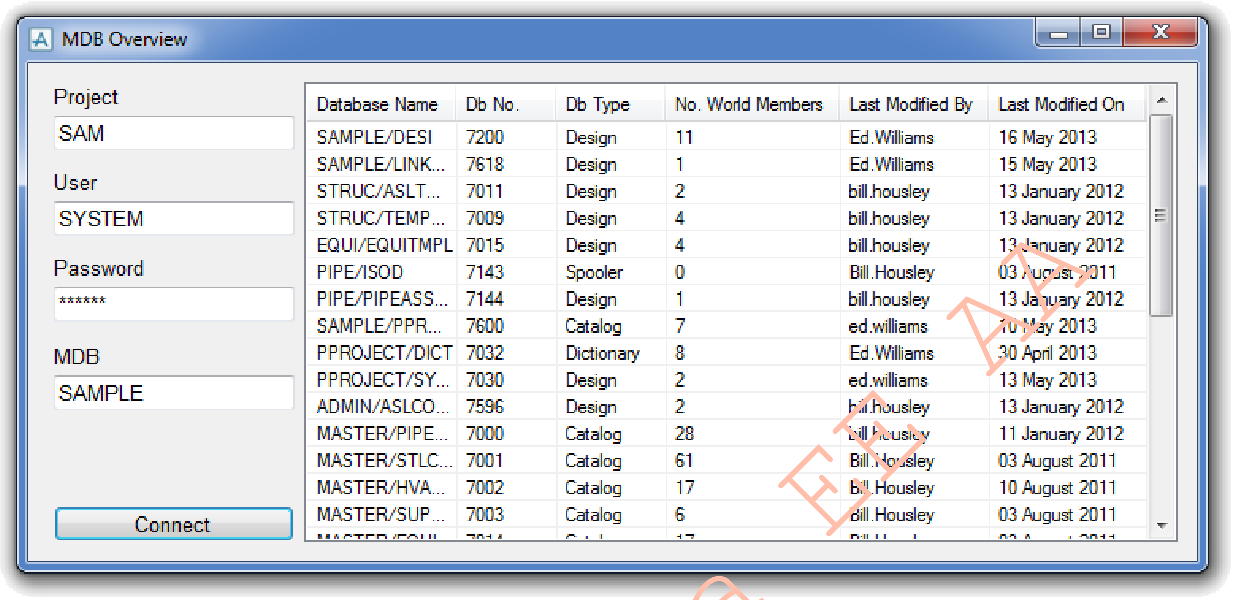
Database Name	Db No.	Db Type	No. World Members	Last Modified By	Last Modified On
SAMPLE/DESI	7200	Design	11	Ed.Williams	16 May 2013
SAMPLE/LINKDOCS	7618	Design	1	Ed.Williams	15 May 2013
STRUC/ASLTMPL	7011	Design	2	bill.housley	13 January 2012
STRUC/TEMPLATE	7009	Design	4	bill.housley	13 January 2012
EQUI/EQUITMPL	7015	Design	4	bill.housley	13 January 2012
PIPE/ISOD	7143	Spooler	0	Bill.Housley	03 August 2011
PIPE/PIPEASSEM	7144	Design	1	bill.housley	13 January 2012
SAMPLE/PPROJCATA	7600	Catalog	7	ed.williams	10 May 2013
PPROJECT/DICT	7032	Dictionary	8	Ed.Williams	30 April 2013
PPROJECT/SYSTEMS	7030	Design	2	ed.williams	13 May 2013
ADMIN/ASLCONFIG	7596	Design	2	bill.housley	13 January 2012
MASTER/PIPECATA	7000	Catalog	28	bill.housley	11 January 2012
MASTER/STLCATA	7001	Catalog	61	Bill.Housley	03 August 2011

Exercise 8 – Extending the standalone example

The standalone example has a single set of login details hard-coded into it. This limits the use of the application. It would be better if the user could provide specific login details before connecting.

Update the worked example to allow the user of the form to change the login details.

Add four **Label** controls, four **TextBox** controls and a **Button** control (as the below screenshot).



Project	SAM
User	SYSTEM
Password	******
MDB	SAMPLE

Database Name	Db No.	Db Type	No. World Members	Last Modified By	Last Modified On
SAMPLE/DESI	7200	Design	11	Ed.Williams	16 May 2013
SAMPLE/LINK...	7618	Design	1	Ed.Williams	15 May 2013
STRUC/ASLT...	7011	Design	2	bill.housley	13 January 2012
STRUC/TEMP...	7009	Design	4	bill.housley	13 January 2012
EQUI/EQUITMPL	7015	Design	4	bill.housley	13 January 2012
PIPE/ISOD	7143	Spooler	0	Bill.Housley	03 August 2011
PIPE/PIPEASS...	7144	Design	1	bill.housley	13 January 2012
SAMPLE/PPR...	7600	Catalog	7	ed.williams	10 May 2013
PPROJECT/DICT	7032	Dictionary	8	Ed.Williams	30 April 2013
PPROJECT/SY...	7030	Design	2	ed.williams	13 May 2013
ADMIN/ASLCO...	7596	Design	2	bill.housley	13 January 2012
MASTER/PIPE...	7000	Catalog	28	bill.housley	11 January 2012
MASTER/STLC...	7001	Catalog	61	Bill.Housley	03 August 2011
MASTER/HVA...	7002	Catalog	17	Bill.Housley	10 August 2011
MASTER/SUP...	7003	Catalog	6	Bill.Housley	03 August 2011

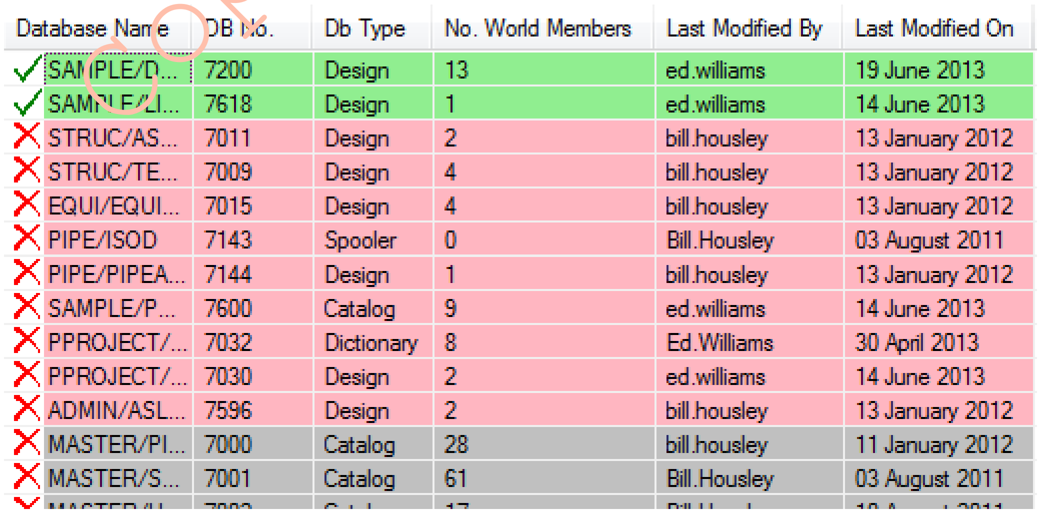
Update the code to use the values entered into the form as the login details.

Build and test the form.

Consider what should happen if the wrong login details are provided. Is a pop-up message appropriate?

Add colour and icons to the interface to indicate read/write access. An ImageList control can be used to make icons available to a ListView control.

- To check if a database is foreign, use `DbAttributeInstance.ISDBFR`
- To check if a user has write access, check if the lock attribute on the world can be set.



	Database Name	DB No.	Db Type	No. World Members	Last Modified By	Last Modified On
✓	SAMPLE/D...	7200	Design	13	ed.williams	19 June 2013
✓	SAMPLE/LI...	7618	Design	1	ed.williams	14 June 2013
✗	STRUC/AS...	7011	Design	2	bill.housley	13 January 2012
✗	STRUC/TE...	7009	Design	4	bill.housley	13 January 2012
✗	EQUI/EQUI...	7015	Design	4	bill.housley	13 January 2012
✗	PIPE/ISOD	7143	Spooler	0	Bill.Housley	03 August 2011
✗	PIPE/PIPEA...	7144	Design	1	bill.housley	13 January 2012
✗	SAMPLE/P...	7600	Catalog	9	ed.williams	14 June 2013
✗	PPROJECT/...	7032	Dictionary	8	Ed.Williams	30 April 2013
✗	PPROJECT/...	7030	Design	2	ed.williams	14 June 2013
✗	ADMIN/ASL...	7596	Design	2	bill.housley	13 January 2012
✗	MASTER/PI...	7000	Catalog	28	bill.housley	11 January 2012
✗	MASTER/S...	7001	Catalog	61	Bill.Housley	03 August 2011

Appendix A – Source code

Appendix A.1 – Source code for Chapter 3

NetTimeZone.cs

```
// NetTimeZone Class
// ========================================================
// Worked example from TM1406
// © AVEVA Solutions Ltd, www.aveva.com

using System;
using System.Collections;
using System.Collections.ObjectModel;

using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class NetTimeZone
    {
        [PMLNetCallable()]
        public NetTimeZone()
        {
        }

        [PMLNetCallable()]
        public void Assign(NetTimeZone that)
        {
        }

        [PMLNetCallable()]
        public string CurrentTimeZone
        {
            get{return TimeZone.CurrentTimeZone.StandardName;}

        }

        [PMLNetCallable()]
        public string GetTimeZoneInfo(string ZoneName)
        {
            try
            {
                DateTimeOffset nowTime = DateTimeOffset.Now;
                TimeZoneInfo UserTimeZone =
                    TimeZoneInfo.FindSystemTimeZoneById(ZoneName);
                DateTimeOffset newTime = TimeZoneInfo.ConvertTime(nowTime, UserTimeZone);

                return newTime.ToString();
            }
            catch (Exception ex)
            {
                throw new PMLNetException(1000, 1, ex.Message);
            }

        }
        [PMLNetCallable()]
        public Hashtable GetSystemTimeZones()
        {
            ReadOnlyCollection<TimeZoneInfo> timeZones =
                TimeZoneInfo.GetSystemTimeZones();
            Hashtable TimeZoneArray = new Hashtable();
            int i = 0;
            foreach (TimeZoneInfo timeZoneInfo in timeZones)
            {
                i++;
                TimeZoneArray.Add(i, timeZoneInfo.Id);
            }

            return TimeZoneArray;
        }
    }
}
```

RandomNumber.cs

```
// RANDOMNUMBER Class
// ========================================================
// Provides PML with the ability to generate Random Numbers
// © AVEVA Solutions Ltd, www.aveva.com

using System;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class RandomNumber
    {

        private bool _rounded;

        [PMLNetCallable()]
        public RandomNumber()
        {
            _rounded = false;
        }

        [PMLNetCallable()]
        public RandomNumber(bool rounded)
        {
            _rounded = rounded;
        }

        [PMLNetCallable()]
        public void Assign(RandomNumber that)
        {
            _rounded = that._rounded;
        }

        [PMLNetCallable()]
        public double Generate()
        {
            Random random = new Random();
            return random.NextDouble();
        }
        private double CalculateValue(double minimum, double maximum)
        {
            return minimum + (maximum - minimum) * Generate();
        }

        [PMLNetCallable()]
        public double Generate(double minimum, double maximum)
        {
            return Generate(minimum, maximum, _rounded);
        }

        [PMLNetCallable()]
        public double Generate(double minimum, double maximum, bool rounded)
        {
            if (rounded)
                return CalculateValue(minimum, maximum);
            else
                return Math.Round(CalculateValue(minimum, maximum));
        }

        [PMLNetCallable]
        public bool Rounded
        {
            get { return _rounded; }
            set { _rounded = value; }
        }

    }
}
```

Appendix A.2 – Source code for Chapter 4

NetCalendarControl.cs

```
// NETCALENDARCONTROL User Control
// ========================================================
// A calendar control for use with on a PML form
// © AVEVA Solutions Ltd, www.aveva.com

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Collections;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    public partial class NetCalendarControl : UserControl
    {
        [PMLNetCallable()]
        public NetCalendarControl()
        {
            InitializeComponent();
        }

        [PMLNetCallable()]
        public void Assign(NetCalendarControl that)
        {
        }

        [PMLNetCallable()]
        public string StartDate
        {
            get { return this.monthCalendar1.SelectionStart.ToLongDateString(); }
        }

        [PMLNetCallable()]
        public string EndDate
        {
            get { return this.monthCalendar1.SelectionEnd.ToLongDateString(); }
        }

        public event PMLNetDelegate.PMLNetEventHandler OnDateSelected;

        private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
        {
            if (OnDateSelected != null)
            {
                ArrayList args = new ArrayList();
                args.Add(this.StartDate);
                args.Add(this.EndDate);
                OnDateSelected(args);
            }
        }

    }
}
```

DatePickerControl.cs

```
// DATEPICKERCONTROL User Control
// ========================================================
// A datepicker control for use with on a PML form
// © AVEVA Solutions Ltd, www.aveva.com

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Collections;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    public partial class DatePickerControl : UserControl
    {
        [PMLNetCallable()]
        public DatePickerControl()
        {
            InitializeComponent();
        }

        [PMLNetCallable()]
        public void Assign(DatePickerControl that)
        {
            this.dateTimePicker1.Value = that.dateTimePicker1.Value;
        }

        [PMLNetCallable()]
        public void SetDate(string DateString)
        {
            try
            {
                this.dateTimePicker1.Value = DateTime.Parse(DateString);
            }
            catch
            {
                throw new PMLNetException(1000, 1,
                    "String cannot be converted into a date");
            }
        }

        public event PMLNetDelegate.PMLNetEventHandler OnDatePicked;

        private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
        {
            if (OnDatePicked != null)
            {
                ArrayList args = new ArrayList();
                args.Add(this.dateTimePicker1.Value.ToLongTimeString());
                args.Add(this.dateTimePicker1.Value.ToLongDateString());
                OnDatePicked(args);
            }
        }
    }
}
```

CheckListBoxControl.cs

```
// CHECKLISTBOXCONTROL User Control
// ==========================================================
// A checklistbox control for use with on a PML form
// © AVEVA Solutions Ltd, www.aveva.com

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using Aveva.PDMS.PMLNet;
using System.Collections;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    public partial class CheckListBoxControl : UserControl
    {
        [PMLNetCallable()]
        public CheckListBoxControl()
        {
            InitializeComponent();
        }

        [PMLNetCallable()]
        public void Assign(CheckListBoxControl that)
        {
        }

        [PMLNetCallable()]
        public void AddValue(string value)
        {
            ListViewItem LVItem = this.listView1.Items.Add(value);
        }

        [PMLNetCallable()]
        public void AddValue(string value, bool state)
        {
            ListViewItem LVItem = this.listView1.Items.Add(value);
            LVItem.Checked = state;
        }

        [PMLNetCallable()]
        public Hashtable GetSelectedItems()
        {
            Hashtable Data = new Hashtable();

            int i;
            for (i = 0; i < this.listView1.CheckedItems.Count; i++)
            {
                int index = Convert.ToInt16(this.listView1.CheckedIndices[i]) ;
                Data.Add(i, index);
            }
            return Data;
        }
        [PMLNetCallable()]
        public void Clear()
        {
            this.listView1.Items.Clear();
        }

        public void SetAllChecked(bool Flag)
        {
            foreach (ListViewItem item in this.listView1.Items)
            {
                item.Checked = Flag;
            }
        }

        public event PMLNetDelegate.PMLNetEventHandler OnChecked;

        private void listView1_ItemChecked(object sender, ItemCheckedEventArgs e)
        {
            if (OnChecked != null)
            {
                ArrayList args = new ArrayList();
                args.Add(e.Item.Index + 1);
                args.Add(e.Item.Checked);
                OnChecked(args);
            }
        }

        private void CheckListBoxControl_Load(object sender, EventArgs e)
        {
            this.Dock = DockStyle.Fill;
        }

        [PMLNetCallable()]
        public void SetCheckImages(string CheckedName, string UnCheckedName)
        {

            this.listView1.StateImageList = null;

            try
            {
                Bitmap CheckedImage = new Bitmap(CheckedName);
                Bitmap UnCheckedImage = new Bitmap(UnCheckedName);

                this.imageList1.Images.Clear();

                this.imageList1.Images.Add("UNCHECKED", UnCheckedImage);
                this.imageList1.Images.Add("CHECKED", CheckedImage);

                this.listView1.StateImageList = this.imageList1;

                CheckedImage.Dispose();
                UnCheckedImage.Dispose();
            }
            catch
            {
                throw new PMLNetException(1000, 1, "The supplied images could not be used");
            }
        }

        [PMLNetCallable()]
        public void StandardCheckBoxes()
        {
             this.listView1.StateImageList = null;
         }
    }
}
```

cSharpCheckListBoxControl.pmlfrm

```
setup form !!cSharpCheckListBoxControl dialog resiz

  import |Aveva.C.Sharp.Training|
  handle ANY
  endhandle
  using namespace |Aveva.C.Sharp.Training|

  container .listFrame NOBOX PMLNETCONTROL |list| anchor all width 30 height 10

  member .list is CHECKLISTBOXCONTROL

exit

define method .cSharpCheckListBoxControl()

  using namespace |Aveva.C.Sharp.Training|

  !this.list = object CHECKLISTBOXCONTROL()
  !this.listFrame.control = !this.list.handle()

  !this.list.addeventhandler(|OnChecked|, !this, |itemSelected|)

endmethod

define method .itemSelected(!data is ARRAY)

  q var !data[0] !data[1]

endmethod
```

Appendix A.3 – Source code for Chapter 5

PdfViewerCntrl.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.IO;

namespace Aveva.C.Sharp.Training
{
    public partial class PdfViewerCntrl : UserControl
    {
        public PdfViewerCntrl()
        {
            InitializeComponent();
        }

        public void OpenPDfFile()
        {
            Cursor.Current = Cursors.WaitCursor;

            if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
                this.webBrowser1.Url = new Uri(this.openFileDialog1.FileName);

            Cursor.Current = Cursors.Default;
        }

        public string GetPdfName()
        {
            return Path.GetFileName(this.openFileDialog1.FileName);
        }
    }
}
```

PdfViewerCmd.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerCmd : Command
    {
        // Store the private members of the command
        private DockedWindow mForm;
        private bool bSkip = false;
        private PdfViewerCntrl _pdfViewerCntrl;
        // Constructor method, supplied with an instance of the WINDOWMANAGER
        public PdfViewerCmd(WindowManager wndManager, PdfViewerCntrl pdfViewerCntrl)
        {
            // Update the KEY of the COMMAND (must be unique for all loaded commands
            base.Key = "Aveva.C.Sharp.Training.PdfViewerCmd";

            // Add an EVENT to look out for when the form is shown
            wndManager.WindowLayoutLoaded += new EventHandler(OnWindowLayoutLoaded);

            _pdfViewerCntrl = pdfViewerCntrl;

            // Create a DOCKED window in the WINDOWMANAGER
            mForm = wndManager.CreateDockedWindow("PdfViewerAddin.PdfViewerCmd",
                "PDF Viewer", _pdfViewerCntrl, DockedPosition.Right);

            mForm.SaveLayout = true;

            // Add an EVENT to look out for when the window is closed
            mForm.Closed += new EventHandler(OnFormClosed);
        }

        // WHen the control is loaded, ensure tyhe CHECKED property is TRUE
        void OnWindowLayoutLoaded(object sender, EventArgs e)
        {
            this.Checked = mForm.Visible;
        }

        // When the CONTROL is closed, ensure the CHECKED property is FALSE
        void OnFormClosed(object sender, EventArgs e)
        {
            // To override a problem with 12.0.SP5
            this.bSkip = true;
            this.Checked = false;
            this.bSkip = false;
        }

        // The CHECKED property is used by toggle gadgets on PDMS toolbars
        public override bool Checked
        {
            get { return mForm.Visible;}
            set { base.Checked = value;}
        }

        // When the COMMAND is executed, with hide or show the form
        public override void Execute()
        {
            // If BSKIP, then stop...
            if (bSkip)
                return;

            // If the FORM is visible, hide it and vice versa
            if (mForm.Visible)
                mForm.Hide();
            else
            {
                mForm.Show();
            }

            // Execute the standard method on the base class
            base.Execute();
        }
    }
}
```

PdfViewerOpenCmd.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerOpenCmd : Command
    {
        private PdfViewerCntrl _pdfViewerCntrl;

        public PdfViewerOpenCmd(CommandManager cmdManager, PdfViewerCntrl pdfViewerCntrl)
        {
            // Update the KEY of the COMMAND (must be unique for all loaded commands
            base.Key = "Aveva.C.Sharp.Training.PdfViewerOpenCmd";

            Command showCommand =
                cmdManager.Commands["Aveva.C.Sharp.Training.PdfViewerCmd"];
            showCommand.CommandExecuted += new EventHandler(showCommandExecuted);

            _pdfViewerCntrl = pdfViewerCntrl;
        }

        public void showCommandExecuted(object sender, EventArgs e)
        {
            this.Enabled = _pdfViewerCntrl.Visible;
        }

        public override bool Enabled
        {
            get { return _pdfViewerCntrl.Visible; }
            set { base.Enabled = value; }
        }

        public override void Execute()
        {
            if (_pdfViewerCntrl.Visible)
            {
                _pdfViewerCntrl.OpenPDfFile();
                this.Value = _pdfViewerCntrl.GetPdfName(); ;
                this.Refresh();
            }

            // Execute the standard method on the base class
            base.Execute();
        }
    }
}
```

PdfViewerAddin.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Aveva.ApplicationFramework;
using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerAddin : IAddin
    {
        private PdfViewerCntrl _pdfViewerCntrl ;

        // Get the description of the class
        public string Description
        {
            get { return "PDF Viewer C# Addin"; }
        }

        // Get the name of the class
        public string Name
        {
            get { return "PdfViewerAddin";}
        }

        // When the ADDIN is started, register the command with the commandmanager
        public void Start(ServiceManager serviceManager)
        {
            // Indicate that the ADDIN has been started in the console window
            Console.WriteLine("Example PDF Viewer C# Addin loaded");

            // Get instances of the COMMANDMANAGER and WINDOWMANAGER
            CommandManager sCommandManager =
                (CommandManager)serviceManager.GetService(typeof(CommandManager));
            WindowManager sWindowManager =
                (WindowManager)serviceManager.GetService(typeof(WindowManager));

            _pdfViewerCntrl = new PdfViewerCntrl();

            // Declare the PDFVIEWERCMD and register it with the COMMANDMANAGER
            sCommandManager.Commands.Add(new PdfViewerCmd(sWindowManager,
                _pdfViewerCntrl));
            sCommandManager.Commands.Add(new PdfViewerOpenCmd(sCommandManager,
                _pdfViewerCntrl));

            ResourceManager rManager =
                (ResourceManager)serviceManager.GetService(typeof(ResourceManager));
            rManager.LoadResourceFile("PdfViewer");

            // Load a UIC file for the Pdf Viewer.
            CommandBarManager commandBarManager =
                (CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));
            commandBarManager.AddUICustomizationFile("PdfViewer.uic", "Pdf Viewer");
        }

        // For completeness, implement the STOP method, but leave it blank
        public void Stop()
        {
        }
    }
}
```

CircularPanelCmd.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Aveva.ApplicationFramework.Presentation;

namespace Aveva.C.Sharp.Training
{
    class CreateCircularPanelCmd : Command
    {
        // Store the private members of the command
        private DockedWindow mForm;
        private bool bSkip = false;

        public CreateCircularPanelCmd(WindowManager wndManager)
        {
            // Update the KEY of the COMMAND (must be unique for all loaded commands
            base.Key = "Aveva.C.Sharp.Training.CreateCircularPanelCmd";

            // Add an EVENT to look out for when the form is shown
            wndManager.WindowLayoutLoaded += new EventHandler(OnWindowLayoutLoaded);

            // Create a DOCKED window in the WINDOWMANAGER
            mForm = wndManager.CreateDockedWindow("CircularPanelAddin.CircularPanelCmd",
                "Create Circular Panel", new CreateCircularPanelCntrl(),
                 DockedPosition.Right);

            mForm.SaveLayout = true;

            // Add an EVENT to look out for when the window is closed
            mForm.Closed += new EventHandler(OnFormClosed);
        }

        // WHen the control is loaded, ensure tyhe CHECKED property is TRUE
        void OnWindowLayoutLoaded(object sender, EventArgs e)
        {
            this.Checked = mForm.Visible;
        }

        // When the CONTROL is closed, ensure the CHECKED property is FALSE
        void OnFormClosed(object sender, EventArgs e)
        {
            // To override a problem with 12.0.SP5
            this.bSkip = true;
            this.Checked = false;
            this.bSkip = false;
        }

        // The CHECKED property is used by toggle gadgets on PDMS toolbars
        public override bool Checked
        {
            get { return mForm.Visible; }
            set { base.Checked = value; }
        }
        // When the COMMAND is executed, with hide or show the form
        public override void Execute()
        {
            // If BSKIP, then stop...
            if (bSkip)
                return;

            // If the FORM is visible, hide it and vice versa
            if (mForm.Visible)
                mForm.Hide();
            else
            {
                mForm.Show();
            }

            // Execute the standard method on the base class
            base.Execute();
        }


    }
}
```

PdfViewerAddin.cs (Exercise 5 modifications)

```
using Aveva.ApplicationFramework;
using Aveva.ApplicationFramework.Presentation;
using System;

namespace Aveva.C.Sharp.Training
{
    class PdfViewerAddin : IAddin
    {
        private PdfViewerCntrl _pdfViewerCntrl ;

        // Get the description of the class
        public string Description
        {
            get { return "Training C# Addin"; }
        }

        // Get the name of the class
        public string Name
        {
            get { return "TrainingAddin";}
        }

        // When the ADDIN is started, register the command with the commandmanager
        public void Start(ServiceManager serviceManager)
        {
            // Indicate that the ADDIN has been started in the console window
            Console.WriteLine("Training C# Addin loaded");

            // Get instances of the COMMANDMANAGER and WINDOWMANAGER
            CommandManager sCommandManager =
                (CommandManager)serviceManager.GetService(typeof(CommandManager));
            WindowManager sWindowManager =
                (WindowManager)serviceManager.GetService(typeof(WindowManager));

            _pdfViewerCntrl = new PdfViewerCntrl();

            // Declare the PDFVIEWERCMD and register it with the COMMANDMANAGER
            sCommandManager.Commands.Add(new PdfViewerCmd(sWindowManager,
                     _pdfViewerCntrl));
            sCommandManager.Commands.Add(new PdfViewerOpenCmd(sCommandManager,
                     _pdfViewerCntrl));
            sCommandManager.Commands.Add(new CreateCircularPanelCmd(sWindowManager));

            ResourceManager rManager =
                     (ResourceManager)serviceManager.GetService(typeof(ResourceManager));
            rManager.LoadResourceFile("PdfViewer");

            // Load a UIC file for the Pdf Viewer.
            CommandBarManager commandBarManager =
                    (CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));
            commandBarManager.AddUICustomizationFile("PdfViewer.uic", "Pdf Viewer");
        }

        // For completeness, implement the STOP method, but leave it blank
        public void Stop()
        {
        }

    }
}
```

Appendix A.4 – Source code for Chapter 6

CreateCircularPanel.cs

```
using System;
using Aveva.Pdms.Database;
using Aveva.Pdms.Geometry;
using Aveva.PDMS.PMLNet;
using Aveva.Pdms.Shared;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class CreateCircularPanel
    {
        private double _radius;
        private double _thickness;

        [PMLNetCallable()]
        public CreateCircularPanel()
        {
        }

        [PMLNetCallable()]
        public void Assign(CreateCircularPanel that)
        {
            _radius = that._radius;
            _thickness = that._thickness;
        }

        [PMLNetCallable()]
        public double Radius
        {
            get { return _radius; }
            set { _radius = value; }
        }

        [PMLNetCallable()]
        public double Thickness
        {
            get { return _thickness; }
            set { _thickness = value; }
        }

        [PMLNetCallable()]
        public bool CanBeCreated(string ElementName)
        {
            // Get the supplied name as a DbElement
            DbElement Element = DbElement.GetElement(ElementName);
            return Element.IsCreatable(DbElementTypeInstance.PANEL);
        }

        [PMLNetCallable()]
        public bool CreateBelowElement(string ElementName)
        {
            // Get the supplied name as a DbElement
            DbElement Element = DbElement.GetElement(ElementName);

            // Check the element is valid
            if (Element.IsNull || !Element.IsValid)
                return false;
            // Check a PANEL can be created
            if (!CanBeCreated(ElementName))
                return false;

            // Create the PANEL element
            DbElement panel = Element.Create(1, DbElementTypeInstance.PANEL);
            panel.SetAttribute(DbAttributeInstance.DESC, "CIRCULAR");

            // If the PANEL is not valid, stop...
            if (!panel.IsValid)
                return false;

            // Create the PLOO as the first member of the PANEL
            DbElement ploop = panel.Create(1, DbElementTypeInstance.PLOOP);

            // Set the thickness of the PLOO, based on the user value
            ploop.SetAttribute(DbAttributeInstance.HEIG, _thickness);

            // Create the 4 PAVEs required to define the shape of the PANEL (with FRAD set)
            DbElement Pave1 = ploop.Create(1, DbElementTypeInstance.PAVERT);
            Pave1.SetAttribute(DbAttributeInstance.POS, Position.Create(-_radius, -_radius, 0));
            Pave1.SetAttribute(DbAttributeInstance.FRAD, _radius);

            DbElement Pave2 = Pave1.CreateAfter(DbElementTypeInstance.PAVERT);
            Pave2.SetAttribute(DbAttributeInstance.POS, Position.Create(-_radius, _radius, 0));
            Pave2.SetAttribute(DbAttributeInstance.FRAD, _radius);

            DbElement Pave3 = Pave2.CreateAfter(DbElementTypeInstance.PAVERT);
            Pave3.SetAttribute(DbAttributeInstance.POS, Position.Create(_radius, _radius, 0));
            Pave3.SetAttribute(DbAttributeInstance.FRAD, _radius);

            DbElement Pave4 = Pave3.CreateAfter(DbElementTypeInstance.PAVERT);
            Pave4.SetAttribute(DbAttributeInstance.POS, Position.Create(_radius, -_radius, 0));
            Pave4.SetAttribute(DbAttributeInstance.FRAD, _radius);

            return true;
        }

        [PMLNetCallable()]
        public bool CreateBelowCe()
        {
            DbElement Element = CurrentElement.Element;
            bool result = CreateBelowElement(Element.GetString(DbAttributeInstance.NAME));
            return result;
        }

    }
}
```

CircularPanelCntrl.cs (Exercise 6 modifications)

```
using System;
using System.Windows.Forms;

using Aveva.Pdms.Database;
using Aveva.Pdms.Shared;

namespace Aveva.C.Sharp.Training
{
    public partial class CreateCircularPanelCntrl : UserControl
    {
        public CreateCircularPanelCntrl()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Use the CreateCircularPanel class from the worked example
            CreateCircularPanel createPanel = new CreateCircularPanel();

            // Get the current element
            DbElement Ce = CurrentElement.Element;

            // Check if a panel can be created below the current element
            if (!createPanel.CanBeCreated(Ce.GetString(DbAttributeInstance.NAME)))
                return;

            // Update the size information from the form
            createPanel.Radius = Convert.ToDouble(numRadius.Value);
            createPanel.Thickness = Convert.ToDouble(numThick.Value);

            // Create the panel
            createPanel.CreateBelowCe();

        }
    }
}
```

Appendix A.5 – Source code for Chapter 7

CollectCircularPanel.cs

```
using System;
using System.Collections;
using Aveva.Pdms.Database;
using Aveva.PDMS.Database.Filters;
using Aveva.Pdms.Shared;
using Aveva.PDMS.PMLNet;

namespace Aveva.C.Sharp.Training
{
    [PMLNetCallable()]
    class CollectCircularPanel
    {

        [PMLNetCallable()]
        public CollectCircularPanel()
        {
        }

        [PMLNetCallable()]
        public void Assign(CollectCircularPanel that)
        {
        }

        [PMLNetCallable()]
        public Hashtable Collect()
        {
            DbElement Element = CurrentElement.Element;
            TypeFilter PanelFilter = new TypeFilter(DbElementTypeInstance.PANEL);
            AndFilter CircularPanelFilter =
                new AndFilter(PanelFilter, new CircularPanelFilter());
            DBElementCollection Collect = new DBElementCollection(Element,
                CircularPanelFilter);

            Hashtable Collected = new Hashtable();
            int i;
            foreach (DbElement Panel in Collect)
            {
                i++;
                Collected.Add(i, Panel.GetString(DbAttributeInstance.NAME);
            }

            return Collected;
        }

    }
}
```

CircularPanelFilter.cs

```
using System;
using Aveva.Pdms.Database;
using Aveva.PDMS.Database.Filters;

namespace Aveva.C.Sharp.Training
{
    public class CircularPanelFilter : BaseFilter
    {
        public override object Clone()
        {
            return new CircularPanelFilter();
        }

        public override bool ScanBelow(DbElement Element)
        {
            return true;
        }

        public override bool Valid(DbElement Element)
        {
            string description = Element.GetString(DbAttributeInstance.DESC);

            if (description == "CIRCULAR")
                return true;
            else
                return false;

        }
    }
}
```

CircularPanelCntrl.cs (Exercise 7 modifications)

```
        private void button2_Click(object sender, EventArgs e)
        {
            // Use the CollectCircularPanel class from the worked example
            CollectCircularPanel CollectObj = new CollectCircularPanel();

            // Collect the panels of the required size
            Hashtable Elements = CollectObj.Collect(Convert.ToDouble(txtRadius.Text));

            // Clear the list
            lstCollectedPanels.Items.Clear();

            // Loop through the elements and add them to the list
            foreach (DictionaryEntry Entry in Elements)
            {
                DbElement Element = DbElement.GetElement(Entry.Value.ToString());
                ListViewItem LVItem = new
                     ListViewItem(Element.GetString(DbAttributeInstance.NAME));
                LVItem.Tag = Element;
                lstCollectedPanels.Items.Add(LVItem);
            }
        }

        private void lstCollectedPanels_SelectedIndexChanged(object sender, EventArgs e)
        {
            ListViewItem selected = this.lstCollectedPanels.SelectedItems[0];
            CurrentElement.Element = (DbElement)selected.Tag;
        }
```

CircularPanelFilter.cs (Exercise 7 modifications)

```
using System;
using Aveva.Pdms.Database;
using Aveva.PDMS.Database.Filters;

namespace Aveva.C.Sharp.Training
{
    public class CircularPanelFilter : BaseFilter
    {
        private double _Radius = 0.0;

        public CircularPanelFilter(double Radius)
        {
            _Radius = Radius;
        }

        public override object Clone()
        {
            return new CircularPanelFilter(_Radius);
        }

        public override bool ScanBelow(DbElement Element)
        {
            return true;
        }

        public override bool Valid(DbElement Element)
        {
            // Get the PLOO and the PAVE elements
            DbElement PLOO = Element.FirstMember();
            DbElement[] PAVEs = PLOO.Members();

            // Check there are four PAVE elements
            if (PAVEs.Length != 4)
                return false;

            // Loop through each PAVE and check the radius
            foreach (DbElement PAVE in PAVEs)
            {
                if (PAVE.GetDouble(DbAttributeInstance.FRAD) != _Radius)
                    return false;
            }

            // If we've reached here, return "true"
            return true;
        }
    }
}
```

Appendix A.6 – Source code for Chapter 8

Form1.cs

```
using System;
using System.Windows.Forms;

using Aveva.Pdms.Database;
using Aveva.Pdms.Standalone;
using Aveva.Pdms.Utilities;

namespace MDBOverview
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            try
            {
                // Try starting the PdmsStandalone
                if (!PdmsStandalone.Start())
                {
                    PdmsStandalone.ExitError("Failed to start");
                }
                // Try opening the SAM project using the PDMSStandalone
                if (!PdmsStandalone.Open("SAM", "SYSTEM", "XXXXXX", "SAMPLE"))
                {
                    PdmsStandalone.ExitError("Failed to login");
                }
            }
            catch
            {
                PdmsStandalone.ExitError("Failed to start");
            }

            // Get the DB array from the current MDB
            Db[] dbs = MDB.CurrentMDB.GetDBArray();

            // Loop through the databases and compile the information
            foreach (Db d in dbs)
            {
                ListViewItem Litem = this.listView1.Items.Add(d.Name);
                Litem.SubItems.Add(d.Number.ToString());
                Litem.SubItems.Add(d.Type.ToString());
                Litem.SubItems.Add(d.WorldMembers().Length.ToString());
                Litem.SubItems.Add(d.LatestSessionOnDb.User);
                Litem.SubItems.Add(d.LatestSessionOnDb.Date.ToLongDateString());
            }
        }
    }
}
```

Form1.cs (Exercise 8 modifications)

```
using System;
using System.Windows.Forms;
using Aveva.Pdms.Database;
using Aveva.Pdms.Standalone;
using Aveva.Pdms.Utilities;

namespace MDBOverview
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                if (!PdmsStandalone.Start())
                    PdmsStandalone.ExitError("Failed to start");

                // Try opening the project using the PDMSStandalone
                if (!PdmsStandalone.Open(txtProj.Text, txtUser.Text, txtPass.Text,
                                         txtMDB.Text))
                    PdmsStandalone.ExitError("Failed to login");

            }
            catch
            {
                PdmsStandalone.ExitError("Failed to start");
            }

            Db[] dbs = MDB.CurrentMDB.GetDBArray();
            foreach (Db d in dbs)
            {
                ListViewItem Litem = this.listView1.Items.Add(d.Name);
                Litem.SubItems.Add(d.Number.ToString());
                Litem.SubItems.Add(d.Type.ToString());
                Litem.SubItems.Add(d.WorldMembers().Length.ToString());
                Litem.SubItems.Add(d.LatestSessionOnDb.User);
                Litem.SubItems.Add(d.LatestSessionOnDb.Date.ToLongDateString());

                if (db.World.IsAttributeSetable(DbAttributeInstance.LOCK))
                {
                    Litem.ImageIndex = 0;
                    Litem.BackColor = Color.LightGreen;
                }
                else
                {
                    Litem.ImageIndex = 1;
                    Litem.BackColor = Color.LightPink;
                }

                if (db.DbItem.GetBool(DbAttributeInstance.ISDBFR))
                    Litem.BackColor = Color.Silver;

            }
        }
    }
}
```

APPENDIX B

Appendix B – Additional information

Appendix B.1 - Database Expressions

Database expressions are PML1 expressions. E.g. (XLEN * 1000). Expressions are of the following type:

- Double
- DbElement
- Bool
- String
- Position
- Direction
- Orientation

There is a **DbExpression** class to hold an expression. DbExpression provides static method **Parse(…)** to create an instance of DbExpression based on a given expression string:

```
DbExpression dbexp = DbExpression.Parse("( SUBSTR( NAMN OF SITE, 1,3 ))");
```

Once an expression has been created it can be evaluated against a DbElement instance. The DbElement instance provides a set of **Evaluate** methods depending on the expression's result type:

```
elem.EvaluateBool(dbexp);
elem.EvaluateString(dbexp);
elem.EvaluateElement(dbexp);
```

If the expression cannot be evaluated an exception is raised.

Appendix B.2 - MDB/DB Operations

MDB and DB operations are provided by **DatabaseService, MDB**, **Project** and **Db** objects implemented in **Aveva.Pdms.Database** assembly in namespace with the same name. They provide a number of administration API's.

If you want to access the current opened project or opened MDB you can simply use static properties that return the current instance.

Example – printing to console current MDB name:

```
Console.WriteLine(MDB.CurrentMDB.Name);
```

If you want to start working with databases the project has to be open. To open a project use the **OpenProject(...)** static method from the DatabaseService class.

Example – opening SAM project:

```
Project proj = DatabaseService.OpenProject("SAM", "SYSTEM", "/XXXXXX");
```

OpenProject(...) takes arguments:
1. Project name
2. User name
3. User password (starting with "/")

Returns null if a project could not be opened.

ⓘ *Note that before exiting the application Project should be closed* ***proj.Close()***

After a project is accessed you can open an available MDB. Use **OpenMDB(...)** static method from Project class.

Example – opening /SAMPLE MDB:

```
MDB mdb = Project.OpenMDB(MDBSetup.CreateMDBSetup("/SAMPLE"));
```

OpenMDB(...) takes the **MDBSetup** argument that stores options for opening the MDB. MDBSetup can be created passing the name of the MDB as a string starting "/" to static **CreateMDBSteput(...)** method.

ⓘ *Note that before exiting the application MDB should be closed **mdb.Close()***

Often the user wants to work with all project data stored in many databases. To access all DbElements you might want to access all the databases in a given MDB. This can be done using the **GetDBArray()** method.

Example – accessing all databases:

```
Db[] databases = mdb.GetDBArray();
```

To access a particular DB use the **GetDB(...)** method that uses the DB number as parameter:

To see other user's changes in multiwrite databases or save work the following methods on the MDB class can be used:

1. **GetWork(...)** – gets saved changes since last got work or MDB open
2. **SaveWork(...)** - saves all changes
3. **QuitWork(...)** – drops all changes
4. **Refresh()** – refreshes all changes (leaves user changes but doesn't save them)

Example – saving work:

```
if(mdb.SaveWork("Training changes"))
       MessageBox.Show("Work has been saved successfully!");
```

SaveWork(...) method takes string parameter to indicate action history.

Appendix B.3 - Invoking PDMS Commands

It may be necessary in some instances to invoke PML commands from .NET using the **Command** class. This is implemented in **Aveva.Pdms.Utilities** and stored under **Aveva.Pdms.Utilities.CommandLine** namespace.

The Command class can be instantiated with its **CreateCommand(…)** static method. Having created an instance it can be executed with its **RunInPdms()** method.

```
if (!Command.CreateCommand("FINISH").RunInPdms())
      return false;
```

Appendix B.4 - Getting Information about Current Project and Session

To get information about the current session, you need to get a reference to the session object, which is stored in the **SystemDB** in the project properties:

```
Db sysDb = Project.CurrentProject.SystemDB;
DbSession dbSession = sysDb.CurrentSession;
```

Now you can refer to the object's properties to get the desired information

```
dbSession.Date
dbSession.Description
dbSession.SessionNumber
dbSession.User
```

To retrieve the project information, this is a bit more complicated, when you want to get properties like Name, User or Type from the project, you'll only need to call **CurrentProject**'s properties.

```
Project.CurrentProject.Name;
Project.CurrentProject.Type;
Project.CurrentProject.User;
```

The remaining properties like project number, message and description must be retrieved from the Database manually.
First you will have to get a reference to the STATUS element which can be found in the DB.

```
Db sys = Project.CurrentProject.SystemDB;
DbElement world = sys.World;
DbElement stat = world.FirstMember(DbElementTypeInstance.STATUS);
```

Now that you have a Status element, you can get it's attributes from the Database.

Example – getting project properties:

```
//Project Description
stat.GetString(DbAttributeInstance.PRJD);
//Project Message
stat.GetAsString(DbAttributeInstance.INFB);
//Project Number
stat.GetString(DbAttributeInstance.PRJN);
```

Appendix B.5 - Deploying an Addin on Network Location

The .NET platform prevents .NET addins running if deployed on a network. This will not usually cause an issue for AVEVA products, for which AVEVA recommends a local installation on each machine but might cause problems for customers running their own add-ins.

.NET security can cause issues when running AVEVA products across the network where the add-in assemblies reside on a different machine to the .NET runtime. The default security level is not set to **Full Trust**, which means that programs may not be able to access resources on the network machine. To overcome this, the intranet security may be set to Full Trust, though this means that any .NET assembly may run. Alternatively, Full Trust may be given to a specified group of strongly named assemblies.

Full Trust is configured using the Code Access Security Policy tool (**Caspol**). First of all the assemblies must be strongly named. Then Caspol is run on each client machine to add all the assemblies on a given server directory to a group and give Full Trust to this group as follows:

To trust all assemblies in a given folder:

The **Caspol** need to be used from .NET 2.0, using the one from .NET 1.1 does not work

```
caspol -m -ag LocalIntranet_Zone -url
    \\<ServerName>\<FolderName>\* FullTrust -n "<Name>" -d "<Description>"
```

OR to trust all assemblies with the same strong name:

```
caspol -m -ag LocalIntranet_Zone -strong –file
    \\<ServerName>\<FolderName>\<assemblyName> -noname -noversion FullTrust -n
    "Aveva" -d "Full trust for Aveva products"
```

where <ServerName> is the UNC (Uniform Naming Convention).

The format of a UNC path is: \\<servername>\<sharename>\<directory>

where:

<servername>	The network name,
<sharename>	The name of the share,
<directory>	Any additional directories below the shared directory.

Caspol can be found in c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\ or is part of the .NET Framework 2.0 SDK.

Appendix B.6 – Pseudo UDAs

If a UDA is declared as "Pseudo", it can have .NET code associated with it. This code can be used to return the value whenever the UDA is queried.

To register a method against a UDA, the following would be required:

1) Get an instance of UDA

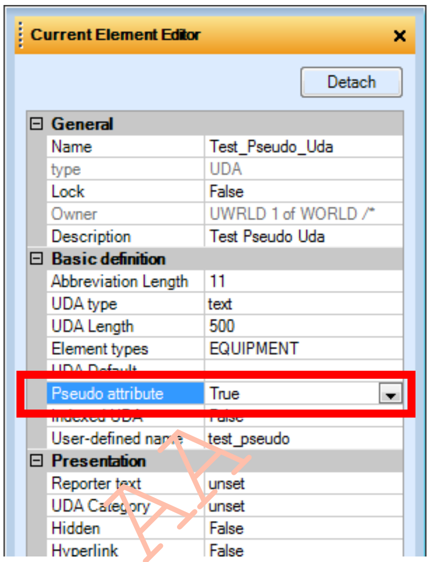
```
DbAttribute uda = DbAttribute.GetDbAttribute(":test_pseudo");
```

2) Define which method should be called

```
using Ps = Aveva.Pdms.Database.DbPseudoAttribute;
Ps.GetStringDelegate dele = new Ps.GetStringDelegate(Method);
```

3) Link the two together

```
Ps.AddGetStringAttribute(uda, dele);
```



General	
Name	Test_Pseudo_Uda
type	UDA
Lock	False
Owner	UWRLD 1 of WORLD /*
Description	Test Pseudo Uda
Basic definition	
Abbreviation Length	11
UDA type	text
UDA Length	500
Element types	EQUIPMENT
UDA Default	
Pseudo attribute	True
Indexed UDA	False
User-defined name	test_pseudo
Presentation	
Reporter text	unset
UDA Category	unset
Hidden	False
Hyperlink	False

This assumes there is a method called "Method" available, that has the correct signature.

For example, to return the first three characters of the element name (uppercase):

```
static private string Method(DbElement ele, DbAttribute att, DbQualifier qualifier)
{
        string ElementName = ele.GetString(DbAttributeInstance.NAMN);
        return ElementName.Substring(0,3).ToUpper();
}
```

It is common for this code to run when PDMS is started (i.e. using a .NET Addin).

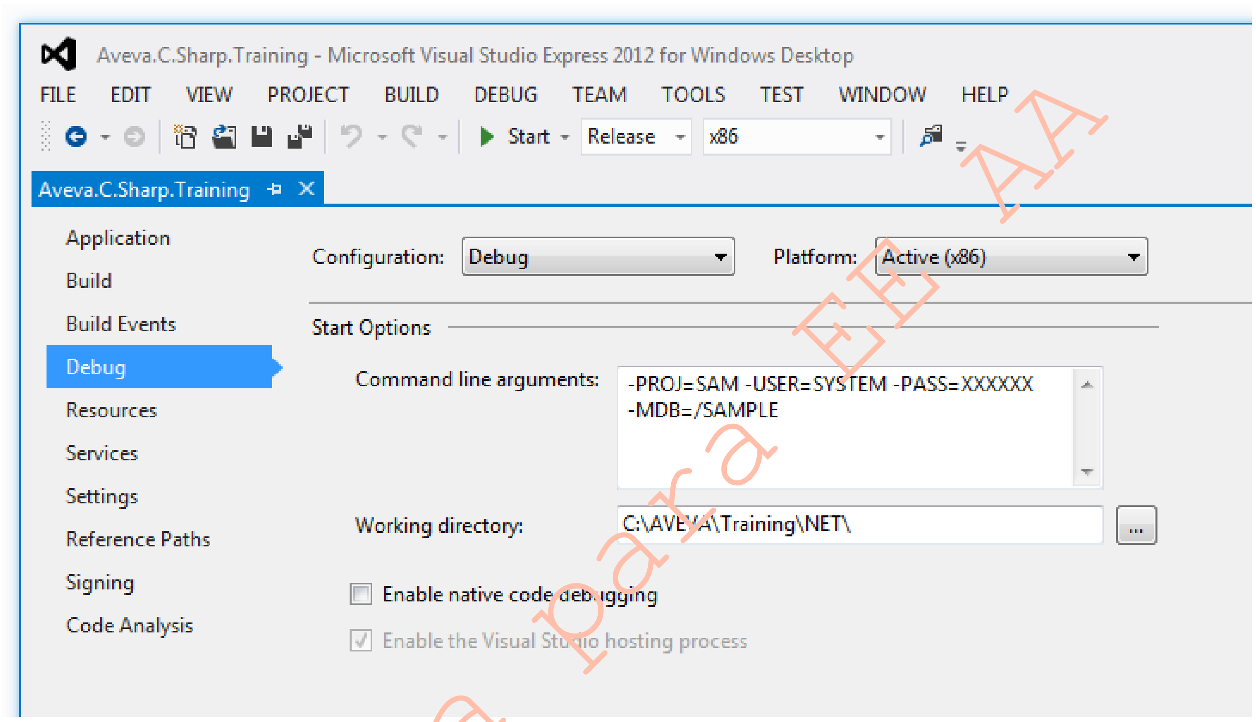
Appendix B.7 – Configuring Visual Studio to start and debug PDMS

Visual Studio has the ability to start an application while debugging. To start PDMS, the following is required:

1) For Visual Studio to set up the correct environment for PDMS to run.
2) For Visual Studio to run the correct PDMS executable with the correct arguments.

The file **Training_vs.bat** has been supplied to setup the correct environment. This is a modified version of Training.bat that sets the correct environment, but starts Visual Studio.

Within a project, under "Properties" the "Debug" tab can be updated to the following:



Configuration: Debug
Platform: Active (x86)

Start Options

Command line arguments: -PROJ=SAM -USER=SYSTEM -PASS=XXXXXX -MDB=/SAMPLE

Working directory: C:\AVEVA\Training\NET\

☐ Enable native code debugging

☑ Enable the Visual Studio hosting process

Start options would need to be set for each configuration (e.g. Debug, Release, X86 etc). The result is that when "Start Debugging" is selected PDMS is lauched and any errors are trapped by Visual Studio.

Depending on the version of Visual Studio, you may also be able to set the exectable at this point.

If not, this information is stored in the projects **.csproj file** (in this case Aveva.C.Sharp.Training.csproj). An additional tag can be added to set the start program. For example:

```
<PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Release|x86'">
  <OutputPath>bin\x86\Release\</OutputPath>
  <DefineConstants>TRACE</DefineConstants>
  <Optimize>true</Optimize>
  <DebugType>pdbonly</DebugType>
  <PlatformTarget>x86</PlatformTarget>
  <ErrorReport>prompt</ErrorReport>
  <CodeAnalysisRuleSet>ManagedMinimumRules.ruleset</CodeAnalysisRuleSet>
  <StartAction>Program</StartAction>
  <StartProgram>C:\AVEVA\Plant\PDMS12.1.SP4\des.exe</StartProgram>
  <StartArguments>-PROJ=SAM -USER=SYSTEM -PASS=XXXXXX -MDB/SAMPLE</StartArguments>
  <StartWorkingDirectory>C:\AVEVA\Training\Net</StartWorkingDirectory>
</PropertyGroup>
```