

# .NET Customisation User Guide

## PML Disclaimer

1.1 AVEVA does not warrant that the use of the AVEVA software will be uninterrupted, error-free or free from viruses.

1.2 AVEVA shall not be liable for: loss of profits; loss of business; depletion of goodwill and/or similar losses; loss of anticipated savings; loss of goods; loss of contract; loss of use; loss or corruption of data or information; any special, indirect, consequential or pure economic loss, costs, damages, charges or expenses which may be suffered by the user, including any loss suffered by the user resulting from the inaccuracy or invalidity of any data created by the AVEVA software, irrespective of whether such losses are suffered directly or indirectly, or arise in contract, tort (including negligence) or otherwise.

1.3 AVEVA shall have no liability in contract, tort (including negligence), or otherwise, arising in connection with the performance of the AVEVA software where the faulty performance of the AVEVA software results from a user's modification of the AVEVA software. User's rights to modify the AVEVA software are strictly limited to those set out in the Customisation Manual.

1.4 AVEVA shall not be liable for any breach or infringement of a third party's intellectual property rights where such breach results from a user's modification of the AVEVA software or associated documentation.

1.5 AVEVA's total liability in contract, tort (including negligence), or otherwise, arising in connection with the performance of the AVEVA software shall be limited to 100% of the licence fees paid in the year in which the user's claim is brought.

1.6 Clauses 1.1 to 1.5 shall apply to the fullest extent permissible at law.

1.7. In the event of any conflict between the above clauses and the analogous clauses in the software licence under which the AVEVA software was purchased, the clauses in the software licence shall take precedence.

## PML Copyright

Copyright and all other intellectual property rights in this manual and the associated software, and every part of it (including source code, object code, any data contained in it, the manual and any other documentation supplied with it) belongs to, or is validly licensed by, AVEVA Solutions Limited or its subsidiaries.

All rights are reserved to AVEVA Solutions Limited and its subsidiaries. The information contained in this document is commercially sensitive, and shall not be copied, reproduced, stored in a retrieval system, or transmitted without the prior written permission of AVEVA Solutions Limited. Where such permission is granted, it expressly requires that this copyright notice, and the above disclaimer, is prominently displayed at the beginning of every copy that is made.

The manual and associated documentation may not be adapted, reproduced, or copied, in any material or electronic form, without the prior written permission of AVEVA Solutions Limited. Subject to the user's rights, as set out in the customisation manuals to amend PML software files contained in the PDMSUI and PMLLIB folders and any configuration files, the user may not reverse engineer, decompile, copy, or adapt the software. Neither the whole, nor part of the software described in this publication may be incorporated into any third-party software, product, machine, or system without the prior written permission of AVEVA Solutions Limited, save as permitted by law. Any such unauthorised action is strictly prohibited, and may give rise to civil liabilities and criminal prosecution.

The AVEVA software described in this guide is to be installed and operated strictly in accordance with the terms and conditions of the respective software licences, and in accordance with the relevant User Documentation. Unauthorised or unlicensed use of the software is strictly prohibited.

Copyright 1974 to current year. AVEVA Solutions Limited and its subsidiaries. All rights reserved. AVEVA shall not be liable for any breach or infringement of a third party's intellectual property rights where such breach results from a user's modification of the AVEVA software or associated documentation.

AVEVA Solutions Limited, High Cross, Madingley Road, Cambridge, CB3 0HB, United Kingdom.

## PML Trademark

AVEVA and Tribon are registered trademarks of AVEVA Solutions Limited or its subsidiaries. Unauthorised use of the AVEVA or Tribon trademarks is strictly forbidden.

AVEVA product/software names are trademarks or registered trademarks of AVEVA Solutions Limited or its subsidiaries, registered in the UK, Europe and other countries (worldwide).

The copyright, trademark rights, or other intellectual property rights in any other product or software, its name or logo belongs to its respective owner.

[illegible]



# .NET Customisation User Guide

---

Contents	Page
 <b>User Guide</b>	
<b>Introduction</b> . . . . .	<b>1:1</b>
<b>.NET Customisation Architecture</b> . . . . .	<b>1:1</b>
Common Application Framework Interfaces . . . . .	1:1
Database Interfaces . . . . .	1:2
Geometry Interfaces . . . . .	1:3
Shared Interfaces . . . . .	1:3
Utilities Interfaces . . . . .	1:4
Graphics Interfaces . . . . .	1:4
<b>Sample Code</b> . . . . .	<b>1:4</b>
AttributeBrowserAddin . . . . .	1:4
ExamplesAddin . . . . .	1:4
NetGridExample . . . . .	1:4
PMLNetExample . . . . .	1:4
PMLGridExample . . . . .	1:5
<b>Reference Documentation</b> . . . . .	<b>1:5</b>
<b>How to Write a CAF Addin</b> . . . . .	<b>2:1</b>
<b>The IAddin Interface</b> . . . . .	<b>2:3</b>
Addin Start-up Performance . . . . .	2:4
<b>The WindowManager</b> . . . . .	<b>2:4</b>
Window Creation . . . . .	2:4
<b>Addin Commands</b> . . . . .	<b>2:7</b>
Writing a Command Class . . . . .	2:8
Command Events . . . . .	2:10

---

<b>Resource Manager</b> .....	<b>2:11</b>
<b>Configuring a Module to Load an Addin</b> .....	<b>2:11</b>
<b>Communicating through Events and Delegate Call-backs</b> .....	<b>2:12</b>
<b>Tracing and Optimisation</b> .....	<b>2:13</b>
<b>Exception Handling in a CAF Addin</b> .....	<b>2:15</b>
<b>Thread safety in a CAF Addin</b> .....	<b>2:16</b>
<b>Menu and Command Bar Customisation</b> .....	<b>3:1</b>
<b>Configuring a Module to Load a UIC File</b> .....	<b>3:1</b>
<b>Editing the UIC File</b> .....	<b>3:2</b>
Selection of Active Customisation File .....	3:3
The Tree .....	3:4
List of Command Tools .....	3:6
Property Grid .....	3:10
Action Buttons .....	3:13
Resource Editor .....	3:14
<b>Database Interface</b> .....	<b>4:1</b>
<b>Data Model Definition Classes</b> .....	<b>4:1</b>
DbElementType .....	4:1
DbAttribute .....	4:2
DbElementTypeInstance .....	4:4
DbAttributeInstance .....	4:4
<b>Element Access</b> .....	<b>4:5</b>
DbElement Basics .....	4:5
Navigation .....	4:7
Getting Attribute Values .....	4:9
Database Modifications .....	4:11
Storage of Rules and Expressions .....	4:16
Comparison of Data with Earlier Sessions .....	4:17
<b>Filters/Iterators</b> .....	<b>4:18</b>
Iterators .....	4:18
Filters .....	4:19
<b>Dabacon Tables</b> .....	<b>4:19</b>
Overview of Dabacon Tables .....	4:19
Table Classes .....	4:20
<b>DBs, MDBs and Projects</b> .....	<b>4:21</b>
MDB Functionality .....	4:21

DB Functionality .....	4:21
<b>Events .....</b>	<b>4:23</b>
Overview of Events .....	4:23
Overview of C# Mechanism .....	4:23
General Capture of DB Changes .....	4:23
Adding Pseudo Attribute Code .....	4:24
DB/MDB Related Events .....	4:25
<b>Units .....</b>	<b>4:26</b>
DbDoubleUnits .....	4:26
DbUnits .....	4:27
DbDoubleDimension .....	4:27
DbDimension .....	4:27
DbDouble .....	4:27
Exception Handling .....	4:28
DbFormat .....	4:28
<b>PMLNet .....</b>	<b>5:1</b>
<b>Design Details .....</b>	<b>5:1</b>
Using PMLNet .....	5:1
.NET Controls .....	5:9
Examples .....	5:10
<b>AVEVA C# Grid Control .....</b>	<b>6:1</b>
<b>Create a C# Addin which Contains an AVEVA Grid Control .....</b>	<b>6:1</b>
<b>Provide Access to the Addin in Design .....</b>	<b>6:3</b>
<b>Use the AVEVA Grid Control with Different Data Sources .....</b>	<b>6:4</b>
<b>Add an XML Menu to the Form .....</b>	<b>6:5</b>
<b>Add an Event to the Addin .....</b>	<b>6:6</b>
<b>Other Functionality Available within the Environment .....</b>	<b>6:9</b>
<b>Use of the C# Grid Control with PML .....</b>	<b>6:10</b>
<b>AVEVA Grid Control API .....</b>	<b>6:10</b>
<b>Input Mask Characters .....</b>	<b>6:16</b>
<b>AVEVA My Data .....</b>	<b>7:1</b>
<b>My Data API .....</b>	<b>7:1</b>
<b>Using My Data from PML .....</b>	<b>7:2</b>
Add an Element .....	7:2
Add an Array of Elements .....	7:2

Remove an Element. ....	7:3
Remove Everything (Elements and Collections) ....	7:3
Remove an Array of Elements ....	7:3
Add a Collection. ....	7:3
Add an Array of Collections ....	7:3
Remove a Collection ....	7:3
Remove an Array of Collections. ....	7:3
Add an Element to a Collection ....	7:3
Add an Array of Elements to a Collection. ....	7:3
Remove an Element from a Collection. ....	7:4
Remove all Elements ....	7:4
Remove all Elements from Collection ....	7:4
Remove all Collections. ....	7:4
Remove an Array of Elements from a Collection ....	7:4
Rename a Collection ....	7:4
Query Elements and Collections ....	7:4
Query Elements in a given Collection ....	7:4
<b>PML File Browser .....</b>	<b>8:1</b>



# 1 Introduction

The .NET Customisation User Guide gives software engineers with experience of software development in C# using Visual Studio guidance on the development of .NET customisation for the AVEVA Plant and AVEVA Marine products.

## 1.1 .NET Customisation Architecture

The introduction of a customisation capability using Microsoft .NET technology has opened up a whole new world of customisation and is particularly relevant for the integration of AVEVA products with other customer systems. .NET API's provided access to various aspects of the product including Graphical User Interface, Database and Geometry.

As part of AVEVA's strategy of 'continual progression' the .NET customisation capability has been introduced in such a way that it can be used alongside the existing PML based customisation. Through the use of PML.NET, an enhancement to PML which allows the PML programmer to call .NET code, customisation which utilizes the strengths of .NET compiled code and PML can be achieved.

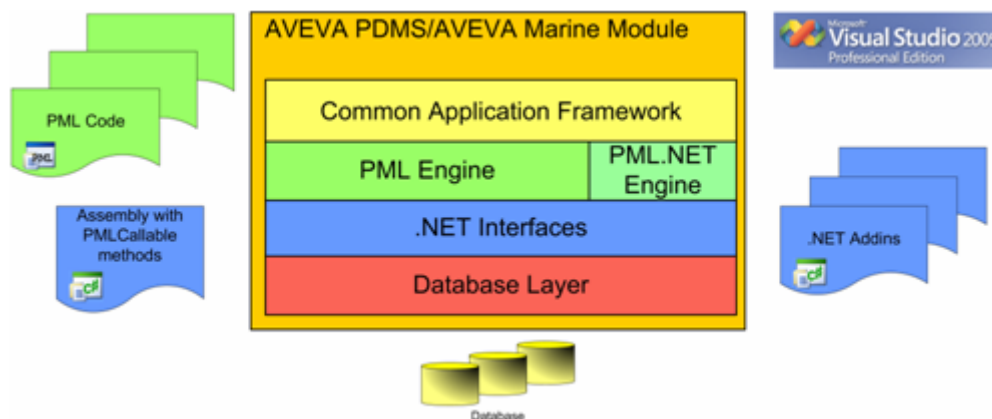


Figure 1:1. .NET customisation Architecture

The above diagram illustrates the two new methods of customisation using .NET technology. The first is via the concept of a .NET Addin and the second using PML.NET. Both methods provide a mechanism whereby a .NET assembly (dll) can be dynamically loaded into a module at runtime.

### 1.1.1 Common Application Framework Interfaces

The Common Application Framework (CAF) is an extensible framework which provides the .NET programmer with access to various services which support both application

development and customisation. The foundations of the CAF are provided by the two interface assemblies:

- Aveva.ApplicationFramework.dll
- Aveva.ApplicationFramework.Presentation.dll

These interfaces provide the following major services:

## Namespace: Aveva.ApplicationFramework

- **AddinManager:** This class provides properties and methods for the management of ApplicationFramework Addins.
- **ServiceManager:** This class defines an interface which provides a means by which the various components of the ApplicationFramework can publish their services. The ServiceManager also acts as a service provider responding to requests for services. It can also be used by applications and application addins to publish additional services.
- **SettingsManager:** This class provides properties and methods for the management of application settings which are stored in settings groups which are persisted between sessions.

## Namespace: Aveva.ApplicationFramework.Presentation

- **CommandBarManager:** This provides access to the menus and commandbars of a CAF based application. It also has methods to load the definition of menus and commandbars from User Interface customisation (UIC) files.
- **CommandManager:** This class defines an interface to provide the presentation framework client with a mechanism for the management of command objects which can be associated with Tools or other User interface objects. The action of invoking a tool (e.g clicking a ButtonTool) will cause the execution of the associated command object. It is possible to associated the same command object with a number of different user interface objects (e.g. ButtonTool on a Menu and a LinkLabel) thereby allowing for the centralisation of these user interface objects action within a command. Various state-like properties of a command (e.g. enabled/checked) would also be reflected in all user interface objects associated with a command. For example, disabling a command would cause all associated user interface objects to be disabled. User interface objects are associated with a command via a CommandExecutor derived class.
- **ResourceManager:** This class defines an interface to provide Addins with a simplified mechanism to access localizable resources. The ResourceManager provides a number of methods which allows an addin to then access the various types of resources (string, image, cursor, icon etc.) which resource files may contain.
- **WindowManager:** This provides access to the main application window, the StatusBar and a collection of MDI and docked windows. It also provides the addin writer with methods to create MDI and docked windows to host user controls.

## 1.1.2 Database Interfaces

The database related interfaces are provided by the interface assemblies:

- Aveva.Pdms.Database.dll & PDMSFilters.dll

This interface has the following main classes:

## Namespace: Aveva.Pdms.Database

- **DatabaseService:** The sole purpose of this class is to open a project.
- **DbAttribute:** The class has two purposes:
  1. Instances of the class are used to identify and pass around attributes

2. The methods allow the retrieval of metadata about an attribute. e.g. type, size, Name, pseudo or not, etc. The class is valid for system attributes and UDAs.
- **DB:** This class provides information on the opened DBs.
  - **DbElement:** Instances of DbElement are used to identify an element. This is the main class for reading and writing to the database. The methods cover
    - element creation
    - element deletion
    - copy
    - getting/setting attributes and rules
    - navigating the DB
    - evaluating database expressions.
  - **DbElementType:** The class has two purposes:
    1. Instances of the class are used to identify and pass around Element types
    2. The methods allow the retrieval of metadata about an Element type. e.g. Name, description, allowed attributes and members etc.

The class is valid for system types and UDETs.

- **DbEvents:** This class contains the mechanisms for subscribing to database events. It covers savework, getwork, undo, redo, flush, refresh, drop events plus capturing general database changes.
- **DbExpression:** Class to hold a database expression. These are the same as PML1 expressions. Methods to evaluate expressions are on the DbElement class.
- **DbPseudoAttribute:** This Class allows pseudo attribute code to be plugged in for UDAs.
- **DbRule:** Class to hold a database rule
- **DbUserChanges:** This is passed through by the 'database change' event. It contains a list of database changes made since the last 'database change' event was raised. Changes may be queried down to the attribute level.
- **MDB:** Methods to do operations on an MDB. e.g. savework, getwork.
- **Table:** Various classes to access the internal Dabacon tables. E.g. the name table.
- **Project:** The main method is to open the MDB.

As well as the class methods there is a lot of functionality that can be accessed via pseudo attributes. The relevant pseudo attributes are listed where relevant.

## 1.1.3 Geometry Interfaces

The geometry related interfaces are provided by the interface assembly:

- AVEVA.Pdms.Geometry.dll

This interface has a number of geometry related classes. See reference documentation.

## 1.1.4 Shared Interfaces

Some general shared interfaces are provided in the interface assembly:

- Aveva.Pdms.Shared.dll

This provides current element, selection changed events and Data listing facilities.

## 1.1.5 Utilities Interfaces

Utility interfaces are provided in the interface assembly:

- Aveva.Pdms.Utilities.dll

This provides messaging, string utilities, tracing, undo and units. It also provides access to the command line but this is not supported

## 1.1.6 Graphics Interfaces

Interfaces to access the drawlist and colours are provided in the interface assembly:

- Aveva.Pdms.Graphics.dll

The rest of this user guide will cover in detail how to write a .NET addin and an assembly containing classes with methods which are callable from PML. It will describe the use of the various .NET interfaces via the use of sample code.

## 1.2 Sample Code

A zip file (Samples.zip) containing the samples can be found in the installation directory. For the projects assembly references to be valid the zip file should be unzipped into the installation directory. It will create a Samples directory containing the sub-directories described below. If they are moved elsewhere then the references to assemblies in the installation directory will need to be updated.

Samples are provided which are compatible with Visual Studio 2005, and Visual Studio 2005 Express.

### 1.2.1 AttributeBrowserAddin

A zip file (Samples.zip) containing the samples can be found in the installation directory. This example project creates an addin which implements a very simple database element attribute browser. It also illustrates the implementation of a command object to control the visibility of the docked window created by the addin.

### 1.2.2 ExamplesAddin

A zip file (Samples.zip) containing the samples can be found in the installation directory. .NET API Examples can be found in the ExamplesAddin project below the Samples directory. This is a C# addin which adds a ComboBoxTool on a Toolbar to the main menu. Each entry in the list runs a particular example. This can be loaded by adding the ExamplesAddin to the applications addin config file. Since these examples create and claim Design elements they need to be run in a Design multi-write database. There is an example config file in the ExamplesAddin directory.

### 1.2.3 NetGridExample

A zip file (Samples.zip) containing the samples can be found in the installation directory. An example of a .NET C# addin containing an AVEVA Grid Control.

### 1.2.4 PMLNetExample

A zip file (Samples.zip) containing the samples can be found in the installation directory. An example of a .NET class that has been made PML callable.

### **1.2.5 PMLGridExample**

A zip file (Samples.zip) containing the samples can be found in the installation directory. An example of an AVEVA Grid Control hosted on a PML form.

## **1.3 Reference Documentation**

Reference documentation for the various .NET interfaces is provided in the form of help files which can be found in the Documentation directory.



## 2 How to Write a CAF Addin

### Architectural Considerations for Addins

Most addins embody at least four separate concepts:

- Specific behaviour as a **CAF Addin**
- Zero or more **graphical user interface controls and dialogues**
- The code specific to this **application**
- The code to **manage the graphical user interaction** with the controls and application.

As in all coding situations there is much benefit to be gained by keeping these concepts separate and the code relating to each one self-contained, or "encapsulated" in object oriented jargon. This keeps the code for each simple, and offers maximum flexibility.

For example GUI controls can often be shared with other addins and applications. The NetGridControl is a good example of a control that can be reused in many situations. When this is the case the control code should be placed in a separate assembly (dll) from the addin and application. At the very least the control code should be in a separate class (file) from the addin code. Whether shared or not the principles of encapsulation advise having no application data specific code in the control.

Similarly if the application specific code needs to be active when the GUI itself is not displayed, for example in teletype (TTY) mode or for batch runs, then it, too, should be in a separate assembly and care will be needed to ensure it is correctly initialised independently of the addin itself.

Keeping the application code, the GUI control code and the GUI management code separate is good practice that leads to clearer and more logical code in all areas - much easier to support and maintain in the long term. Inevitably the addin, the application, the management and the controls need to communicate during operation. There is a natural hierarchical relationship between them. The following diagram shows the principal lines of communication between the components of the Addin and between the Addin and the AVEVA host program. The circles denote the interfaces that each component exposes - its own API (application programmable interface) and other standard interfaces. The arrows illustrate the natural lines of communication between each component and the interfaces of the other components.

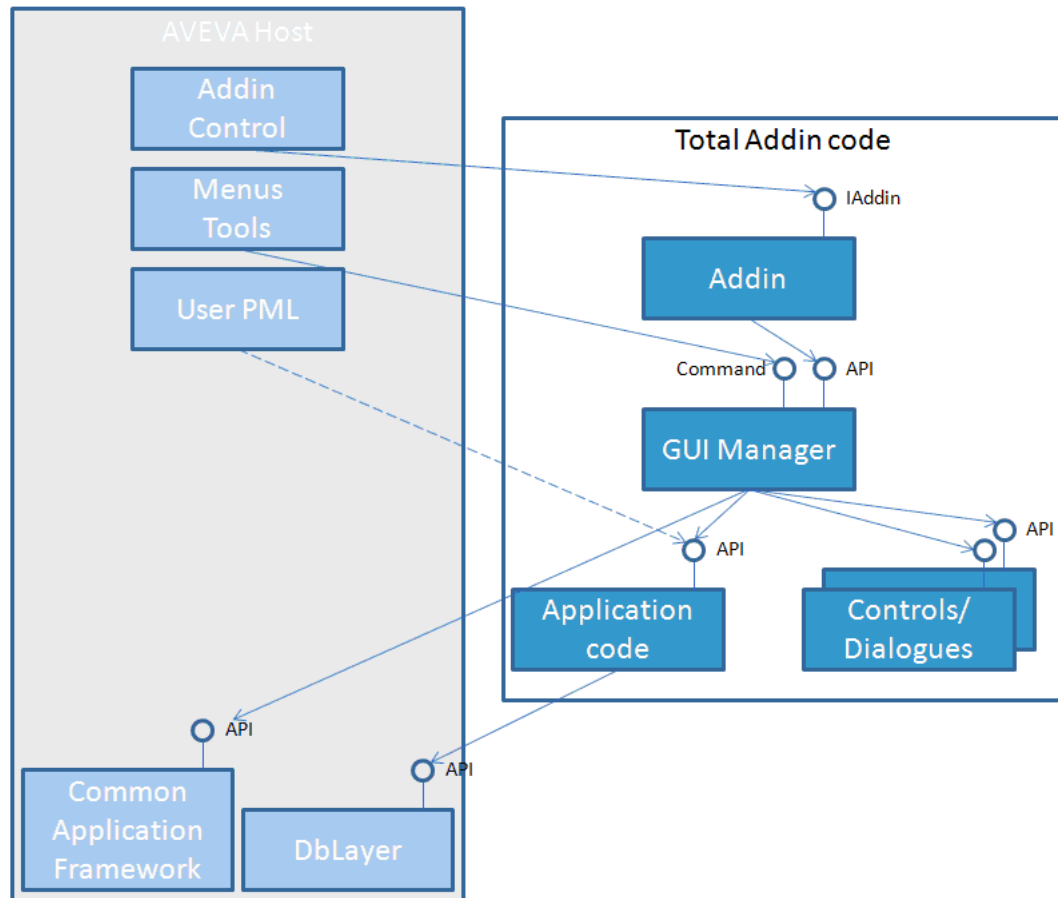


Figure 2:1. Writing an Addin

Occasionally there is a need for communication between these components that goes against the natural hierarchy - either upwards or across to other components on the same level. One important mechanism for doing this while maintaining logical encapsulation is by registering delegate functions to be executed on specific events. The definition of these events is part of the API exposed by the component notifying the event. The delegate (or call-back) function is part of the component that must respond to the event. This is discussed further later.

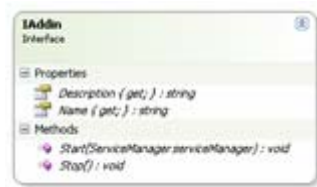
As another example of good encapsulation, Addins themselves should be self contained and dependencies between them should be avoided.

The specific responsibilities of the **Addin** code are as follows:

- To respond to the PDMS or Marine host module to identify itself and to start and stop the addin - this is done by implementing the CAF IAddin interface functions.
- To build and register the required menus and toolbars - once only normally.
- To create any docking windows required and registers them with the CAF WindowManager - once only. Determine whether the docked window settings are to be saved between runs of the PDMS/Marine module.
- To manage the initialisation of any GUI controls and dialogues required and add them to the docked window if appropriate.
- To initialise any associated application code (but not necessarily the application data - see below).



## 2.1 The IAddin Interface



What distinguishes an addin from any other .NET assembly is that an addin must contain a class which implements the IAddin interface. When the CAF loads and starts an addin it interrogates the assembly for the class which implements this interface, creates an instance of this class and calls the IAddin.Start method.

The following code is a simple example of a class which implements the IAddin interface.

This is the AttributeBrowserAddin.cs file which is part of the AttributeBrowserAddin sample project which can be found in the samples directory. The Start method performs the initialization of the Addin.

```
using System;
using System.Collections.Generic;
using System.Text;
// Add additional using statements
using Aveva.ApplicationFramework;
using Aveva.ApplicationFramework.Presentation;

namespace Aveva.Presentation.AttributeBrowserAddin
{
    public class AttributeBrowserAddin : IAddin
    {
        #region IAddin Members

        public string Description
        {
            get
            {
                return "Provides a simple attribute browser";
            }
        }

        public string Name
        {
            get
            {
                return "AttributeBrowserAddin";
            }
        }

        public void Start(ServiceManager serviceManager)
        {
            // Create Addins Windows
            // Get the WindowManager service
            WindowManager windowManager = (WindowManager)serviceManager.Get-
```

```
Service(typeof(WindowManager));
    // Create a docked window to host an AttributeListControl
    DockedWindow attributeListWindow = windowManager.CreateDocked-
Window("Aveva.AttributeBrowser.AttributeList", "Attributes", new
AttributeListControl(), DockedPosition.Right);
    // Docked windows created at addin start should ensure their lay-
out is saved between sessions.
    attributeListWindow.SaveLayout = true;

    // Create and register addins commands
    // Get the CommandManager
    CommandManager commandManager = (CommandManager)serviceMan-
ager.GetService(typeof(CommandManager));
    ShowAttributeBrowserCommand showCommand = new ShowAttribute-
BrowserCommand(attributeListWindow);
    commandManager.Commands.Add(showCommand);
}

public void Stop()
{
}

#endregion
}
```

Figure 2.2. AttributeBrowserAddin Sample - AttributeBrowserAddin.cs

### 2.1.1 Addin Start-up Performance

One of the benefits of keeping the application code separate from the code of associated controls, as recommended above, is that it makes it much easier to make separate decisions about when to undertake action especially, but not only, at start-up. It is normal that all windows, controls and dialogues are built and initialised when the addin starts up - normally when the PDMS or Marine module itself starts up - but it is not necessary to populate it with data unless and until the window is shown. Unnecessary calculations of this kind can add appreciably to the time taken to start up the PDMS/Marine module giving a poor user experience. Normally a slight delay at the point where the window in question is first shown goes almost unnoticed by the user and is therefore much more acceptable.

The principle of delaying calculations until the point where the result is required is known as "lazy evaluation". Lazy evaluation is useful not only at start up, but can also be used, for example, to limit the extent of calculation done. The NetGridControl, for example, exploits this to calculate fully only the part of the table which is visible at any one time, and thereby greatly improving responsiveness for large data sets.

## 2.2 The WindowManager

### 2.2.1 Window Creation

An addin will typically expose its functionality to the user through the use of a graphical user interface. This can take the form of forms which are shown from menu or commandbar buttons or the addin writer may wish the addins graphical user interface to be hosted inside

a dockable window or in an MDI child window. The facilities provided by the CAF WindowManager can be used to create these two types of window.

```
// Create Addins Windows
// Get the WindowManager service
WindowManager windowManager =
(WindowManager)serviceManager.GetService(typeof(WindowManager));
// Create a docked window to host an AttributeListControl
DockedWindow attributeListWindow =
windowManager.CreateDockedWindow("Aveva.AttributeBrowser.AttributeList",
"Attributes", new AttributeListControl(), DockedPosition.Right);
// Docked windows created at addin start should ensure their layout
is saved between sessions.
attributeListWindow.SaveLayout = true;
```

Figure 2:3. Using WindowManager to create a Docked Window

The above code shows the use of the ServiceManager.GetService method to retrieve the WindowManager service and the use of the CreateDockedWindow method to create a docked window to host the AttributeListControl. The first argument, Key, to the CreateDockedWindow method needs to be a unique window identifier. To help avoid clashes between docked windows created by addins running within the same application then it is recommended to adopt the <CompanyName>.<AddinName>.<WindowName> naming convention for this property.

An MDI window can be created via use of the WindowManager.CreateMdiWindow method.

As illustrated in this example, an important step when creating docked windows is the correct setting of the SaveLayout property. This property controls whether information about the layout and docking position is saved between application sessions. The code defines the default docking position as DockedPosition.Right, but the user can interactively change this and the persistence of this user preference between sessions is desirable. One aspect of the saving (serialisation) and restoring (de-serialisation) of this layout data is that this only works if only those docked windows for which there is layout information are present when the layout data is de-serialized. Therefore it is important that any docked window created during addin startup has this property set to true. If there is a mismatch between the number of docked windows in existence when the layout data is de-serialized then you will get one of the following warning message then the application starts up:

```
Warning: Loading Window Layout - Missing DockedWindow:
Aveva.AttributeBrowserWarning: Failed to restore docked
window layout
```

```
Warning: Loading Window Layout - Extra DockedWindow:
Aveva.AttributeBrowserWarning: Failed to restore docked
window layout
```

You should only see these warnings when either adding or removing an addin from a module.

### IWindow Interface

The DockedWindow and MdiWindow both implement the IWindow interface which has the following methods and properties:

**void Hide()** - Conceals the window from the user.

**void Show()** - Displays the window to the user.

**void Float()** - Displays the window as a floating window.

**void Dock()** - Docks the window within the main window.

**void Close()** - Destroys the window removing it from the windows collection.

**System.Windows.Forms.Control Control** - Gets the control displayed in the window.

**bool Enabled** - Gets or sets whether the window is enabled.

**bool Floatable** - Gets or sets whether the window is floatable.

**int Height** - Gets or sets the height of the window.

**bool IsFloating** - Gets the floating state of a window.

**string Key** - Gets the Key of the window in the WindowsCollection.

**string Title** - Gets or sets the title/caption of the window.

**bool Visible** - Gets or sets the visible state of the window.

**int Width** - Gets or sets the width of the window.

**Size MaximumSize** - Get or sets the maximum size of the window.

**Size MinimumSize** - Get or sets the minimum size of the window.

### Window Events

The Docked and MDI Windows also support a number of events such as Closed, Activated, Deactivated, Resized.

### WindowManager Events

The window manager also supports two events:

**event System.EventHandler WindowLayoutLoaded** - Occurs at startup just after the window layout has been loaded. This event can be used to update the state of commands which are being used to manage the visibility of a docked window. (see the ShowAttributeBrowserCommand.cs in the AttributeBrowserAddin sample)

**event WindowEventHandler WindowAdded** - Occurs when a new docked or MDI window is created.

### The StatusBar

The CAF also provides the addin writer with an interface to the StatusBar located at the bottom of the main application window.



The StatusBar is accessed from the StatusBar property of the WindowManager.

```
StatusBar statusBar = windowManager.StatusBar;
```

It has the following properties which can be used to control the StatusBar and its contents:

**bool Visible** - Gets or sets the visibility of the StatusBar.

**string Text** - Gets or sets the text to display in the default StatusBar text pane.

**int Progress** - Gets or sets the progress bar value [0-100]. If this is set to 0 then the progress bar is hidden.

**string ProgressText** - Text to describe the action being tracked by the progress bar.

**bool ShowDateTime** - Gets or sets whether the Date and Time should be displayed on the StatusBar.

**bool ShowCapsLock** - Gets or sets whether the panel showing the CapsLock state is displayed on the StatusBar.

**bool ShowNumLock** - Gets or sets whether the panel showing the NumLock state is displayed on the StatusBar.

**bool ShowScrollLock** - Gets or sets whether the panel showing the ScrollLock state is displayed on the StatusBar.

**bool ShowInsertMode** - Gets or sets whether the panel showing the InsertMode is displayed on the statusbar.

**StatusBarPanelsCollection Panels** - Gets the collection of application defined StatusBar panels.

The StatusBar allows the user to create additional panels using the Panels collection property. The StatusBarPanelsCollection currently has methods to create StatusBarTextPanel objects and add them to the StatusBar.

```
// Add a new panel to contain the project name.
StatusBar statusBar = windowManager.StatusBar;
StatusBarTextPanel projectNamePanel =
statusBar.Panels.AddTextPanel("Aveva.ProjectName", "Project : " +
Project.CurrentProject.Name);
projectNamePanel.SizingMode = PanelSizingModeAutomatic;
// Get the panel image from the addins resource file.
projectNamePanel.Image = resourceManager.GetImage("ID_PROJECT_ICON");
```

Figure 2.4. : Code to create a StatusBar Panel containing the project name.

The StatusBarTextPanel object has a number of properties which control its content and behaviour. It also supports PanelClick and PanelDoubleClick events. For details of these please refer to the reference help file.

## 2.3 Addin Commands

User access to an Addin's functionality is typically accessed via main menus, context menus and tools on a commandbar or grouped on a command ribbon. In common with most Windows applications, the CAF provides an interactive interface via the CommandBarManager for the creation and customisation of menus, commandbars and ribbon groups and the various types of tool that they can contain. The CAF offers a mechanism for publishing the capabilities of the Addin to the CommandBarManager for this purpose, namely the Command object. The Command is a C# class that provides this interface for the CommandBarManager, and also provides a standard communication channel between the user interface and the Addin for the execution of the command and the feedback of state such as "enabled", "visible" and "checked". This is therefore a good mechanism for keeping the requirements of the user interface separate from the definition of the Addin itself. The code of the Command objects provides part of the formal interface of the "GUI Manager" component of the Addin as described in [The IAddin Interface](#).

For historical reasons the CAF also support a traditional event style interface where the programmer provides event handlers to respond to the various events generated as a result

of user interaction with the user interface. However this traditional approach to user interface development although still supported by the CAF is not recommended. There are known issues with this older approach, not least being an incompatibility with newer user interface styles such as command ribbons. It should also be noted here that old style tool bars are themselves not compatible with command ribbons. Thus any Addin that needs to be deployed with older style menus/command bars as well as newer style command ribbons should avoid the use of free standing tool bars.

When the user interacts with the CommandBarManager to build or customise menus, command bars or ribbon groups and their tools, the result is stored in a "User Interface Customization" (UIC) file. This is a XML file containing the definitions of the menus, commandbars and their contents including the values of their various properties. Details of how to create these UIC files will be covered later in this manual. The UIC file provides the link between the user interface and the Command objects for the Addin. The UIC definition is loaded at start-up and the links to the Addin Commands are then created using the services of the Command Manager. Selection of the menu entry or clicking on the button will then cause the associated command to be executed.

One of the advantages of a command based model is that it forces the decoupling of the user interface or presentation layer from the application logic. The application logic has no direct reference to the presentation layer entities. If application state needs to be reflected in the user interface then the application modifies the state of the command. The command knows which user interface entities are associated with it and takes the necessary steps to ensure that its internal state is reflected in the user interface. This is easily achieved for state such as "enabled", "visible" or "checked", but becomes complex when dynamic application state needs to be reflected in user interface entities such as a combo-box.

### 2.3.1 Writing a Command Class

The example below is for a simple command class use to manage the visibility state of a docked window, in this case the AttributeBrowser docked window. The CAF provides an abstract base class from which every Command class must inherit. The constructor for a Command class should set the base class Key property which is used to reference the command from within a UIC file.

The Command base class has the following methods and properties which can be overridden by a derived Command class.

**void Execute()** : This method must be overridden to provide the command execution functionality.

Optionally you can specify a collection of arguments to the execute method:

**void Execute(ArrayList arguments)** : This will automatically use the correct overload of Execute() which matches the argument types given in the ArrayList. If no match is found then nothing is called. You can use the support property **Executes** to obtain a list of valid Execute() overloads and the method **FindExecute(ArrayList arguments, out MethodInfo info)** to determine if an argument list matches an overload.

**CommandState GetState()** : This method is called by the CAF to update the state of the contents of a context menu. The returned value is a CommandState enumeration for various states of a command. This enumeration can be treated as a bit field; that is, a set of flags. Bit fields can be combined using a bitwise OR operation. The command state is then reflected by the user interface.

**String Description** : A description for the command.

**void Refresh(string context)** : This method will be called whenever the `CommandManager.ApplicationContext` property is changed. This gives the command the opportunity to update its Enabled or Visible state in response to this context change. This command state change would then be reflected in the user interface.

The Command base class also has a number of properties which are use to update the command state following user interface changes or vice-versa.

**bool Checked:** If associated with a user interface entity such as a `StateButtonTool` then this property and the corresponding state of the user interface entity are kept synchronised.

**bool Enabled:** Changes to this property are reflected in all associated user interface entities.

**ArrayList List:** This property allows a command to communicate a list of string values to the user interface. This can be used when a command is associated with for example a `ComboBoxTool`.

**int SelectedIndex:** This property is updated to indicate which item from a list has been selected by the user.

**object Value:** This property holds the currently value of an associated user interface entity.

**bool ValueChanged:** Before calling the execute method the CAF sets this property if the value of the user interface entity has changed. The flag is cleared when execution has finished.

**bool Visible:** Changes to this property are reflected in all associated user interface entities.

Registering a command with the CAF is done by adding an instance of a command class to the `CommandManagers.Commands` collection.

```
ShowAttributeBrowserCommand showCommand = new
ShowAttributeBrowserCommand(attributeListWindow);
commandManager.Commands.Add(showCommand);
```

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

namespace Aveva.Presentation.AttributeBrowserAddin
{
    /// <summary>
    /// Class to manage the visibility state of the AttributeBrowser docked
    window
    /// This command should be associated with a StateButtonTool.
    /// </summary>
    public class ShowAttributeBrowserCommand : Command
    {
        private DockedWindow _window;
        /// <summary>
        /// Constructor for ShowAttributeBrowserCommand
        /// </summary>
        /// <param name="window">The docked window whose visibility state
        will be managed.</param>
        public ShowAttributeBrowserCommand(DockedWindow window)
        {
```



```

        // Set the command key
        this.Key = "Aveva.ShowAttributeBrowserCommand";
        // Save the docked window
        _window = window;
        // Create an event handler for the window closed event
        _window.Closed += new EventHandler(_window_Closed);
        // Create an event handler for the WindowLayoutLoaded event
        WindowManager.Instance.WindowLayoutLoaded += new
EventHandler(Instance_WindowLayoutLoaded);
    }
    void Instance_WindowLayoutLoaded(object sender, EventArgs e)
    {
        // Update the command state to match initial window visibility
        this.Checked = _window.Visible;
    }
    void _window_Closed(object sender, EventArgs e)
    {
        // Update the command state when the window is closed
        this.Checked = false;
    }
    /// <summary>
    /// Override the base class Execute method to show and hide the win-
    dow
    /// </summary>
    public override void Execute()
    {
        if (this.Checked)
        {
            _window.Show();
        }
        else
        {
            _window.Hide();
        }
    }
}

```

Figure 2.5. AttributeBrowserAddin Sample - ShowAttributeBrowserCommand.cs

## 2.3.2 Command Events

The Command base class provides two events `BeforeCommandExecute` and `CommandExecuted` which enable the addin writer to write code to respond to the execution of any command object registered with the `CommandManager`. A command object can be retrieved from the `CommandManager` if its `Key` is known and event handlers can be attached to these events.

```

Command anotherCommand =
commandManager.Commands[ "KeyOfCommand" ];
anotherCommand.BeforeCommandExecute += new
System.ComponentModel.CancelEventHandler(anotherCommand_BeforeCommandExecute);
anotherCommand.CommandExecuted += new

```



```
EventHandler(anotherCommand_CommandExecuted);
The BeforeCommandExecute event handler is of type
CancelEventHandler and is passed a CancelEventArgs object
which enables the command execution to be cancelled by setting
the Cancel property to true.
void anotherCommand_BeforeCommandExecute(object sender,
System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = true;
}
```

## 2.4 Resource Manager

Another CAF service is the ResourceManager which provides addins with a simplified mechanism to access localizable resources. The ResourceManager supports the loading of ResourceSets which consist of a number of locale specific resource files with a given "basename".

```
resourceManager.LoadResourceFile("AttributeBrowserAddin");
```

This will load all resource files with a basename of "AttributeBrowserAddin". E.g.

```
AttributeBrowserAddin.resources (Invariant language locale)
```

```
AttributeBrowserAddin.de.resources (German locale)
```

```
AttributeBrowserAddin.ko-KR.resources (Korean-Korea locale)
```

Depending on the machine's regional settings resource lookups will firstly try to locate the resource from the corresponding locale specific resource file. If this is not found it will defer to the language invariant resource file.

The ResourceManager provides a number of methods which allow an addin to access the various types of resources (string, image, cursor, icon etc.) which these resource files may contain. Each of the resource retrieval methods takes as an argument a resource identifier string and an optional resourceset name. If the resourceset name is omitted then it will default to a resourceset with the same name as the addin retrieving the resource.

## 2.5 Configuring a Module to Load an Addin

Having written an addin the next step is to get it to be loaded by a module. Each CAF based application has an XML addin configuration file which contains a list of the addins that the application should load at startup. The default location for this file is in the same directory as the application executable. It has a filename of the form <Module Name>Addins.xml. For example Design has a file called DesignAddins.xml. The content of this file is reproduced below. By default the addins are also expected to be in the same directory as the application executable. You can however specify the full path to the addin assembly including the use of UNC pathnames. If during addin development you do not wish to modify the addin configuration file in the install directory (this is particularly likely to be the case if you are using a server based installation) then the default location of the addin configuration file can be overridden using the environment variable CAF\_ADDINS\_PATH. You can then edit a copy of the file and point this environment variable at the directory you copy it to.

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<string>ExplorerAddin</string>
<string>DrawListAddin</string>
<string>MyDataAddin</string>
<string>HistoryAddin</string>
<string>ReferenceListAddin</string>
<string>PipeCheckAddin</string>
<string>OutputAddin</string>
<string>FindAddin</string>
<string>AttributesAddin</string>
<string>C:\Documents and Settings\User1\My Documents\Visual Studio
2005\AttributeBrowserAddin\AttributeBrowserAddin\bin\Debug\Attribute-
BrowserAddin</string>
</ArrayOfString>
```

## 2.6 Communicating through Events and Delegate Call-backs

The classic way to communicate between GUI controls (or the host module) and the code of applications while maintaining the object oriented principle of encapsulation is for the application to register call-backs functions as delegates to be called when certain events occur in the control or the module. In this way the specific code to deal with the application data resides in the application itself leaving the control or module independent of the application.

The modules and standard GUI controls of PDMS or Marine have a number of events built-in as standard. These are discussed elsewhere in this document:

- [Events](#) discusses use of the built-in database events.
- [Add an Event to the Addin](#) describes an example of using the NetGridControl and its standard events.
- [Command Events](#) describes the standard events associated with a .NET Command object.

Users who build their own controls can also define their own system of events for communication with their application code. This is a standard part of the C# language, but the syntax is not straight-forward. The following example illustrates the use of delegates and events.

### Code in the control:

```
/// <summary>Event Handler definition for My Control</summary>
public delegate void MyControlEventHandler(ArrayList args);
/// <summary>After Data Filter is changed</summary>
public event MyControlEventHandler DataFilterChanged;
/// <summary>Another event ...</summary>
public event MyControlEventHandler AnotherEvent;

// ...

// Raise DataFilterChanged event
if (DataFilterChanged != null)
{
    ArrayList args = new ArrayList();
```

```
// set args as required ...
DataFilterChanged(args);
}
```

**And in the application code:**

```
// register a call-back function to handle Data Filter Changed
events
myControl.DataFilterChanged +=
    new MyControlEventHandler (myControl_DataFilterChanged);

// ...

/// <summary>Handle the consequences of a changed data filter</
summary>
private void myControl_DataFilterChanged(ArrayList args)
{
    // ...
}
```

And of course it is entirely possible for some or all of the application code to be implemented in PML. For this purpose the CAF provides an event and call-back mechanism to communicate between C# and PML applications thus enabling effective encapsulation of behaviour in the code whichever combination of languages is chosen. Mixed language C#/PML working is covered in [PMLNet](#).

## 2.7 Tracing and Optimisation

Suggestions to minimise the speed of start-up of addins have been made above. There are 2 built-in ways to monitor this and investigate the reasons if poor performance is suspected.

Reviewing Addin Start-up Time

Target start-up time for an Addin should be no more than 1 second, if possible. To find out how long your Addin takes to start-up set the following environment variable:

```
set AVEVA_CAF_TRACING=PERFORMANCE
```

Output similar to the following will then be displayed in the Console Window:

```
AVEVA PDMS Design Mk12.1.SP1.0[3430] (WINDOWS-NT 5.1) (5 Nov 2010
: 18:49)
```

```
(c) Copyright 1974 to 2010 AVEVA Solutions Limited
```

```
Issued to APDian.steele-ukcamsplsNO-TE-St
```

```
Loading addins took 0.793 seconds
Starting addin Explorer took 1.108 seconds
Starting addin DrawList took 0.135 seconds
Starting addin MyData took 0.810 seconds
Starting addin History took 0.110 seconds
Starting addin ReferenceList took 0.060 seconds
Starting addin PipeCheck took 0.683 seconds
Starting addin Output took 1.580 seconds
Starting addin Find took 0.343 seconds
Starting addin Links Addin took 0.126 seconds
```

```
Starting addin Attributes took 1.015 seconds
Starting addin Status Controller took 0.128 seconds
Total time starting addins 6.098 seconds
Cleared existing tools in 0.009 seconds.
Loading UIC file C:\AVEVA\PDMSEXE\design.uic
Defined new tools in 0.039 seconds.
Loading UIC file C:\AVEVA\PDMSEXE\StatusController.uic
Defined new tools in 0.009 seconds.
Loading UIC file C:\AVEVA\PDMSEXE\CoreSchematicMenu.uic
Defined new tools in 0.008 seconds.
Loading UIC file C:\AVEVA\PDMSEXE\BAS.uic
Defined new tools in 0.000 seconds.
Loading UIC file C:\AVEVA\PDMSEXE\UserDesign.uic
Defined new tools in 0.000 seconds.
Loading XML GUI took 0.068 seconds
Processing UILoaded event tool 0.057 seconds
Loading CommandBar layout took 0.801 seconds
DockManager.LoadFromXML in 1.118 seconds.
Loading Window Layout took 1.119 seconds
```

### Standard PDMS Tracing

The standard facilities found in the classes of the Aveva.Pdms.Utilities.Tracing namespace can be used for tracing performance problems as well as for other general coding investigations.

The **PdmsTrace** class gives access to all the facilities of the PDMS trace system and provides a means of outputting general trace messages and values during processing. PdmsTrace functions also enable querying and setting of PDMS trace flags; give access to the current state of the call stack; and allow starting and stopping of CPU time tracing.

The **TraceEntry** class provides the means to trace the entry and exit of functions as processing takes place. As the nesting of function calls gets deeper and deeper the function entry messages are indented further and further. Though this output can quickly get verbose it can frequently give evidence of problems in the code - for example functions being called too frequently ... or not at all!

To work correctly in C#, TraceEntry needs to know about entry and exit from functions in correct chronological order. There is a difficulty with this in C# because of the way that "garbage collection" works in that language. This means, in particular, it is necessary to ensure that the function **exit** event occurs at the right moment. This can be achieved by using the Dispose method explicitly:

```
TraceEntry tr = TraceEntry.Enter("Start", (TraceNumber)101);
// ...
// code being traced ...
// ...
tr.Dispose();    // explicit call on exit
```

There are still problems with this however as you will need to catch every "return" statement individually. Also the Dispose() method will never be called if an exception occurs. A simpler and more robust way is to do this job using the C# "using" command as follows:

```
using (TraceEntry tr = TraceEntry.Enter("Start", (TraceNumber)101))
```

```
{
    // ...
    // code being traced ...
    // ...
} // implicit call to Dispose() on exit from the block
// even if there are exceptions or multiple return statements
```

If you are using these facilities to investigate the start-up of an addin, you will need to set the chosen trace flag from your code:

```
PdmsTrace.SetTraceFlag((TraceNumber)101, 1);
```

And, finally, you can use the PDMS CPU time profiling options in conjunction with TraceEntry by using:

```
PdmsTrace.StartCPUTrace();
// ...
// code being timed ...
// ...
PdmsTrace.StopCPUTrace(true);
```

## 2.8 Exception Handling in a CAF Addin

The code of all components of a CAF Addin is executed within the event loop of the host AVEVA program. Any exception raised within the Addin code, or any code called by it, that is not caught by the addin code itself will have to be handled in the event loop. The AVEVA host has effective ways to handle some classes of exceptions - namely PdmsExceptions and PMLNetExceptions. In the situation where the host program is in the process of executing PML, then the PML first has an opportunity to handle the exception itself. If the PML does not handle the exception, or if no PML is currently being executed, the host program handles these exceptions by notifying the user of the exception text and number interactively via MessageBoxes or the CommandLine display as appropriate. Event loop processing then continues normally.

For all other classes of exception the host event loop has no standard procedure and the outcome is that the host program terminates untidily. It is therefore very poor programming style to allow general exceptions to escape from the Addin. A relatively easy way to prevent this is to trap exceptions systematically at every entry point between the host event loop and the component interfaces - for example in every Command.Execute function:

```
/// <summary>
/// Execute
/// </summary>
public override void Execute()
{
    try
    {
        // Do the required function.
        DoTheJob();
    }
    catch (System.Exception ex)
    {
    }
```

```

        // Pass on PdmsExceptions and PMLNetExceptions.
        // Handle all others:
        //     (perhaps by raising a PdmsException).
    }
}

private void DoTheJob()
{
    // Do the required function...
}

```

This recommendation does not, of course, replace the routine practice of handling exceptions at the appropriate points within the Addin code when they are anticipated and can be dealt with effectively. It simply provides a safety net for unanticipated and otherwise unhandled exceptions.

## 2.9 Thread safety in a CAF Addin

C# allows code to use multiple threads and there are situations where this can be useful in an Addin. However this technique should be used with care as the host AVEVA program is not itself multiple thread safe. Thus there can be only one thread at a time which executes functions in DbLayer, CAF or other host core code.

## 3 Menu and Command Bar Customisation

User access to functionality provided via an addin is normally provided through the use of menus and/or tools on a commandbar. Reference was made earlier, in the section about addin commands, that the CAF provides a mechanism to allow the menus and commandbars for an application to be defined in a "User Interface Customisation" (UIC) file. This section provides details of how a CAF based application can be configured to load a UIC file, and how the UIC file can be edited using the applications interactive user interface customisation tool. Newer modules (such as Tags) use the new Ribbon commandbar, this is configured in the same way as the original toolbar (detailed below).

### 3.1 Configuring a Module to Load a UIC File

Each CAF based application has an XML configuration file which contains a list of UIC files that the application should load at start-up. The default location for this file is in the same directory as the application executable. It has a filename of the form <Module Name>Customisation.xml. For example the AVEVA Marine module Hull Design has a file called HullDesignCustomisation.xml. The content of this file is reproduced below. By default the UIC files are also expected to be in the same directory as the application executable. You can however specify the full path to the UIC file including the use of UNC pathnames. It is also possible to define a project specific UIC file. The string "\$1" in the UIC path will be replaced with the current project name.

If during addin development you do not wish to modify the customisation configuration file in the install directory (this is particularly likely to be the case if you are using a server based installation) then the default location of the customisation configuration file can be overridden using the environment variable CAF\_UIC\_PATH. You can then edit a copy of the file and point this environment variable at the directory you copy it to.

```
<?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="Hull General" Path="AVEVA.Marine.UI.HullGeneral.uic" />
    <CustomizationFile Name="Hull Design" Path="AVEVA.Marine.UI.HullDesign.uic" />
    <CustomizationFile Name="Project" Path="$1.uic" />
  </UICustomizationFiles>
</UICustomizationSet>
```

The order of the UIC files in this configuration file is significant. They are loaded in order since it is possible for a UIC file to define a tool which is hosted in a menu or on a commandbar defined in a UIC file already loaded.

A new UIC file can be added to a module simply by adding a new line to the corresponding customisation configuration file. The actual content of the UIC file will be created using the interactive user interface customisation tool described below.

As well as adding to the customisation configuration file an addin can also load a UIC file directly using the AddUICustomisationFile method of the CommandBarManager.

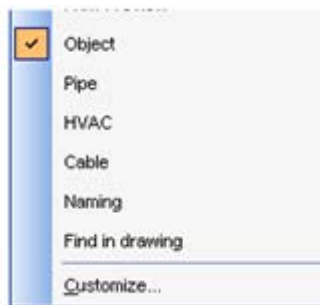
```
// Load a UIC file for the AttributeBrowser.
CommandBarManager commandBarManager = (CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));

commandBarManager.AddUICustomizationFile("AttributeBrowser.uic",
"AttributeBrowser");
```

This UIC file will be loaded before those define in the Customisation configuration file, so it must create its own menu or commandbar to host its tools.

## 3.2 Editing the UIC File

The content of a UIC file is created using the CAF based applications built-in interactive user interface customisation tool which is accessed via the Customize entry at the bottom of the commandbar area popup menu.



The customisation dialog is comprised of seven main areas:

1. CommandBar Preview Area - displays a preview of a selected CommandBar.
2. Active Customisation File - allows the selection of a loaded customisation file for editing.
3. Tree showing CommandBars, Menubars, Context Menus and Resource Files.
4. List of tools.
5. Property grid
6. Action buttons
7. Resource Editor - displayed when a resource file is selected.



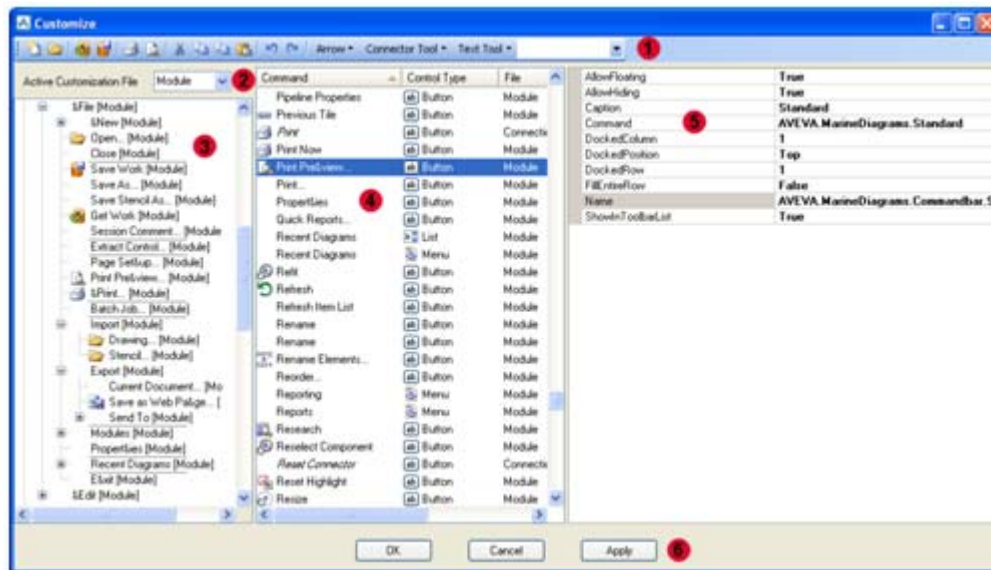


Figure 3:1. Customisation Dialog



Figure 3:2. Customisation Dialog - Resource Editor

### 3.2.1 Selection of Active Customisation File 2

The customisation system supports the concept of multiple configuration files.

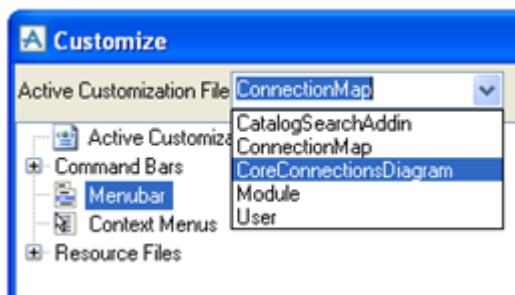


Figure 3.3. Selecting the Active Customisation File

Any number of levels of customisation file can be defined either using the module customisation configuration file or the `CommandBarManager`. `AddUICustomizationFile()` method, and they are layered on-top of each other in the order they appear in the list. Selecting an entry in the list will update the tree view (3) and listview (4) with all configuration information defined in the selected file and those above it. Any items in the tree or listview which are not defined in the currently active customisation file are displayed with an italic font in grey. Any customisation file which a user does not have write access to does not appear in the drop-down list, but its contents are included in the tree and list of tools.

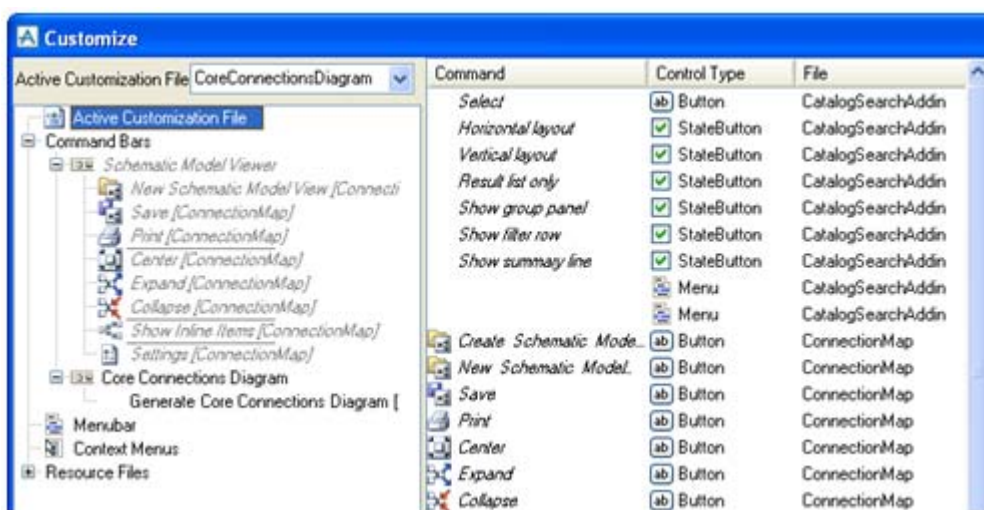


Figure 3.4. Customisation Dialog: Indication of editable items

Here the Button "Save" is defined in the "ConnectionMap" customisation file. It is displayed in italics along with the "Schematic Model Viewer" Command Bar since these items are not defined in the currently selected customisation file.

The selected customisation file will also be made the active customisation file and any edits will only be possible to customisation information defined in this file.

### 3.2.2 The Tree 3

The tree is populated with a representation of the Active Customisation file: CommandBars and their contents: Menubars and their entries and Context Menus defined in each of the

configuration files above and including the currently selected file. It also contains a list of the currently loaded resource files. The Tree View supports the following user interactions:

### Selecting a Node in the Tree

The property grid (5) is updated with the corresponding object so that its properties can be edited. Only objects which are defined in the currently active customisation file can have their properties modified. Selecting a node in the tree which represents an object defined in a non-active file will update the property grid, but it will be disabled thus preventing modification.

If the selected node represents a CommandBar then a representation of the CommandBar will be displayed in the Preview Area (1).

If the selected node represents a resource file then the resource editor (7) will be displayed instead of the property grid.

### Drag & Drop within the Tree

The tools owned by a MenuTool or a CommandBar can be re-ordered by dragging the tool to the new desired position. MenuTools owned by the MenuBar can also be re-ordered using drag and drop.

### Node Context Menus



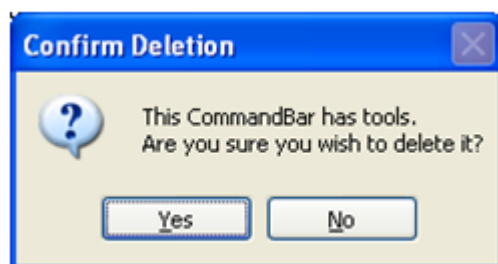
The context menu for the CommandBars Node contains the following operations:

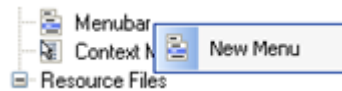
**New Command Bar** - Creates a new CommandBar with unique default name (CommandBar<N>).



The context menu for a CommandBar node contains the following operations:

**Delete** - Delete the CommandBar. If the CommandBar contains tools then a confirmation MessageBox is displayed.



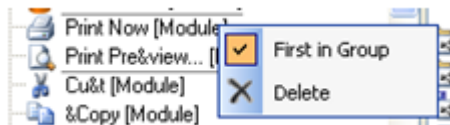


The context menu for the MenuBar node contains the following operations:

**New Menu** - Creates a new MenuTool with a unique default name (Menu<N>).

The context menu for the Context Menus node contains the following operations:

**New Menu** - Creates a new MenuTool with a unique default name (Menu<N>).



The context menu for a Tool node contains the following operations:

**First in group** - Marks the tool instance as being the first in a group. It then gets a separator drawn above it.

**Delete** - Removes the tool instance from the MenuTool or CommandBar.

### 3.2.3 List of Command Tools 4

The list is populated with each of the tools defined in each of the configuration files above and including the currently selected file.

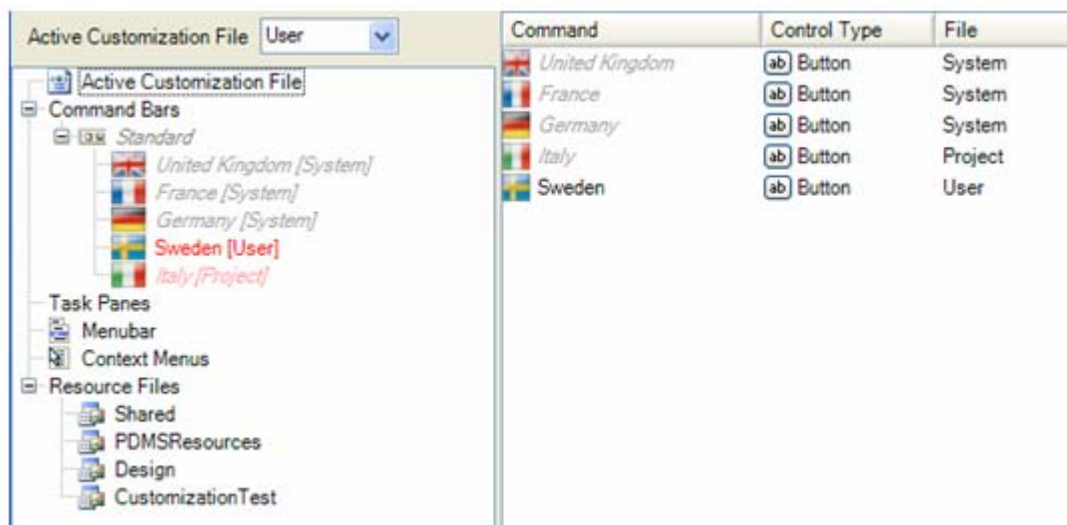
The list supports the following user interactions:

#### Selecting a Node in the List

The property grid (5) is updated with the corresponding tool object so that its properties can be edited. Only objects which are defined in the currently active customisation file can have their properties modified. Selecting a tool which is defined in a non-active file will update the property grid, but it will be disabled thus preventing modification.

#### Drag & Drop from the List to the Tree

Tools can be dragged from the list and dropped into a MenuTool or a CommandBar in the Tree. It is also possible to add tools to a menu or commandbar defined in a higher level customisation file. This allows the user to integrate their tools into an existing system defined menu hierarchy and extend system toolbars as well as creating new top level menus and new commandbars. One possible risk of adding tools to menus or toolbars defined at a higher level is that the higher level structure might at a later date be modified making these tools "orphaned", or making the data which is stored to indicate their position within the host invalid. To warn of this, these tools are displayed in red within the tree. Here the "User" defined button "Sweden" had been added to the "System" defined commandbar "Standard". Also a button "Italy" defined at project level has been added to this "System" commandbar.

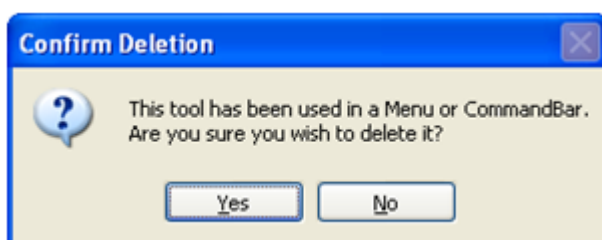


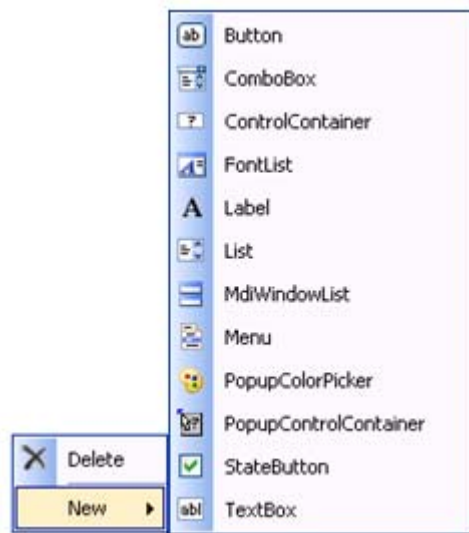
### List Context Menu

The list has the following operations on a context menu:



**Delete** - Deletes the currently selected tool. If the selected tool is not editable then the delete operation is disabled. If the tool has been used then a warning message is displayed to confirm that the delete should be carried out.



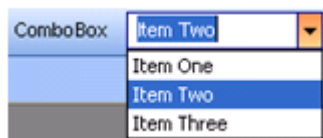


Creates a new tool of the required type with a unique default name (Tool<N>) and adds it to the list.

### Tool Types



When placed on a CommandBar the Button tool can have either an icon or a text caption. In a menu both the caption and the icon, if one is defined are displayed. The button tool simply executes the associated command when clicked.



The ComboBox tool allows selection from a list of items. The associated command provides the list via its List property.



The ControlContainer supports the hosting of any WinForms control. Here it is shown hosting a ProgressBar. The control it should host is set via the Control property which presents a list of the controls registered with the CAF using the CommandBarManager.Controls Hashtable property.

```
ProgressBar progBar = new ProgressBar();
progBar.Value = 50;
commandBarManager.Controls.Add("ProgressBar", progBar);
```



The FontList tool allows selection of a font from a pulldown list of installed fonts.



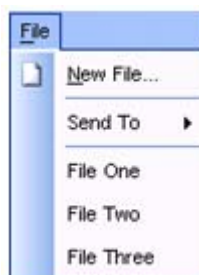
The Label tool allows the addition of a static icon and/or caption.



The List tool can only be use in a menu and will create a menu entry for each of the items returned from the List property of the associated Command object. This is typically used to implement a MRU list of files.

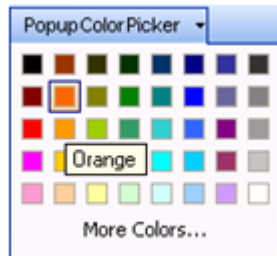


The MdiWindowList tool can also only be added to a menu. It supports the standard operations on the applications MDI Windows.





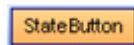
Menu tools can be added to both an existing menu to create a sub-menu or to a CommandBar.



The PopupColorPicker displays a standard colour picker in a popup window.



The PopupControlContainer tool allows any WinForms control to be hosted in a popup window. Here the MonthCalendar control is being hosted.



The StateButton tool supports the Checked state.



The TextBox tool allows the entry of any string value.

### Sorting List via Heading

It is possible to sort the list via interaction with the list column headings.

## 3.2.4 Property Grid 5

The property grid allows editing the various properties of the tools and CommandBars. The tree view (3) or the list of command (4) is kept up to date with any property changes.





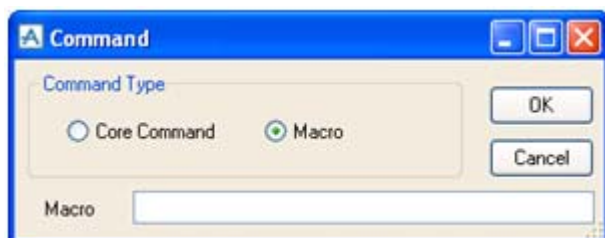
Caption	Get Work
Category	
Command	AVEVA.MarineDiagrams.FileGetWork 
Icon	 GetWork.png
MenuToolUpdateOptions	None
Name	AVEVA.MarineDiagrams.Button.FileGetWork
Shortcut	None
Tooltip	Get Work

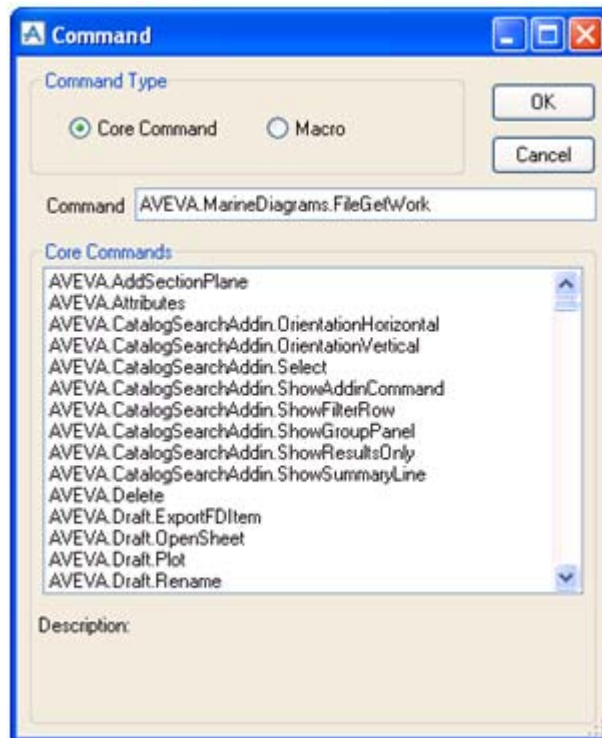
Figure 3:5. Customisation Dialog : Properties Grid

The command property can be changed by clicking on the button displayed when selecting the command property. This will display the following dialog:



By default the command type is set to Macro. The uses a special command class to invoke the entered single PML command. This enables the PML developer to use the interactive customisation tool to create their own menus and commandbars without needing to write C# command classes.

Changing the command type to "Core Command" will change the dialog to:



Here a list of the currently loaded CAF Command objects is displayed for selection.

### Command Tool Properties

The following properties can be set for all Command Tools:

- ApplicationContext
  - Currently unused
- Arguments
  - List of arguments supplied to the Command
- Caption
  - Text to display on CommandBar
- Command
  - Command to run (see above)
- DisplayStyle
  - Determines how the tool is displayed on the CommandBar
    - Default- The tool is displayed in a default style.
    - DefaultForToolType - The tool is displayed based on the default for the tools type and its location.
    - TextOnlyAlways - The tool is always displayed as text only.
    - TextOnlyInMenus - The Tool is displayed as a graphic when located on a Toolbar, and displayed as text when located on a Menu.
    - ImageAndText - The tool is displayed using its assigned image and text. This setting is ignored when the item is on a top-level menu.
    - ImageOnlyOnToolbars - The Tool is displayed as a graphic when located on a Toolbar, and displayed as image and text when located on a Menu.

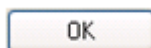
- Icon
  - Icon to display on CommandBar
- KeyTip
  - Tip is shown on the ribbon when the Alt key is pressed, aids quick navigation
- Name
  - Unique name for this tool
- Shortcut
  - Shortcut key to activate this tool
- Tooltip
  - Plain tool-tip text to display on mouse-over
- TooltipFormattedText
  - Formatted tool-tip text to display on mouse-over
- TooltipTitle
  - Title for tool-tip

The following properties are available on some Command Tools:

- AutoComplete
  - Sets whether the edit portion is automatically updated to match an item in the list as new characters are typed.
- Control
  - Control embedded in container
- Editable
  - Sets if the text is editable by the user
- MaxLength
  - Maximum number of characters the user is allowed to enter
- Password
  - Display text in password mode (hidden)
- ValueList
  - List of values to display
- Width
  - Tool width on the CommandBar

### 3.2.5 Action Buttons 6

The customisation dialog has the following action buttons:



**OK** - Saves any modified customisation files and updates the application user interface with any changes and closes the customisation dialog.



**Apply** - Updates the application user interface with any changes.



**Cancel** - Restores the application user interface to state defined by customisation files if any changes have been applied and closes the customisation dialog.

### 3.2.6 Resource Editor 7

The resource editor enables each of the currently loaded resource sets to be edited. It supports String, Icon and Bitmap resources. Each resource set can comprise of a number of resource files, comprising of the "Invariant Language" resource file and any number of locale specific resource files.

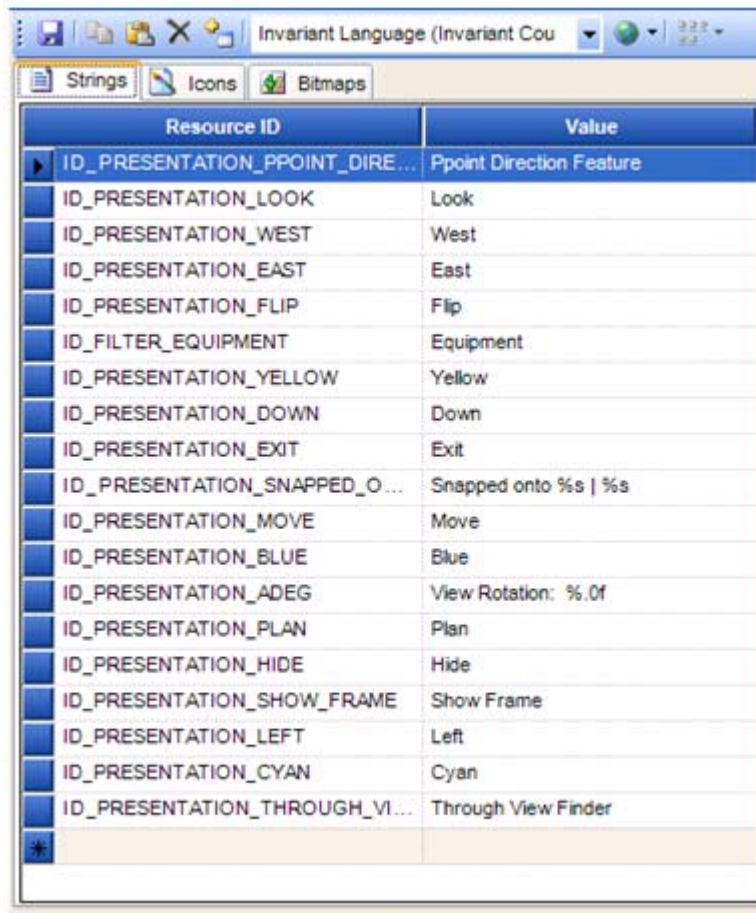


Figure 3.6. Customisation Dialog : View of string resource editor

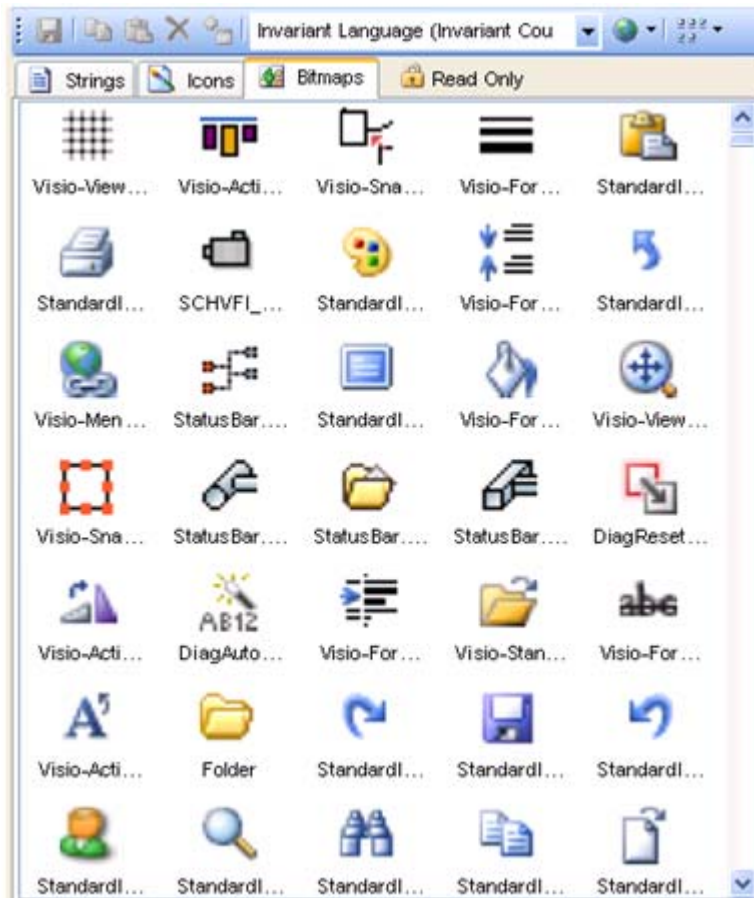


Figure 3.7. Customisation Dialog: View of bitmap resource editor

### Resource Editor Command Bar



The resource editor command bar contains the following operations:



Save the resource file.



Copy a resource, so it can be pasted into another resource file.



Paste a resource.



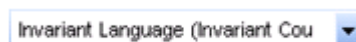
Delete a resource.



Add a new resource (Icons and Bitmaps only). New bitmaps or icons can be added by drag-and-drop from a windows explorer. The resource id is generated from the filename capitalised and prefixed with "ID\_". If there is a clash with an exiting resource then the following dialog will request confirmation of replacement.



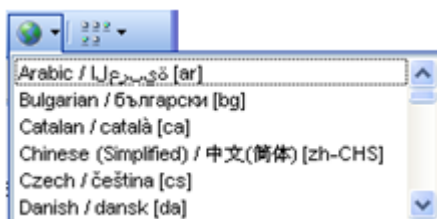
String resource can be added for modified by editing them in the grid.



Select locale specific resource file for editing.



Add a new locale specific resource file to the resource file set.

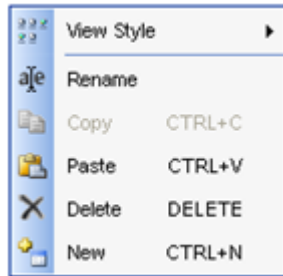


Choosing a locale from the list of locales will add a new resource file specific for this locale. This can then be selected for editing.




Change layout style.

The icon and bitmap tabs also have a popup menu which provides access to these functions as well as a rename function. Double clicking on a resource also enters the rename mode.



### Using Resources for Properties

Properties of tools which are strings or the Icon property can use resources for their value. Reference to a resource is made using the form <ResourceSet Name>:<Resource Id> as shown below:

Caption	Resize
Category	
Command	
Icon	 StandardIcons:ID_RESIZE
MenuToolUpdateOptions	None
Name	Aveva.Resize
Shortcut	None
Tooltip	MyResources:ID_RESIZE_TOOLTIP

The caption and icon properties can be set on a tool in the tool list by drag-and-drop from the resource editor onto the tool.

### Standalone Resource Editor

There is a standalone version of the resource editor available to enable resource files to be created and edited. This program is called "ResourceEditor.exe" and can be found in the installation directory.

When started it will display a file selection dialog to enable you to choose an existing resource file or give the name of a new one.





## 4 Database Interface

The classes fall into the following groups:

1. Data model definition classes
2. Basic database access, query and modification.
3. Filters and iterators
4. Dabacon table access
5. DB, MDB, Project access
6. Events and pseudo attribute plugins

### 4.1 Data Model Definition Classes

The classes are:

- DbElementType
- DbAttribute
- DbElementTypeInstance
- DbAttributeInstance

#### 4.1.1 DbElementType

##### Overview

There is a single class for all element types.

The purpose of the class is to:

- Instances of the class identify the element type. i.e. methods on other classes take a DbElementType as an argument rather than a string to denote an Element type. The DbElementType has 'value' semantics for comparison purposes. i.e. comparing of two DbElementTypes will always return true if they refer to the same type, even if they are different objects.
- Access the metadata associated with element types.

External to PDMS/Marine, the hash value can be used to uniquely identify an Element type. The hash value is a 32 bit integer.

##### Constructors

In C# a DbElementType object may be obtained in one of three ways:

1. Use the globally defined instances in DbElementTypeInstance. This is the recommended and easiest way to obtain a DbElementType.

2. The static 'GetElementType' method may be used to return a DbElementType given the Element Type name. This is only needed for UDETs. The colon must be included as part of the name.
3. There is also a static 'GetElementType' method that may be used to return a DbElementType given the hash value. This is only needed where a hash value has been stored outside of PDMS/Marine.
4. Various methods on other classes, such as DbElement will return a list of DbElementTypes.

### Methods

The methods allow the Element type metadata to be accessed. E.g.

- Name
- Description
- BaseType (for UDETs)
- List of attributes
- List of UDAs
- Allowed members
- Allowed owners
- Types that may appear above this element type
- Types that may appear below this element type
- Element types that may be referenced from a given attribute on this element type
- Database types in which it occurs

### Related ENUMS

DbNounManual- Manual category

DbType - DB type

### Related Pseudo Attributes

Pseudo attributes that provide similar information on a particular element instance (i.e. on a DbElement) are:

Attribute Name	C# Data Type	Description
HLIS	DbElementType[ ]	List of all possible types in owning hierarchy
LIST	DbElementType[ ]	List of all possible member types
LLIS	DbElementType[ ]	List of all possible types in member hierarchy
OLIS	DbElementType[ ]	List of all possible owner types
REPTXT	String	Reporter text used for element type
ATTLIST	DbAttribute[ ]	List of all visible attributes for element
PSATTS	DbAttribute[ ]	List of pseudo attributes
UDALIS	DbAttribute[ ]	List of UDAs

## 4.1.2 DbAttribute

### Overview

This is very similar to DbElementType. There is a single class for all attributes.

The purpose of the class is to:

- Access the metadata (i.e. data about data) associated with attributes. E.g. type, name, length
- Identify attributes. i.e. methods on other classes should always take a `DbAttribute` rather than a string as an argument to denote the attribute. Any comparison of attribute identity should be done by comparing `DbAttribute` objects.

The class should not be confused with the attribute value. The actual Attribute value for a particular Element can only be accessed via the `DbElement` class. Comparing two `DbAttributes` just compares whether they identify the same attribute, the comparison does not look at attribute values in any way.

External to PDMS/Marine, the hash value can be used to uniquely identify an Attribute. The hash value is a 32 bit integer.

### Constructors

In C# a `DbAttribute` object may be obtained in the following ways:

1. Use globally defined instances. Each attribute has a globally declared instance in the `DbAttributeInstance` class. This is the standard way of specifying an attribute.
2. Look up the `DbAttribute` given the attribute name. This is only needed for UDAs, since all other attributes can be obtained from the global instances. The colon must be included as part of the name.
3. Look up the `DbAttribute` given the attribute hash value. Generally this is only needed if reading the hash value from an external system.
4. Various methods on other classes, such as `DbElement`, will return a list of `DbAttributesTypes`.

### Methods

The methods allow the following metadata to be accessed:

Attribute Type

Units

Name

Description

Category

Size

Allowed Values

Allowed ranges

Is a UDA

Is a pseudo attribute

Whether the attribute may  
take a qualifier

### Example:

Find the type of attribute XLEN. We use the global instance of XLEN on the DbAttributeInstance class.

```
using ATT=Aveva.Pdms.Database.DbAttributeInstance;
DbAttributeType xlenType= ATT.XLEN.Type;
```

### Related ENUMS

DbAttributeUnit - Type of units, e.g. distance or bore or none.

DbAttributeType - Type of attribute. One of

```
INTEGER = 1,
DOUBLE = 2,
BOOL = 3,
STRING = 4,
ELEMENT = 5,
DIRECTION = 7,
POSITION = 8,
ORIENTATION = 9
```

DbAttributeQualifier - used to determine what sort of qualifier an attribute has

## 4.1.3 DbElementTypeInstance

This class contains instances of DbElementType for every element type in PDMS/Marine. These instances may be used wherever a DbElementType is expected. E.g. if a method MyMethod() takes an DbElementType, then you could write:

e.g.

```
MyMethod(Aveva.Pdms.Database.DbElementTypeInstance.EQUIPMENT);
```

It is often convenient to add a using statement for these instances. E.g.

```
using NOUN=Aveva.Pdms.Database.DbElementTypeInstance;
MyMethod(NOUN.EQUIPMENT);
```

## 4.1.4 DbAttributeInstance

This class contains instances of DbAttribute for every element type in PDMS/Marine. These instances may be used wherever a DbAttribute is expected. E.g. if a method MyMethod() takes a DbAttribute, you could write:

e.g.

```
MyMethod(Aveva.Pdms.Database.DbAttributeInstance.XLEN);
```

It is often convenient to add a using statement for these instances. E.g.

```
using ATT=Aveva.Pdms.Database.DbAttributeInstance;
MyMethod(ATT.XLEN);
```

## 4.2 Element Access

### 4.2.1 DbElement Basics

#### Overview

This section describes the `DbElement` class. The `DbElement` class is the most widely used class and it covers a large proportion of the database functionality that will be used in practise.

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.

#### Constructors

An instance of a `DbElement` may be obtained as follows:

- There is a static `GetElement()` method with no arguments to return a 'null' `DbElement`.
- There is a static `GetElement()` method which returns a `DbElement` given a name. This name should include the '/'.  
e.g.
- There is a static `GetElement()` method which returns a `DbElement` given a ref (two long int array) and type. This is only needed where a reference has been stored externally to PDMS/Marine.
- There are many methods on various classes which return `DbElements`

```
DbElement vess1 = DbElement.GetElement("/VESS1");
```

#### Identity

The `DbElement` object encapsulates the identity of the database object. Any comparison of database objects must be done using `DbElements`. The `DbElement` has 'value' semantics for comparison purposes. i.e. comparing of two `DbElements` will always return true if they refer to the same element, even if they are different objects. `DbElement` instances should be used in all cases where the identity of an element is being passed or stored.

A `DbElement` can be identified externally to PDMS/Marine by a combination of the ref number AND type. The ref number is a two long integer, for example: =123/4567.

#### Element Validity

A `DbElement` need not represent a 'valid' element. There are a number of reasons why a `DbElement` might be invalid:

- The element is in a DB not opened.
- The element has been deleted.

There is a `IsValid()` method to test if a `DbElement` is valid.

If the `DbElement` is invalid then all attribute access and database navigation will fail for that `DbElement`.

### Error Handling

The error handling techniques used are:

1. Some methods raise a `PdmsException`
2. Some methods return false if the operation can not be done.
3. For navigation operations, if the navigation does not succeed then a 'null' element is returned. A null element can be tested using the 'IsNull' method. It will have a reference of =0/0.

### Basic Properties

`DbElement` has the following basic methods:

`ToString()` - Returns the Name of the element. If unnamed, it returns the constructed name.

`GetElementType()` - Returns the `DbElementType`.

There are a number of pseudo attributes that return slight variations on name and type, as below.

Type related:

Attribute Name	C# Data Type	Qualifier	Description
ACTTYPE	<code>DbElementType</code>		Type of element
AHLIS	<code>DbElementType (200)</code>		List of actual types in owning hierarchy
OSTYPE	<code>DbElementType</code>		Shortcut for "Type of owner"
TYPE	<code>DbElementType</code>		Type of the element, ignoring UDET
TYSEQU	<code>int</code>		Type Sequence Number

Name Related:

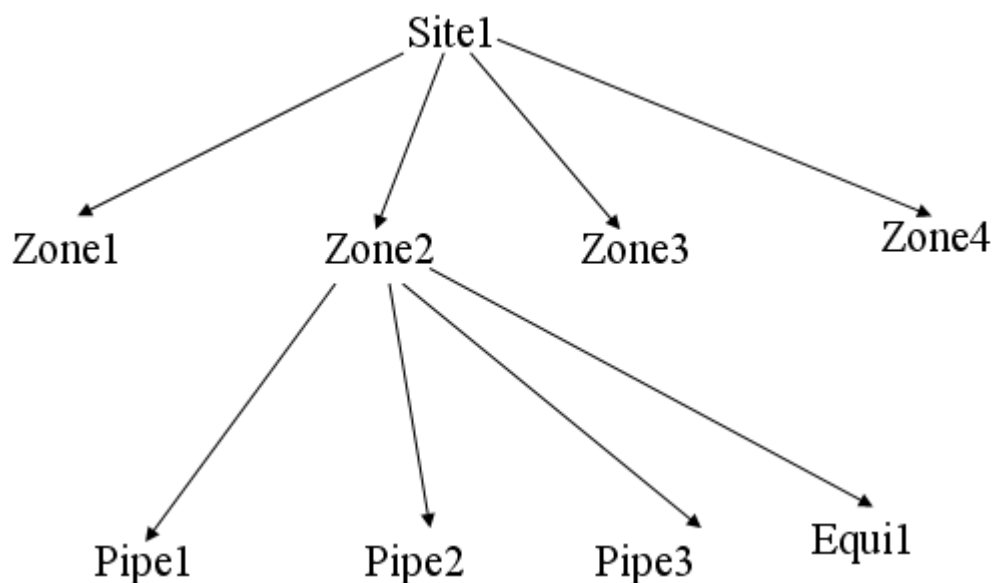
Attribute Name	C# Data Type	Qualifier	Description
CUTNAM	<code>String</code>	<code>int</code>	Full name of element, truncated to n characters
CUTNMN	<code>String</code>	<code>int</code>	Full name of element (without leading slash) truncated to n characters
FLNM	<code>String</code>		Full name of the element
FLNN	<code>String</code>		Full name of the element (without leading slash)
ISNAMED	<code>Bool</code>		True if element is named
NAMESQ	<code>String</code>		Type. sequence number and name of element
NAMETY	<code>String</code>		Type and name of the element

Attribute Name	C# Data Type	Qualifier	Description
NAMN	String		Name of the element (without leading slash)
NAMTYP	String		Type and full name of element

## 4.2.2 Navigation

### Basic Navigation

There are basic methods to navigate the primary hierarchy. e.g. consider the following hierarchy:



If we are sitting at Zone2, we can navigate as follows:

```
using NOUN=Aveva.Pdms.Database.DbElementTypeInstance;
DbElement zone = DbElement.GetElement("/Zone1")
DbElement 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;
temp=zone2->LastMember(); // temp is now Equi1
DbElement temp=pipe1.Next(NOUN.EQUIPMENT); // temp is Equi1
DbElement temp=pipe1.Previous(); // temp is 'null' as there is no
previous element. This can only be tested using the 'IsNull' method
```

Scanning the database is a very common operation. For this reasons there are additional iterator and filter classes that ease this task. These are described in the section on Filters/Iterators.

**Pseudo Attributes Relating to Element Navigation**

Attribute Name	Data Type	Qualifier	Description
ALLELE	DbElement[ ] )	DbElementType	All elements in the MDB of a particular type
CONNECTIONS	DbElement[ ]		Connections
CONNECTIONS H	DbElement[ ]		Connections for all descendants
CONNER	String	Int	Connection error message
DDEP	Int		Database depth within hierarchy (World is 0)
FRSTW	DbElement	String	Reference of first world of given DB type in current MDB
MAXD	Int		DB hierarchy depth of lowest level item beneath element
MBACK	DbElement[ ]	*DbElementType	Members in reverse order
MCOU	Int	*DbElementType	Number of Element Members of Given type
MEMB	DbElement[ ]	*DbElementType	All members, or members of specific type
OWNLST	DbElement[ ]		Owning hierarchy
PARENT	DbElement	*DbElementType	Reference of ascendant element of specified type
SEQU	Int		Sequence Position in Member List
TYSEQU	Int		Type Sequence Number

'\*' - qualifier is optional

**Secondary Hierarchies**

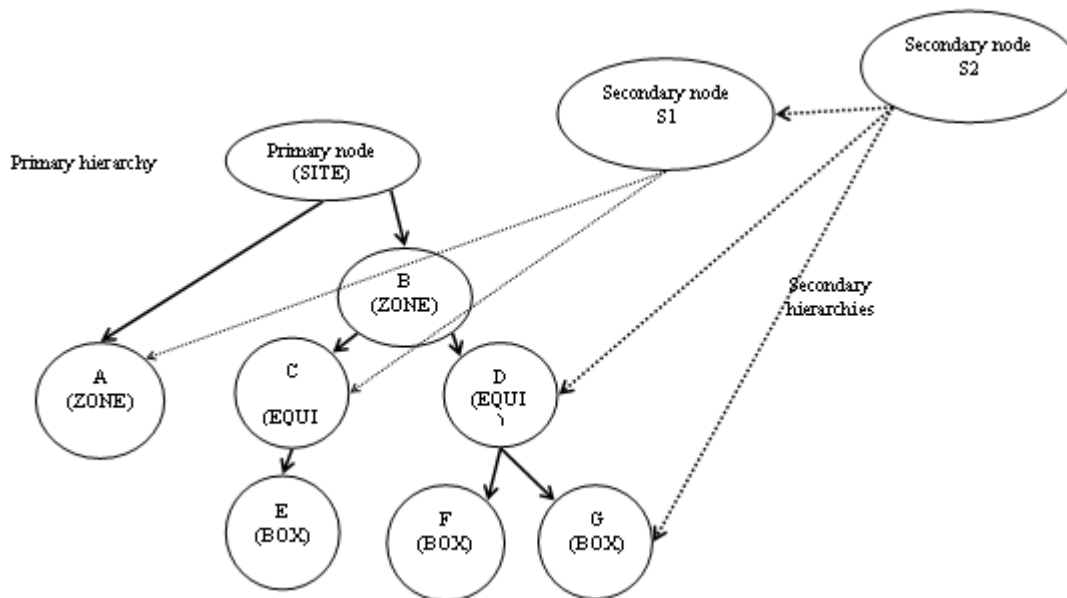
Standard navigation methods do not work for descending a secondary hierarchy. Pseudo attribute SMEMB must be used. E.g. to get the design elements below a GPSET, you must query the SMEMB attribute on the GPSET.

Pseudo attributes relating to secondary hierarchies are:

Attribute Name	Data Type	Description
GPRXS	DbElement[ ]	Group proxy items referring to element
GROUPS	DbElement[ ]	GPSETs in which element occurs
SEXPND	DbElement[ ]	Recursive expansion of SMEMB
SITEM	DbElement	Reference from secondary hierarchy
SMEMB	DbElement[ ]	Immediate members in secondary hierarchy

The difference between SMEMB and SEXPND is that SEXPND allows for recursive secondary hierarchies. Where there are no recursive possibilities, SMEMB and SEXPND will be the same. The following example illustrates the difference between SMEMB and SEXPND:





In the example there are two secondary nodes S1, S2. S1 holds element A, C, and S2 holds S1,D,G.

For S1,

SMEMB returns A,C

SEXPND returns A,C

For S2,

SMEMB returns S1,D, G

SEXPND returns A,C, D, G

### 4.2.3 Getting Attribute Values

#### Basic Mechanism

The attributes available on for a DbElement will depend on its type. E.g. a site will have different attributes to a branch. 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 (a DbAttribute object). There are separate methods for each attribute type (int, double etc), plus separate methods for single values or arrays.

e.g.

```
using ATT=Aveva.Pdms.Database.DbAttributeInstance;
double length=myEle.GetDouble(ATT.XLEN);
```

This call returns the value of attribute XLEN. If 'myEle' is not a BOX then an exception will be raised.

If there is any doubt as to whether 'myEle' is a BOX or not, then there are a set of methods that return false if the attribute does not exists. E.g.

e.g.

```
using ATT=Aveva.Pdms.Database.DbAttributeInstance;
double length;
```

```
if (!myEle.GetValidDouble(ATT_XLEN,length)) {
    // handle error in some way
}
```

In addition there is a `IsAttributeValid()` method that can be used to test if an attribute is valid or not.

The basic mechanism works for all attributes including UDAs and pseudo attributes.

The attribute types supported are:

int, int[]

double, double[]

bool, bool[]

string, string[]

DbElement, DbElement[]

DbElementType, DbElementType[]

DbAttribute, DbAttribute[]

Position

Direction

Orientation

Expression

**Note:** The methods that are called `GetAttribute`, `GetAttributeArray` are the 'getattribute' methods that return `DbAttributes`. Similarly the `GetElement`, `GetElementArray` methods are the 'getattribute' methods that return `DbElements`. This is confusing since `GetElement` is also the name of the method to return a `DbElement` given a name. We may change the names of these in the future.

### List of Valid Attributes

There are two lists of valid attributes:

1. The list of system attributes. Typically these attributes can be queried and set. These can be queried using the `GetAttributes()` method.
2. There is a list of pseudo attributes that may be queried. Typically this list is large, running into the hundreds. This list can be obtained via the PSATTS attribute. N.B. querying PSATTS can be slow.

Related pseudo attributes are:

Attribute Name	Data Type	Description
ATTLIST	DbElementType[]	List of all visible attributes for element
ATTRAW	DbElementType[]	List of raw attributes
PSATTS	DbElementType[]	List of pseudo attributes
RLIS	DbElementType[]	List of rules set

UDALIS	DbElementType[]	List of UDAs
UDASET	DbElementType[]	List of UDAs set

### Qualifier

Many attributes take a qualifier. The qualifier is the extra information to make the query. Examples of where a qualifier is used are:

1. Querying a ppoint position (PPOS) requires the ppoint number
2. The ATTMOD attribute can be used to query when an attribute was modified but it needs to be given the identity of the attribute.
3. A direction/position may be queried with respect to another element

The definition of what pseudo attributes take what qualifier is described in the data model reference manual.

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, although few current attributes require multiple valued qualifiers. There is a separate method to set the WRT element.

There is a set of query routines that take a qualifier as an extra argument.

e.g. to query the bore of ppoint 1:

```
DbQualifier qual=new DbQualifier();
qual.Add(1);
double bore=myele.GetDouble(ATT.PPBO, qual);
```

### Getting an Attribute as a Formatted String

If the attribute value is to go onto a form then the value must be formatted correctly. The correct formatting is not always obvious or available. Therefore there are special methods to return any attribute as a formatted string. This will format the attribute into the form that would appear at the command line. For example if the attribute is a position and we are working in finch unit, then something like 'W 39'4.7/16 N 59'0.85/128 U 4'0.31/128' might be returned.

The method to do this is `GetAsString()`.

**Note:** There is no generalised method for setting an attribute from a string.

## 4.2.4 Database Modifications

### Overview

The following modifications are allowed:

- Set attribute
- Create element
- Delete element
- Copy element
- Change type
- Move element

### The Modification Pipeline

When an element is modified, it is not simply the case of directly updating the database. For example, when setting an attribute the following sequence takes place:

1. Check that the attribute value is the right type
2. Check against any valid values or ranges
3. If the new value is the same as before, then return.
4. Check against the user access control defined in the system DB.
5. Check that the DB is open in write
6. Check LOCK flag
7. Do claim check, and claim if needed if DB is 'implicit claim'
8. Make attribute specific checks.
9. Make the actual database change.
10. Add to change list
11. Invoke attribute specific follow up code.
12. Update any dynamic rules affected

It can be seen that making what appears to be a simple modification is actually quite complicated.

In particular a lot of errors are possible. Errors may be generated by any of the steps 1-2 plus steps 4-8.

If an error occurs the code will raise a `PdmsException`. The calling code needs to allow for this.

Clarification of some of these errors is as follows:

**Check that the attribute value is the right type** - i.e. if a text attribute then the new value must be text. E.g. setting DESC to 99 will raise an error.

**Check against any valid values or ranges** - This will check the value against any user provided criteria stored in the dictionary DBs.

**Check access control** - An error will be raised if the modification is not allowed. An error will also be raised at this point if the attribute is not valid for the given element.

**Check that the DB is open in write** - An error will be raised if 'read only'.

**Check LOCK flag** - An error will be raised if the LOCK flag is true.

**Do claim check, and claim if needed (and implicit claim is allowed)** - An error will be raised if:

- The DB uses explicit claims and the primary element has not been claimed.
- The primary element is claimed by another user or extract.

**Call the legality checking routines to ensure that the modification is valid** - These checks are made by the plugged in application specific code. Thus potentially any error could be raised.

### Claiming Elements

If a DB has been set up to be 'implicit' the user will claim elements as needed without having to do explicit claims.

There are some methods on `DbElement` to claim/release individual elements or a hierarchy of elements. If working on a hierarchy of elements, and errors occur, then the system will still do what it can.

There are also methods on the `MDB` class to claim/release a list of elements. Performance is improved by claiming or releasing elements in bulk, thus in general the `MDB` methods should be used rather than the ones on `DbElement`.

The granularity of claim is at the level of primary element. This is described in the data management reference manual.

### Pseudo Attributes Relating to Claims

Attribute Name	C# Data Type	Description
CLMID	String	Unique system ID of user claiming element
CLMNUM	int	User or extract number claiming element. Extract numbers are negative
CLMTIE	DbElement[]	Reference to elements that are automatically claimed along with this element
NPDESC	DbElement[]	List of non primary offspring
OKCLA	bool	True if element may be claimed
OKCLH	Bool	True if element and hierarchy may be claimed
OKREL	bool	True if element may be released
OKRLH	Bool	True if element and hierarchy may be released
PRIMTY	Bool	True if element is primary
PRMMEM	bool	True if there are any primary elements amongst descendants
PRMOWN	DbElement	Primary owning element (will be itself if primary)
USCLHI	DbElement[]	Elements in descendant hierarchy claimed to this user
USERC	String	User name of user claiming element
USNCLH	DbElement[]	Elements in descendant hierarchy not claimed to this user

### Set Attribute

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

```
ele.SetAttribute(ATT.DESC, "Example description");
```

There are no methods that take a qualifier.

The code should always be prepared to handle any exceptions raised.

There is a boolean method `IsAttributeSettable` to test if a given attribute may be set. This does not take the new value, so it is possible that the actual setting will still fail if the new value is invalid.

### Creating Element

Creation is relatively straightforward. Elements can be created:

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

If creating an element below a given element then the position in the list must be given. E.g.

```
DbElement myNewPipe=myZoneElement.Create(1,NOUN.PIPE);
```

This creates a new PIPE below 'myZoneElement' as the first member. If the position is beyond the end of the current members list, it will create it at the end.

An exception will be raised if the required type of element cannot be created at the required point.

There is a boolean method `IsCreatable` to test if an element of a given type could be created at the required location.

### Deleting element

The method to delete an element is `Delete()`. e.g.  

```
MyEle.Delete();
```

All descendants in the primary hierarchy will be deleted. It will not delete elements in any secondary hierarchies.

There is a boolean method `IsHierarchyDeleteable` to test if an element can be deleted.

### Copy

As with `Delete`, the standard `COPY` method copies the primary hierarchy. The following points are relevant when copying:

- Certain attributes are not copied. e.g. NAME.
- For Name attributes, automatic renaming is possible
- The element may be copied to an existing element, or as a new element.
- Cross references within the copied hierarchy are automatically updated to the copied element. References external to the copied hierarchy are left unchanged.

There are further details in the reference manuals.

An example of the `DbElement` method is:

```
myEle1.Copy(myEle2);
myEle1.CreateCopyHierarchyAfter(myEle2,copyoption);
```

The first example copies 'myEle2' on top of myEle1. The second example copies myEle2 and all its descendants to a new element after myEle1.

There are methods to:

- Copy a single element to an existing element
- Copy an element and all its descendents to an existing element
- Copy an element and all its descendents to a new element
- Copy common attributes ('CopyLike') between existing elements

There is a separate class `DbCopyOption` to hold the options when doing a copy. These options are only available when copying an element and all its descendents.

The element types must match if copying to an existing element.

There is a boolean method `IsCopyable` to test if an element can be copied.

### Moving Element

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

- Inset 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.

An error will be raised if the element is not allowed at the new location.

There is a boolean method `IsInsertable` to test if an element can be moved.

Currently there is not an exposed C# method for moving elements between DBs. This will be exposed shortly.

### Changing Type

It is possible to change the type of certain elements.

When a type is changed, the attributes are copied from the old to the new value. Any attributes not valid for the new type are lost. Thus a round trip of changing back to the original type may represent a loss of data.

### Pseudo Attributes Relating to Modifications

Attribute Name	Data Type	Qualifier	Description
DACCLA	bool		True if DAC allows element to be claimed
DACCOH	bool		True if DAC allows element hierarchy to be copied to another DB
DACCOP	bool		True if DAC allows element to be copied to another DB
DACCRE	bool	DbElementType	True if DAC allows element to be created
DACDEL	bool		True if DAC allows element to be deleted
DACERR	String	DbAttribute	Returns the DAC error
DACEXH	Bool		True if DAC allows element hierarchy to be exported
DACEXP	Bool		True if DAC allows element to be exported
DACISS	Bool		True if DAC allows element to be issued
DACMOD	bool	DbAttribute	True if DAC allows attribute of element to be modified
MODATT	Bool	DbAttribute	True if attribute of element can be modified

MODEL	Bool	DbAttribute	True if element can be deleted
MODERR	string	DbAttribute	Returns the error text that would occur if attribute was modified

## 4.2.5 Storage of Rules and Expressions

### Database Expressions

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

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

Database expressions are stored in various places in PDMS/Marine as follows:

- As part of a rule
- Parameterisation of the catalogue

There is a DbExpression class to hold an expression. An DbExpression may be obtained in one of the following ways:

- Use the static method `internalParse()` that creates a DbExpression from a string. e.g. `internalParse("XLEN * 100" )`
- Retrieve an expression from a rule
- Retrieve an expression from a catalogue parameter

Having got an DbExpression there are two things that can be done with it:

1. Turn it back into text
2. Evaluate it against a given element

The methods to evaluate an expression against an element are on the DbElement class. There are different methods depending on the expected result of the expression. The method names start with 'Evaluate'. The method that returns a double has an argument for the units (distance/bore/none). The result will always be in millimetres (mm). E.g.

```
DbExpression expr = DbExpression.Parse("DIAM OF PREV + 2");
double dval;
DbAttributeUnit units = DbAttributeUnit.DIST;
dval = nozz1.EvaluateDouble(expr4, units);
```

In this case the core system can work out that it is a distance, since DIAM is a distance. Thus although we specified that it was a distance, it was not strictly needed. Thus if the distance units were inch or finch then the '+2' would be interpreted as '+2inches'.

However consider:

```
DbExpression expr = DbExpression.Parse("10");
double dval;
DbAttributeUnit units = DbAttributeUnit.DIST;
dval = nozz1.EvaluateDouble(expr4, units);
```



In this case we do not know if "10" is 10mm or 10 inches. We must tell the system that it is a 'distance' unit. The system then interrogates the current distance units to return the result. If the current distance units is mm, then dval will be 10.0. If the current distance units is inch/finch then "10" is interpreted to mean 10 inches and is hence returned as 254. N.B. if the value is required to be formatted for output, then it must be converted back in all cases. A method will be added to return values in local units at a later point.

### Rules

PDMS/Marine rules consist of an expression and a dynamic/static flag. When constructing a rule, the expression type is also needed.

There are methods on DbElement to:

- Set a rule for any attribute.
- Get a rule for any attribute
- Evaluate a rule
- Verify that the rule result is up to date

### Pseudo Attributes Relating to Rules and Expressions

Attribute Name	Data Type	Qualifier	Description
NRULEE	Int		Number of rule inconsistencies on element
RCOU	Int		Number of rules on element
RULEER	String	int	Text of "nth" rule error
RULSET	DbAttribute[]		List of rules set

## 4.2.6 Comparison of Data with Earlier Sessions

There are no explicit methods exposed in C# which cover comparison across sessions. It is not currently possible to set the comparison date in C#.

There are however a number of pseudo attributes that can be accessed as follows:

Attribute Name	Data Type	Qualifier	Description
ATTMOD	Bool	DbAttribute	True if specified attribute has been modified this session
ATTMODC	bool	DbAttribute	True if specified attribute has been modified since comparison date
ATTMODLIST	DbAttribute []	int	List of attributes modified since given session
ATTMODLISTC	DbAttribute []		List of attributes modified since comparison date
CRINFO	string		DB creation information
DBSESS	int		Last DB session
ELECRE	bool	int	True if created since given session
ELECRC	bool		True if created since comparison date
ELEDEL	bool	int	True if deleted since given session

Attribute Name	Data Type	Qualifier	Description
ELEDEL	bool		True if deleted since comparison date
ELEMOD	bool	int	True if modified since given session
ELEMODC	bool		True if modified since comparison date
EXMOD	bool		True if element modified in this extract
HIST	Int[]	DbAttribute	History of sessions in which element or specified attribute were made
LASTM	string	DbAttribute	Date of last modification
MSESS	int	DbAttribute	Last session number
PRVSES	int	int	Previous Session to that specified
RULEMOD	bool	DbAttribute	True if rule modified this session
RULEMODC	Bool	DbAttribute	True if rule modified since comparison date
RULEMODLIST	DbAttribute []	int	List of rules modified since given session
RULEMODLISTC	DbAttribute []		List of rules modified since comparison date
SESCLA	int		Session Of Claim
SESSC	string	int	Comment for specified session
SESSCA	Int[]		List of pairs of DB number, last session number for all referenced DBs
SESSCH	Int[]		As for SESSCA, but searches all descendants for referenced DBs
SESSD	string	int	Date of specified Session
SESSM	int	DbAttribute	Session Of Last Modification
SESSNO	int		Current opened Session
SESSU	string	int	User creating specified session
USERM	string	DbAttribute	User making last modification

## 4.3 Filters/Iterators

### 4.3.1 Iterators

The `DBElementCollection` class can be used to iterate through the database hierarchy. The iterator is created with a root element and an optional filter. There are then methods to step through the hierarchy and return the element at the current location.

E.g. to look for all nozzles below a given element

```
TypeFilter filt = new TypeFilter(DbElementTypeInstance.NOZZLE);
DBElementCollection collection;

collection = new DBElementCollection(ele, filt);
```

The collection can be iterated through using foreach

```
foreach (DbElement ele in collection)
{
    ... Do something
}
```

The iterator class has been written to avoid unnecessarily scanning parts of the database that will not match the filter. e.g. if looking for all boxes then the iterator will not bother to look below pipes.

### 4.3.2 Filters

There are a variety of ready built filter classes available. The main ones are:

**TypeFilter** - True for given Element type(s)

**AttributeFalseFilter** - True if given attribute is false

**AttributeTrueFilter** - True if given attribute is true

**AttributeRefFilter** - True if given attribute value matches given DbElement

**AndFilter** - AND two existing filters.

**OrFilter** - OR two existing filters

**BelowFilter** - Filter to test if element is below an element for which the given filter is true.

## 4.4 Dabacon Tables

### 4.4.1 Overview of Dabacon Tables

In the Dabacon schema we can define an attribute to be an indexed attribute. Indexed attributes go into a Dabacon table.

Tables consist of a list of key/value pairs. The key may be a string (ntable), integer (itable) or reference (ftable). The 'value' part of the pair is always the refno of the element having that attribute value. e.g. internally a name table may look something like:

.....

Janet = 123/456

John = 321/543

Jonathon = 111/321

...

Thus for a given name, the corresponding element can rapidly be found without having to scan the entire MDB.

The keys in a name table must be unique. Integer and reference tables may contain the same key many times. e.g. internally a reference table may look like:

....

=123/456 =234/555

=123/456 =222/333

=123/456 =211/999

=123/458 =203/909

In the above table, element =123/456 has appeared in the table three times. This means that three different elements were all referencing =123/456 for this particular attribute. This is common. e.g. catalogue references are reference tables, and for these there may be hundreds of elements referencing the same catalogue component. A further point worth noting with respect to tables:

- Each DB has its own table. The entries in that table are for 'values' in that DB. Thus for the above ftable, =234/555 etc must be in this DB. However =123/456, =123/458 may be in a different DB. Indeed there may well be further references to =123/458 from other DBs. Hence to find all references to =123/456 we need to check the tables in each opened DB.

### 4.4.2 Table Classes

There are C# classes that enable direct access to the Dabacon tables. The classes are defined as follows:

NameTable - for iterating through name tables

RefTable - for iterating through reference tables

IntTable - for iterating through integer tables

These classes work on a single DB. In most cases, we really want to iterate through all DBs in the MDB as if it was one table. Thus for integer and name tables there are iterator classes that go across the whole MDB as follows:

MdbNameTable - for iterating through a name table across entire MDB

MdbIntTable - for iterating through an integer table across entire MDB

There is no method for iterating through a reference table since the order of references in a table is not meaningful.

There are also methods to return all entries for a given key. This is particularly useful for reference tables; these are methods of the MDB class.

In C# there is a NameTable class. An example of C# code that looks for all names starting with the letter 'B' is as follows:

```
string nam1=new string("/B");
string nam2=new string("/C");
NameTable ntable = new MdbNameTable(db, DBAttribute.NAME, nam1, nam2);
using (ntable)
{
    foreach (Element ele in ntable)
    {
        // Do something here with each element
    }
}
```

Freeing search tokens

Dabacon allocates search tokens when scanning tables. It has a limited number of search tokens. If you get to the end of the table then the token is freed automatically. However if you stop the search before you get to the end then you must explicitly free the token.

The IDispose method is used to free the underlying Dabacon token. Therefore ALWAYS scan the Dabacon table within a 'using' block to ensure that the Dispose method is called. If

the underlying token is not freed then errors will be output to the PDMS/Marine command line of the form:

```
"xx dabacon search tokens - Expected none "
```

The error generating code does not actually free the tokens. So typically you will see the number climb as more tokens are used up and not freed. The messages will be generated until the database is closed.

## 4.5 DBs, MDBs and Projects

### 4.5.1 MDB Functionality

The MDB class exposes methods that act on the currently opened MDB. This is a singleton class. It contains methods to

- open/close an MDB
- savework/quitwork
- query claims
- claim/release elements
- extract operations, e.g. flush, refresh, extract claim, extract release, extract drop.
- get opened DBs
- return an element(s) of a given name
- return elements with matching values in name and integer tables (as described in previous section)

Related pseudo attributes are:)

Attribute Name	Data Type	Description
OKDROP	bool	True if element may be dropped
OKRLEX	bool	True if element may be extract released

### 4.5.2 DB Functionality

When an MDB is opened, a DB class instance is created for each opened DB.

The functionality offered by this class falls into the following categories:

- Query the properties of the DB. e.g. number, MULTIWRITE or not
- Returning the world element for that DB
- Session information. e.g. when was the session written and by whom. There is a DbSession object that holds details on a database session.
- Return the DbElement in the system DB that represents this DB. This DbElement can then be used to query system attributes and additional pseudo attributes. The additional pseudo attributes available on the DbElement representing the DB are described below:

Attribute Name	Data Type	Qualifier	Description
ACCEDB	string		DB access
CLAIM	String		Implicit or explicit claims
CLCCNT	int		Claim list changes count

Attribute Name	Data Type	Qualifier	Description
CSESS	String	int	Comment entered for specified session
DACC	String		Database access
DBLC	DbElement[]		List of LOCs belonging to a DB
DCLA	String		Database claim mode
DSESS	String	int	Date of specified session
EXPIRY	String		Protected DB Expiry date
EXTALS	DbElement[]		Extract ancestors
EXTCLS	DbElement[]		Extract children
EXTDES	DbElement[]		Extract descendants
EXTFAM	DbElement[]		Extract family
FILENAME	String		DB filename
FOREIGN	string		Originating project
HCCNT	int		Header/Extract list changes count
ISDBFR	bool		True if database is foreign
ISDRDB	Bool		True if database has drawings
ISWORK	bool		True if a working extract
LINKSN	int		Linked session on parent extract
LPROT	bool		Protected DB Flag
MDBCLS	DbElement[]		List of MDBs in which DB is current
MDBLS	DbElement[]		List of MDBs to which DB belongs
NACCNT	int		Non-additive changes count
NAMEDB	string		DB name
NXTDBN	int		Next DB number
NXTEXN	int		Next DB extract number
NXTFDB	int		Next DB file and DB number
NXTFLN	int		Next DB file number
PSESS	int	int	Previous session number
SIZEDB	int		Size of DB
STPDBS	DbElement[]		Stamps containing db
STPSES	Int[]		Sessions used in at least one stamp
STYPDB	string		DB sub type
TYPEDB	string		DB type
USESS	string	int	User creating session

In addition for any DbElement in any DB, the following pseudo attributes provide information on the current DB:

Attribute Name	Data Type	Qualifier	Description
DBAC	String		DB Access, MULTIWRITE or UPDATE
DBCL	String		DB Claim
DBCNTL	Bool		True if element is in a control DB

DBEXNO	int		DB Extract Number
DBFI	String		DB filename
DBFN	int		DB file number
DBNA	String		DB name
DBNU	int		DB Number
DBREF	DbElement		Reference of DB element
DBTY	String		DB type (DESI, CATA etc)
DBVAR	Bool		True if element is in a variant DB
DBWRITE	Bool		True if element is in a writable DB
WDBNA	STRING		Working DB name

## 4.6 Events

### 4.6.1 Overview of Events

The database events fall into the following groups;

- General capture of Database changes
- Writing Pseudo attributes for UDAs
- DB/MDB related events

### 4.6.2 Overview of C# Mechanism

The generalised mechanism in C# is as follows:

- There is a delegate to define the event signature.
- There is a method to add a delegate to the list of event handlers to be called.

### 4.6.3 General Capture of DB Changes

DBlayer maintains a change list of Database changes. Users can subscribe to this list of changes.

The list of changes is encapsulated in the DbUserChanges class. The DbUserChanges class has methods to determine what changes have been made. E.g. elements created or deleted or modified.

The delegates are:

```
public delegate void ChangeEventHandler(DbUserChanges changes);
```

To add a subscriber (handler), the method is:

```
public static void AddChangeEventHandler(ChangeDelegate plug)
```

This will be in the DbEvents class.

The subscribers receive a DbUserChanges class instance which has a list of elements changed and what the changes are.

The events are fired as follows:

- At the end of running in a PML macro
- After the execution of a PDMS/Marine command on the command line
- When any action on a form has completed.

The event will only be fired if database changes have been made.

#### 4.6.4 Adding Pseudo Attribute Code

Code can be plugged in to calculate the value of pseudo attributes. The code to do this must be registered in PDMS/Marine by passing in a C# delegate.

Code can be registered in two ways:

1. Against a specific UDA. The code will then be invoked for all elements having this UDA.
2. Against a specific UDA and a specific element type. The code will only be invoked for all elements of that type.

The same UDA may have multiple delegates registered for different element types.

There is a different delegate for each attribute type. E.g. for integer attributes the delegate is:

```
public delegate double GetDoubleDelegate(DbElement ele, DbAttribute att,
int qualifier);
```

These are defined in the DbPseudoAttribute class.

The user must write a method that matches the method signature of the delegate. E.g. to write pseudo attribute code for a 'double' attribute, the user must write a method that has the signature defined by 'GetDoubleDelegate'. i.e. a method that takes a DbElement, a DbAttribute, int and returns a double.

E.g. the following method would be valid:

```
// Double delegate for UDA
static private double VolumeCalculation(DbElement ele, DbAttribute
att, int qualifier)
{
    // calculate the volume by multiplying the lengths along each side
    double x=ele.GetDouble(ATT.XLEN);
    double y=ele.GetDouble(ATT.YLEN);
    double z=ele.GetDouble(ATT.ZLEN);
    // Result of UDA must be returned
    return (x * y * z);
}
```

An instance of the delegate containing the method must then be created and registered with PDMS/Marine.

There are separate methods to register the different types of delegates. There are also separate methods to add a plugger for a particular element type. E.g. the two methods to add a GetDoubleDelegate are:

```
public static void AddGetDoubleAttribute(DbAttribute att, GetIntDelegate
plug)

public static void AddGetDoubleAttribute(DbAttribute att, DbElementType
type, GetIntDelegate plug)
```

An example of registering a delegate is:

```
using System;
using NOUN=Aveva.Pdms.Database.DbElementTypeInstance;
using Ps=Aveva.Pdms.Database.DbPseudoAttribute;
namespace Aveva.PDMS.Shared.Tests
```



```

{
    static public void RegisterDelegate()
    {
        // get uda attribute
        DbAttribute uda=DbAttribute.GetDbAttribute(":VOLUME");

        // Create instance of delegate containing "VolumeCalculation"
method
        Ps.GetDoubleDelegate dele=new Ps.GetDoubleDelegate(VolumeCal-
culation);
        // Pass delegate instance to core PDMS. This will be invoked
later
        // when :VOLUME is queried.
        // In this case registry for all valid element types.
        Ps.AddGetDoubleAttribute(uda,dele);
    }
}

```

Code may be plugged by UDET as well as the base type. The following criteria are used to locate the right plugged code:

1. If a UDET, look for a delegate plugged by UDET and attribute.
2. Look for a delegate plugged by base type and attribute
3. Look for a delegate plugged by matching attribute only.

e.g. you could add three delegates to calculate :WEIGHT. You could add one that calculates the :WEIGHT on a :MYELE, one that calculates the :WEIGHT of SCTN and one that calculates WEIGHT for any other element for which :WEIGHT is valid.

A delegate only needs adding once at start up.

The events do not allow for errors. Thus if the value can not be calculated then the pseudo attribute code should return a sensible default.

### 4.6.5 DB/MDB Related Events

The following events can be captured:

1. Committing of any cached changes to the database. Called once prior to a Savework, undo, quitwork, setMark, Getwork
2. Pre and post events for the following actions: Savework, Getwork, undo, Redo, drop, flush, quit, refresh. These are only called if the action succeeds, and are there to cope with the changes. The pre events should not make any database changes.
3. Failure event for Savework, flush
4. Clearing out any local caches.
5. Claim/Release events

Except for quit, the pre actions are really called post action but with the DB in the pre action state. The pre action is only called if the action succeeds.

Most of the pre and post actions fall into just two categories:

1. Changes made to the database file itself, but not our view of it (Savework, Flush, Refresh, drop).
2. Changes in our view of the database, but no changes to the database file (Getwork, undo, Redo, quit).

The interested parties are different in the two cases. When changing the database file, the interested parties are 3rd party systems outside of PDMS/Marine. When changing our view of the database it is the current PDMS/Marine session that needs to reflect the changes. E.g. update the 3D graphics and explorer.

The events raised reflect this split.

The events are:

- *CommitPending* - This is called to commit any outstanding change to the database. It is called prior to setmark, undo, redo, Savework, quit, Getwork.
- *ClearCache()* - This will be called after doing a temporary switch to a different session in a DB, or after doing a Getwork, undo, redo, quit.

**Note:** *ClearCache* may be called thousands of times within a single operation. Thus any event handlers must be fast.

- Pre and Post events for local changes. i.e. changes affecting the current view of the data. These will be Getwork, undo, redo.
- Pre and Post events for changes affecting the data on the DB. These will be Savework, refresh, drop, flush, refresh. These will pass the list of changes. For 'global' DBs, if the parent extract is at a different location, the flush event will be raised by the child extract.

**Note:** In this case the flush could still fail at the parent. Alternatively the daemon needs to raise the flush event.

- Post claim/release events
- Post failed Getwork/Savework

## 4.7 Units

Real dimensioned quantities may be created or returned as objects of type `DbDouble` which have a dimension, `DbDoubleDimension`, current units, `DbDoubleUnits`, and a real value which may be presented as a string using a format object, `DbFormat`. Standard dimensions and units are defined by the enumerated types `DbDimension` and `DbUnits` returned by `DbDoubleDimension` and `DbDoubleUnits` respectively. The `DbElement` Get/Set interface supports both undimensioned and dimensioned attributes which may be formatted using the `DbFormat` object. The following classes are available in the `Aveva.Pdms.Database` interface.

### 4.7.1 DbDoubleUnits

Represents a standard or compound unit constructed via static constructors. For example, construct a compound unit from an expression

```
DbDoubleUnits u1 = DbDoubleUnits.GetUnits("kg.m.s-2");
```

or construct a standard unit from a standard enumeration

```
DbDoubleUnits u1 = DbDoubleUnits.GetUnits(DbUnits.MM);
```

There is a static method to get all the defined standard and compound units

```
DbDoubleUnits[] units = DbDoubleUnits.AllUnits();
```

and properties to get the description, dimension, conversion factor etc. For example, get the dimension of given units

```
DbDimension d1 = DbDoubleUnits.GetUnits(DbUnits.MM).Dimension;
```

### 4.7.2 DbUnits

Enumeration of standard units. For example:

```
DbUnits.KG
```

### 4.7.3 DbDoubleDimension

Represents a standard or compound dimension constructed via static constructors. For example, construct a compound dimension from an expression

```
DbDoubleDimension d1 = DbDoubleDimension.GetDimension("volt/m");
```

or construct a standard dimension from a standard enumeration

```
DbDoubleDimension d1 =  
DbDoubleDimension.GetDimension(DbDimension.DIST);
```

There are properties to get the current units for given dimension, all units for given dimension etc. For example, get all the units of given dimension

```
DbDoubleDimension mass =  
DbDoubleDimension.GetDimension(DbDimension.MASS);  
DbDoubleUnits[] massUnits = mass.Units;
```

DbAttribute also has a property, Dimension, which returns the dimension of given attribute. For example,

```
DbAttribute udl = DbAttribute.GetDbAttribute(":UD1");  
DbDoubleDimension dimension = udl.Dimension;
```

### 4.7.4 DbDimension

Enumeration of standard dimensions. For example

```
DbDimension.MASS
```

### 4.7.5 DbDouble

Represents a real dimensioned quantity which can be constructed from an expression and/or a format or returned from DbElement. For example, create a DbDouble from a real value

```
DbDouble d1 = DbDouble.Create(1.0);
```

or create a DbDouble from a string expression

```
DbDouble d1 = DbDouble.Create("1.0kg");
```

or get a dimensioned attribute from DbElement as a DbDouble

```
DbElement bran = DbElement.GetElement("/100-B-1-B1");  
DbDouble hbor = bran.GetDbDouble(DbAttributeInstance.HBOR);
```

DbDouble's may also be used to set dimensioned attributes. For example, setting the HBOR of a branch

```
DbElement bran = DbElement.GetElement("/100-B-1-B1");  
DbDouble hbor = DbDouble.Create("300", fbore)  
bran.SetAttribute(DbAttributeInstance.HBOR, hbor);
```

Quantities of dimension BORE may be created as follows from a real value

```
DbDouble bore = DbDouble.CreateBore(300);
```

or a string expression and a format

```
DbFormat fbore = DbFormat.Create();
```

```
fbore.Dimension = DbDoubleDimension.GetDimension(DbDimension.DIST);
fbore.Units = DbDoubleUnits.GetUnits(DbUnits.FINC);
DbDouble bore = DbDouble.CreateBore("12", fbore);
```

returning the nearest nominal bore depending on the current BORE units. There are also properties to return the units, dimension etc. For example, get the units of given DbDouble

```
DbDouble d1 = DbDouble.Create("2m");
DbDoubleUnits u1 = d1.Units;
```

To return a formatted string, For example

```
DbFormat ftemp = DbFormat.Create();
ftemp.Dimension = DbDoubleDimension.GetDimension(DbDimension.TEMP);
ftemp.Units = DbDoubleUnits.GetUnits("degRankine");
ftemp.Label = "degRan";
DbDouble t1 = DbDouble.Create("3 celsius");
String tempStr = t1.ToString(ftemp);
```

and to convert to different units

```
DbDouble d1 = DbDouble.Create("1kg");
DbDouble d2 = d1.ConvertUnits(DbDoubleUnits.GetUnits(DbUnits.LB));
```

DbDouble also implements the expected binary operator overloads '+', '-', '\', '\*', '==', '!=', '<', '<=', '>' and '>=' allowing arithmetic operations and comparison of dimensioned quantities. For example, adding 2 quantities of the same dimension together

```
d1 = DbDouble.Create("1 kg");
d2 = DbDouble.Create("2.2046 lb");
d3 = d1 + d2;
```

#### 4.7.6 Exception Handling

Arithmetic errors or errors constructing DbDouble's will throw a PdmsException containing the error message. For example, adding 2 quantities of different dimensions together will throw a PdmsException which can be caught

```
d1 = DbDouble.Create("1 kg");
d2 = DbDouble.Create("2 mm");
try
{
    d3 = d1 + d2;
}
catch (PdmsException ex)
{
}
```

#### 4.7.7 DbFormat

This object allows formatting of DbDouble's as a string having properties to set the dimension, decimal points, label etc. For example, formatting a temperature

```
t1 = DbDouble.Create("0 celsius");
DbFormat tempFmt = DbFormat.Create();
tempFmt.Dimension =
DbDoubleDimension.GetDimension(DbDimension.TEMP);
tempFmt.Units = DbDoubleUnits.GetUnits(DbUnits.CELSIUS);
```

```
tempFmt.DecimalPoints = 1;
tempFmt.Label = "degC";
string tempStr = t1.ToString(tempFmt);
or constructing a DbDouble from a string and given format
DbFormat f1 = DbFormat.Create();
f1.Dimension = DbDoubleDimension.GetDimension(DbDimension.DIST);
f1.Units = DbDoubleUnits.GetUnits(DbUnits.INCH);
DbDouble d1 = DbDouble.Create("1", f1);
```



## 5 PMLNet

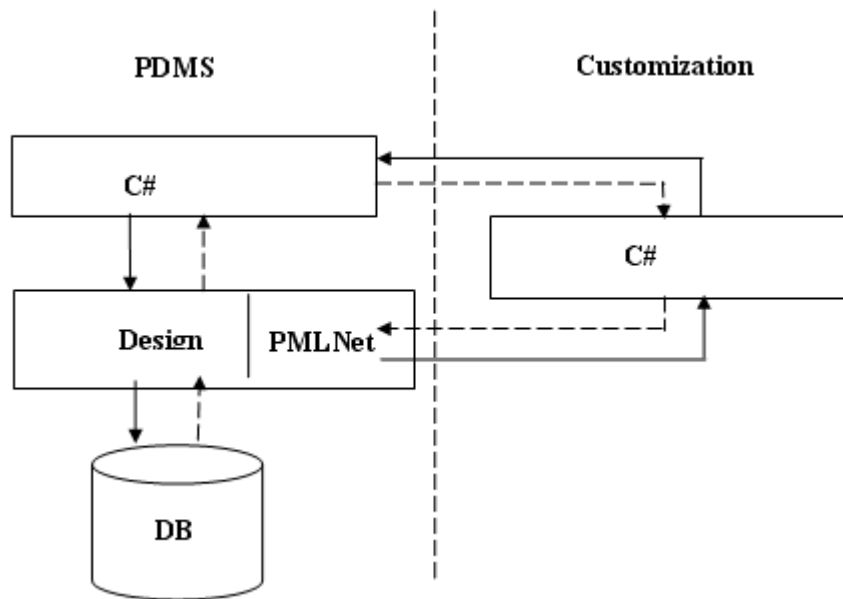
PMLNet allows you to instantiate and invoke methods on .NET objects from PML proxy objects. PML proxy class definitions are created from .NET class definitions at run time. These proxy classes present the same methods as the .NET class which are described using custom attributes. Proxy methods are passed arguments of known types which are marshalled to the corresponding method on to the .NET instance. The PML proxy objects behave just like any other PML object.

### 5.1 Design Details

PML callable assemblies are loaded by PDMS/Marine using the IMPORT syntax. Assemblies may be defined in potentially any .NET language, for example managed C++, C# or VB.NET. The PMLNet Engine loads a given assembly using reflection. The assembly may be located in the %PDMSEXE% directory, a subdirectory below %PDMSEXE% a mapped drive or on a UNC path. When the assembly is loaded PML class definitions are created for each PML callable class within the assembly. The PMLNet Engine only loads assemblies which are marked with the custom attribute PMLNetCallable. Only classes and methods which are marked as PMLNetCallable are considered. In order to create a valid PML Proxy class definition the .NET class and its methods must adhere to certain rules. Once an assembly has been loaded instances of PMLNetCallable classes may be created. No additional code to interface between PML and .NET is necessary. This is provided by the PMLNetEngine.

#### 5.1.1 Using PMLNet

The diagram below shows how PDMS/Marine may be customised using PMLNet. A number of .NET API's are available which allow access to the current database session, drawlist, geometry and other functionality. Users are able to write their own managed code which accesses PDMS/Marine via these C# API's. It is not possible to directly call PML from C#. However there is an event mechanism which allows PML to subscribe to events raised from C# (these are shown in dashed lines below). Events are also raised when the database changes and can be subscribed to from C# (also shown in dashed lines). In this example the external C# assemblies share the same database session as Design i.e. they run in the same process and therefore see the same data.



### Limitations

- Only .NET classes which are marked as PMLNetCallable and adhere to certain rules can be called from PML (these rules are described later)
- Module switching does not persist .NET objects. Core PML objects defined in FORTRAN or C++ are not persisted either.
- Passing other PML system/user objects to .NET e.g. DIRECTION, ORIENTATION, ... is not possible. It is possible to pass database references to .NET either as an array or String. It is also possible to pass an existing instance of a PML Net Proxy to .NET.
- It is not possible to call directly PML objects from .NET. The only way to call PML from .NET is via events.
- It is not possible to enter 'partial' namespaces as you might in C# and expect them to be concatenated.

E.g. the sequence:

```
USING NAMESPACE 'Aveva.PDMS'
!netobj = object namespace.NETObject ( )
```

will give an error.

### Objects and Namespaces

In order to identify and load a particular class definition the PML proxy class is assumed to have the same name as the .NET class (PML callable classes must be case independent). This class name is passed to the PMLNetEngine which creates the .NET instance e.g.

```
!a = object netobject()
```

creates an instance of the .NET class netobject. To specify in which assembly this class is defined and resolve any name clashes the user needs to also specify the namespace in which the .NET class is defined using the following syntax:

```
USING NAMESPACE <string>
```

Where <string> is the namespace in which the class is defined.



e.g. 'AVEVA.PDMS.PMLNetExample'

The namespace is considered to be case independent. This namespace will remain current until it goes out of scope (e.g. at end of macro). When the user types:

```
!netobj = object NetObject ( )
```

then all namespaces in the current scope will be searched to find a match. In this example, if 'Aveva.PDMS. PMLNetExample' is not currently in scope, then the error:

*(46,87) PML: Object definition for NETOBJECT could not be found.*

will be raised.

## Object Names

Object names can consist of any alpha but not numeric characters (this restriction is imposed by PML). They are treated as case-independent. However, it is no longer necessary to define them in upper case - any mixture of upper and lower case letters will have the same effect.

## Query Methods

The query methods on an object have been enhanced as follows:

(a) Querying an object will show the namespace name as well as the object name:

e.g.

```
q var !x
```

```
< AVEVA.PDMS.NAMESPACE.NETOBJECT>
```

```
AVEVA.PDMS.NAMESPACE.NETOBJECT
```

(b) There is a new query method to list all the methods of an object (including constructors)

```
Q METH/ODS
```

e.g.

```
q meth !x
```

```
<AVEVA.PDMS.NAMESPACE.NETOBJECT>AVEVA.PDMS.NAMESPACE.NETOBJECT
```

```
NETOBJECT ( )
```

```
NETOBJECT (REAL)
```

```
ADD (REAL)
```

```
REMOVE (REAL)
```

```
ASSIGN(AVEVA.PDMS.NAMESPACE.NETOBJECT)
```

```
DOSOMETHING(REAL, REAL, REAL)
```

Note that query methods will not list the methods on objects of type ANY, even though such methods are available on all objects.

(c) A new query:

```
Q NAMESP/ACES
```

lists the namespaces currently in scope.

## Global Method

There is a new global method on all objects:

```
.methods()
```

which returns a list of the methods on the object as an array of strings.

e.g.

```
!arr = !x.methods()
q var !x
```

returns:

```
<ARRAY>
[1] <STRING> 'NETOBJECT ( )'
[2] <STRING> 'NETOBJECT (REAL)'
[3] <STRING> 'ADD (REAL)'
[4] <STRING> 'REMOVE(REAL)'
[5] <STRING> 'ASSIGN(AVEVA.PDMS.NAMESPACE.NETOBJECT)'
[6] <STRING> 'DOSOMETHING(REAL, REAL, REAL)'
```

Importing an assembly

Before an instance of a .NET object can be instantiated the assembly containing the class definition must be loaded. This is done using the IMPORT syntax as follows

```
IMPORT <string>
```

Where <string> is the case-independent name of the assembly

## Method Arguments

Only PML variables of the following types may be passed to methods on .NET classes. In the table below the PML variable type is in the left column and the .NET equivalent variable type is in the right column. Data is marshalled in both directions between PML and .NET by the PMLNetEngine.

PML	.NET
REAL	double
STRING	string
BOOLEAN	bool
ARRAY (these can be sparse and multi-dimensional)	Hashtable
OBJECT Any existing PML Net instance	PMLNetCallable class

Arguments to PML Proxy methods are passed by reference so can be in/out parameters (in .NET output arguments must be passed by reference).

### Value Semantics

PML Gadgets and DB elements have reference semantics when they are copied whereas all other objects have value semantics when they are copied. This is controlled by the Assign() method on the .NET class. So, for example if the Assign() method here copies the value then

```
!a = object netobject()
!a.val(1)
!b = !a
!b.val(2)
then
q var !a.val() returns 1
and
q var !b.val() returns 2
```

i.e. !a and !b **do not** point to the same object.

In order to perform either a shallow or deep copy of the member data inside the .NET class the Assign() method must be defined on the Net class (see rules). This is analogous to overriding the operator "=" in C++.

### Method Overloading

Overloading of methods is supported for all variable types in PML so a .NET Proxy can be created from a .NET class which has overloaded methods.

### Custom Attributes

The custom attribute [PMLNetCallable()] is used to describe the PML interface for a .NET class. This metadata allows the PML callable assemblies to be self-describing. This clearly defines the class and allows an assembly to expose a subset of its public interface to PML. The PMLNetEngine uses this metadata to decide which .NET class definitions can be created in PML. Reflection is used to load an assembly and create PML class definitions. All classes and methods for which PML Proxy class definitions will be created must be marked as PMLNetCallable. The assembly itself must also be marked as PMLNetCallable.

So, a PML callable .NET class in C# looks like this:

```
[PMLNetCallable()]
namespace PMLNet
{
    [PMLNetCallable()]
    public class PMLNetExample
    {
        [PMLNetCallable()]
        public PMLNetExample()
        {
        }

        [PMLNetCallable()]
        public void DoSomething(double x, double y, double z)
        {
        }
    }
}
```

```

        z = x + y;
    }
}

```

This class has a default constructor and a single method. Both the constructor and method are marked as PMLNetCallable along with the class itself.

The assembly itself must also be marked as PMLNetCallable. This is normally done in the AssemblyInfo file as follows

```
using Aveva.PDMS.PMLNet;
```

```
[assembly: PMLNetCallable()]
```

### Private Data and Properties

In PML there is no concept of private members or methods - everything is public. Access to public data in .NET must be via properties or get/set accessor methods. Properties in .NET class are defined as get and set methods in PML. So, for example the following PMLNetCallable property in .NET

```

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

```

would have the following Proxy methods in PML

```

REAL VAL()
VAL(REAL)

```

### Scope

PML variables are of two kinds: global and local. Global variables last for a whole session (or until you delete them). A local variable can be used only from within one PML function or macro. The lifetime of the .NET instance is controlled by the scope of the PML proxy.

### Instantiation

Classes can have any number of overloaded constructors but must have a default constructor which is marked as PMLNetCallable. The PML Proxy constructor instantiates an instance of the underlying .NET class. When the proxy goes out of scope the destructor destroys the underlying .NET instance.

### ToString() Method

The string() method is available on all PML objects. For a .NET Proxy this will call the ToString() method on the .NET instance. If the ToString() method is overridden on the .NET class then this will be called.

### Method Names

PML is case independent, so it is not possible to have MyMethod() and MYMETHOD() in .NET. PML will report non-unique object/method names.

### Double Precision

Doubles are used in PMLNet to store reals and ints so doubles must be used in .NET (integers are not available in PML)

### Events

Events on PMLNet objects may be subscribed to from PML. A PML callback on a particular instance may be added to an event on another PMLNet instance. Events are defined by a .NET component by associating the delegate PMLNetEventHandler with the event. This delegate has the signature

```
__delegate void PMLNetEventHandler(ArrayList __gc *args);
```

Where args is an array of event arguments of any PMLNet type (see table of valid types). The following code associates this delegate with an event

```
[PMLNetCallable()]
public class PMLNetExample
{
    [PMLNetCallable()]
    public event PMLNetDelegate.PMLNetEventHandler PMLNetExampleEvent;

    [PMLNetCallable()]
    public PMLNetExample ()
    {
    }

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

    [PMLNetCallable()]
    public void RaiseExampleEvent()
    {
        ArrayList args = new ArrayList();
        args.Add("PMLNetExampleEvent ");
        args.Add("A");
        if (PMLNetExampleEvent != null)
            PMLNetExampleEvent(args);
    }
}
```

This event can then be caught in PML by adding an appropriate callback to the instance raising the event

```
!n = object pmlnetexample()
!c = object netcallback()

!handle = !n.addeventhandler('pmlnetexampleevent', !c, 'callback')
```

Where

- !n is the PMLNet instance on which the event will be raised
- !c is the instance of a PML object with a method callback() with the appropriate arguments

At some later time the event handler may be removed

```
!n.removeeventhandler('pmlnetexampleevent', !handle)
```

where !handle is the handle of the PMLNet delegate returned by addeventhandler().

Netcallback is a PML object defined as

```
define method .callback(!array is ARRAY)
    !args = 'NETCALLBACK object ' + !array[0] + !array[1]
    $P $!args
endmethod
```

## Error Handling

Exception handling is placed around the Invoke method to handle .NET method invocation exceptions like TargetException, ArgumentException etc. The result of catching such an exception is to ultimately return a PMLException object from PMLNetProxy::Invoke() which results in a PML exception (1000,n) being thrown where 1000 is the module number for PMLNet. .NET can throw its own PML exceptions. The exception to throw is PMLNetException. For example

```
throw new PMLNetException(1000, 1, "PMLNetExample Exception");
```

This may then be handled inside a PML macro i.e.

```
handle(1000,1)
...
endhandle
```

Any other exception within the loaded assembly itself is caught by the global exception handler inside PDMS/Marine.

## Rules for Calling .NET

Certain rules must be adhered to when defining a .NET class which is PML callable. These are enforced by the PMLNetEngine when an assembly is imported. They are

- PML callable assemblies must be marked as PMLNetCallable and reside in the %PDMSEXEC% directory, subdirectory of the application or UNC path.
- Only classes may be PML Callable (this excludes Structures, Interfaces, Enums, ...).
- A PML callable class must be marked as PMLNetCallable.
- A PML callable method must be marked as PMLNetCallable.
- A PML callable method can only pass valid argument types (see table of types).
- PML callable classes and methods must be public.
- PML callable methods with default arguments cannot be defined.
- PML callable class and method names must be case independent.
- PML callable classes must have an Assign() method.
- PML callable classes must have a public default constructor which is PMLNetCallable.

If these rules are not adhered to then errors are reported to the trace log when the assembly is loaded and a PML class definition will not be created. If the class definition has not been defined then the following PML error will result

(46,87) PML: Object definition for XXX could not be found.

## Tracing

In order to output trace to a log file and the console window add the following lines to the exe's config file

```
<system.diagnostics>
  <switches>
    <add name="PMLNetTraceSwitch" value="4" />
  </switches>
</system.diagnostics>
<appSettings>
  <add key="PMLNetTraceLog" value="C:\temp\PMLNetTrace.log" />
</appSettings>
```

This will create the file PMLNetTrace.log in C:\temp and log all the valid class definitions as they are imported.

## 5.1.2 .NET Controls

.NET controls can be hosted on a PML form. In order to do this PML provides a container gadget which can host the control. This container gadget has attributes to set and get its size and position and may be added to any PML defined form. It has similar behaviour to a Frame gadget in terms of docking, anchoring and positioning within an owning form. An instance of the .NET control is instantiated from PML. The PML container provides a method to add an instance of a .NET control to it. The .NET control may raise events which may be handled by PML. In order to customise the context menus of the .NET control from PML the ability to define a PML menu which can be shown when the .NET control raises an event is provided.

### Creating a Container

A container on a form that can host the .NET control can be created in the following way in the form setup

```
container .exampleContainer PmlNetControl 'example' dock fill width 30
height 20
```

which can be docked and positioned.

### Hosting .NET Control

The control may be added to the container by setting the container's control to the .NET control's handle.

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

```
!this.exampleControl = object PMLNetExampleControl()
!this.exampleContainer.control = !this.exampleControl.handle()
```

### Events

Events on the control are supported by PML delegates already described. These allow you to add popup menus to the control for example. Events may be subscribed to by adding an event handler as follows to the .NET control

```
!this. exampleControl.addeventhandler('OnPopup', !this,
'rightClickGrid')
```

where the method to call when the event is fired is defined as follows

```
define method .rightClickGrid(!data is ARRAY)
    !this.exampleContainer.popup = !this.examplePopup
    !this.exampleContainer.showPopup(!data[0], !data[1])
endmethod
```

and the menu shown by the method which is added to the container is defined as follows

```
menu .examplePopup popup
    !this.examplePopup.add( 'CALLBACK', 'Add to 3D View',
        '!this.addToThreeDView()' )
```

### 5.1.3 Examples

The following examples are available in the Samples directory -

#### **PMLGridExample**

Example of a .NET grid hosted on a PML form

#### **PMLNetExample**

Example of a PML Callable assembly



## 6 AVEVA C# Grid Control

This section describes how to use an instance of an AVEVA C# Grid Control on a C# form. The example which is used in this documentation is supplied with the product. The Visual Studio project containing the source code is located in the following directory:

Samples\NetGridExample

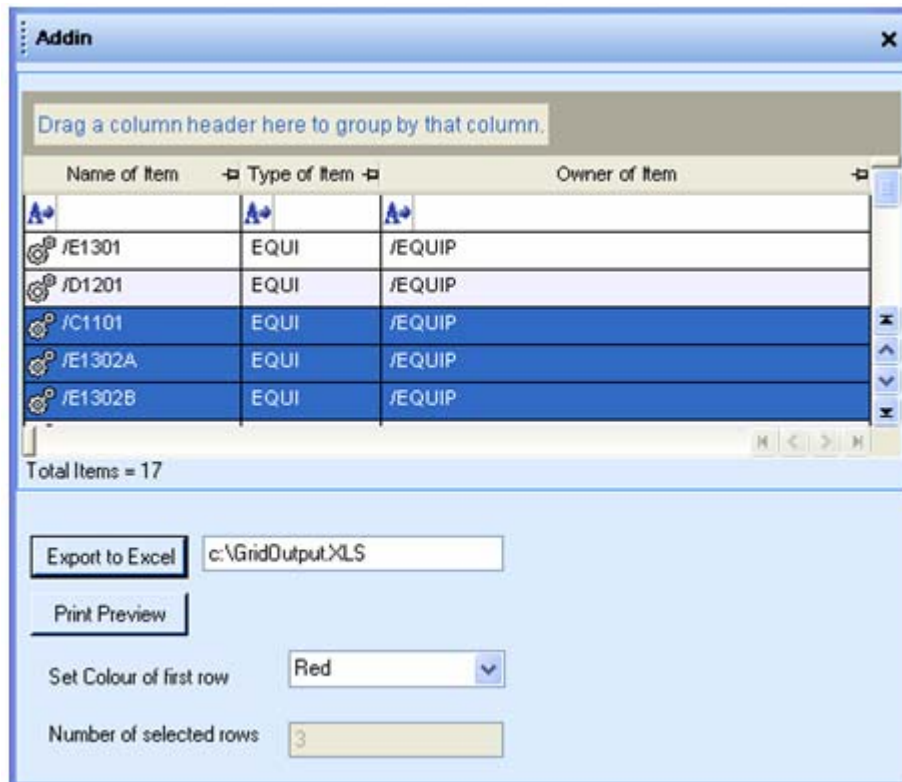
**Note:** That you can only use the AVEVA C# Grid Control on your own Visual Studio form if you have a design time license for Infragistics.

**Note:** That you can run this C# Addin with PDMS/Marine without having an Infragistics license.

### 6.1 Create a C# Addin which Contains an AVEVA Grid Control

The supplied C# project is an example of a C# addin for use with PDMS/Marine. The C# addin includes an AVEVA C# Grid Control and some other Visual Studio WinForms gadgets to demonstrate how the Grid Control can be used with PDMS/Marine.

The addin creates the docked window shown below when run within the PDMS/Marine Design module.



The example C# code collects all the Equipment items within the current project and lists them inside the grid along with attribute values for Name, Type, and Owner.

The data in the grid can be selected, sorted and filtered.

**Note:** The following features of the example addin. These features make use of the published Grid Control API (see the section below entitled "AVEVA Grid Control API").

1. The data in the grid can be exported to a Microsoft Excel file (XLS) by entering a path name in the text box beneath the grid and by clicking on the "Export to Excel" button.
2. The grid content can be previewed and printed by clicking on the "Print Preview" button.
3. The colour of the first row in the grid can be set either by choosing a colour in the drop down list, or by typing a valid colour into it.
4. The number of selected rows in the grid is written to a read-only text box. Note that this feature makes use of an event on the grid. (See the section below entitled "Adding an event to the C# Grid Control").
5. Notice that there are two different context menus available on the grid. One is available when equipment items are selected:

Name of Item	Type of Item	Owner of Item
/ELEC01-NOZZLE	EQUI	/ELECT
/DRNACC1	EQUI	/UNDERGROUND-EQUIP
/DRNACC2	EQUI	/UNDERGROUND-EQUIP
/DRNACC3	EQUI	/UNDERGROUND-EQUIP
/DRNACC4	EQUI	/UNDERGROUND-EQUIP

And the other menu is available from the header bar at the top of the grid:

Name of Item	Type of Item	Owner of Item
/ELEC01-NOZZLE	EQUI	/ELECT

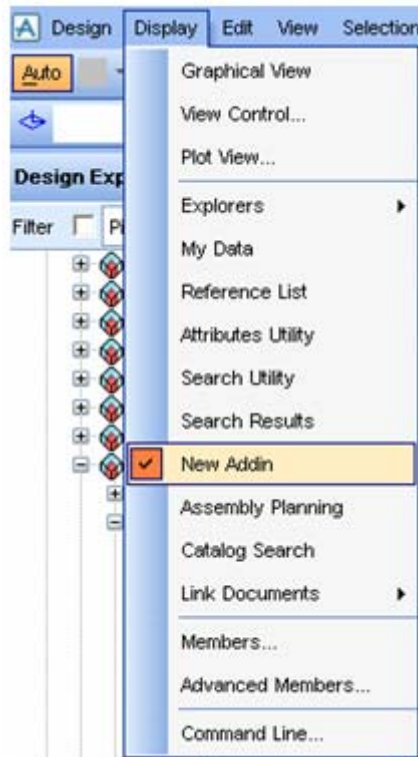
**Note:** That these menu options will only become available once you have made some modifications to the Design.uic file through menu customisation. See below for instructions on how to do this.

## 6.2 Provide Access to the Addin in Design

In order to see the addin inside Design you will need to do the following:

1. Build the addin.DLL inside the Microsoft Visual Studio project.
2. The majority of the menus and toolbars in these two AVEVA modules are still currently defined in PML. If you wish access to your addin functionality to be provided from a additional entry on an existing menu then it is necessary to add the following line shown in red to the PML form file appdesmain.pmlfrm which resides within the directory: "PMLLIB\design\forms". The other lines are shown to help you locate where to add the PML menu. This will add a PML menu option beneath the Display menu in Design, and will allow you to open the addin.

```
!menu.add('TOGGLE', 'My Data', '', 'My Data')
!menu.add('TOGGLE', 'Reference List', '', 'Reference List')
!menu.add('TOGGLE', 'Attributes Utility', '', 'Attributes Utility')
!menu.add('TOGGLE', 'Search Utility', '', 'Find Utility')
!menu.add('TOGGLE', 'Search Results', '', 'Output Utility')
!menu.add('TOGGLE', 'New Addin', '', 'New Addin')
```



Note that you could also create your own toolbar which contained a menu item to open/close the addin.

## 6.3 Use the AVEVA Grid Control with Different Data Sources

The supplied C# addin populates the grid with database items and their attribute values.

The following data sources are available for this method of working. See the AVEVA Grid Control API section below for further information.

- NetDataSource(String TableName, Array Attributes, Array Items)
- NetDataSource(String TableName, Array Attributes, Array AttributeTitles, Array Items)
- NetDataSource(String TableName, Array Attributes, Array Items, String Tooltips)

There are two other ways that an instance of the grid can be populated with data:

1. Populate the grid with non-database data

The grid can be populated with collections of non-database data.

The following data sources are available for this method of working. See the AVEVA Grid Control API section below for further information.

- NetDataSource(String TableName, Array columns, Array of Array of rows)
  - NetDataSource(String TableName, Array columns, Array of Array of rows, String Tooltips)
2. Populate the grid with non-database data direct from a Microsoft Excel file.

The grid can be populated with non-database data from a Microsoft Excel file. The first row of the Microsoft Excel file will supply the headings for the grid and the other rows will supply the row data.

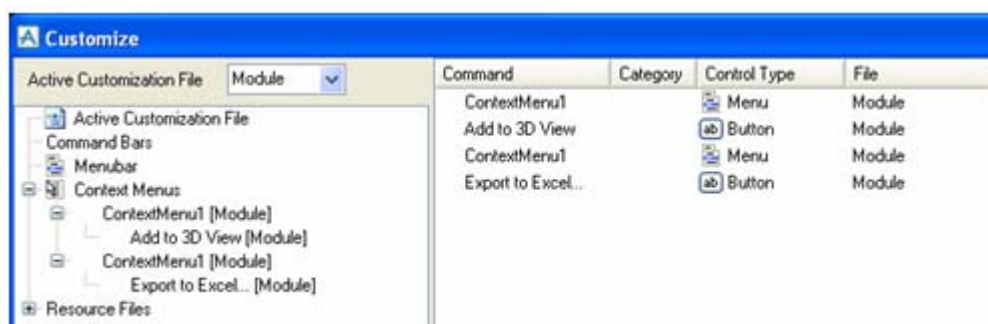
The following data source is available for this method of working. See the AVEVA Grid Control API below for further information.

- NetDataSource(String TableName, string PathName)

## 6.4 Add an XML Menu to the Form

Code has been added to the supplied addin to allow two different context menus to be used with the grid: a selection menu, and a header menu.

To enable this to work with your session you will need to use the Menu Customisation utility described earlier in this document.



To add the menus you will need to do the following in the Menu Customisation utility:

1. Add a new Context Menu, which has its name property set to: NewAddin.SelectMenu. This name is used by the code to locate the set of menus to display when the user makes a context menu selection on one or more equipment items in the grid. The C# method below from the addin class, is the code which loads the menu. The menus which you create in the Menu Customisation Utility will be stored in the Design.uic file.

AllowTearaway	False
AutoUpdate	False
Caption	<b>ContextMenu1</b>
Category	
Command	
DropDownArrowStyle	Default
Icon	<input type="checkbox"/> No Image or resource id specified
IsContextMenu	<b>True</b>
Name	<b>NewAddin.SelectMenu</b>
OptionSetKey	
PopupStyle	Menu
Shortcut	None
Tooltip	

```
private void mCommandBarManager_UILoaded(object sender, EventArgs e)
{
    String contextMenuKey = "NewAddin.SelectMenu";
    if(mCommandBarManager.RootTools.Contains(contextMenuKey))
```

```

        {
            MenuTool menu = (MenuTool)mCommandBarManager.Root-
Tools[contextMenuKey];
            mAddinControl.SelectionMenu = menu;
        }

        contextMenuKey = "NewAddin.HeaderMenu";
        if(mCommandBarManager.RootTools.Contains(contextMenuKey))
        {
            MenuTool menu = (MenuTool)mCommandBarManager.Root-
Tools[contextMenuKey];
            mAddinControl.HeaderMenu = menu;
        }
    }

```

2. Create a button and set the command to AVEVA.DrawList.Add. When this menu is used in the Addin it will add the selected equipment item(s) to the 3D view.
3. Assign the button to the context menu named NewAddin.SelectMenu. You can create other menu options (either from existing commands which have been exposed, or from existing PML methods and functions) in the same way and assign these to the context menu.
4. Create a second Context menu named NewAddin.HeaderMenu. This name is used by the code to locate the set of menus to display when the user makes a context menu selection on the header bar in the grid.
5. Create a button and set the command to AVEVA.Grid.ExportToExcel. When this menu is used in the Addin it will export the grid data to a Microsoft Excel file.
6. Assign the button to the context menu named NewAddin.HeaderMenu. You can create other menu options in the same way and assign these to the context menu.

## 6.5 Add an Event to the Addin

There are several events which have been exposed for the AVEVA Grid Control. These are:

### AfterCellUpdate

No Arguments

Event to alert the user that a cell has been updated.

### AfterCellPosChanged

No Arguments

Event to alert the user when a column is dragged to a new position, resized, or un/fixed.

### AfterCellSelectChange

No Arguments

Event to alert the user when the cell selection is changed.

### AfterRowsDeleted

No Arguments

### AfterRowFilterChanged(Array)

- Array[0] is the tag of the column in which the filter was changed.  
Event to alert the user when a column filter is changed.

### AfterRowInsert

No Arguments

Event to alert the user that a row has been inserted.

### AfterRowUpdate

No Arguments

Event to alert the user that a row has been updated.

### AfterSelectChange (Array)

- Array[0] contains the ID of each of the selected rows

An array of selected items is passed through to the calling application. See the example C# code snippet below. The example code shows that the number of selected items is written to a text box.

```
private void netGridControl1_AfterSelectChange(System.Collections.ArrayLst
args)
{
    //Print the number of the selected rows in textbox2
    if (args == null)
    {
        return;
    }
    Hashtable al = new Hashtable();
    al = (Hashtable)args[0];
    if (al == null)
    {
        return;
    }
    this.textBox2.Text = al.Count.ToString();
}
```

Refer to [Events](#) for Event handling.

### BeforeCellActivate (Array)

- Array[0] is the Row Tag of the cell
- Array[1] is the Column Tag of the cell

### BeforeCellUpdate (Array)

- Array[0] is the new value the user typed in
- Array[1] is the Row Tag of the cell
- Array[2] is the Column Tag of the cell
- Array[3] is a Boolean. If you set it to "false" then the action is cancelled
- Array[4] is a string. If you set Array[3] to "false" then this value can be set to give the reason why the new value cannot be allowed. The message will be displayed in the cell tooltip.

Cell data is passed back to the calling application as an array before a change is made in a cell of the grid (This assumes that the grid is in editable mode or bulk editable mode). The fourth argument of the array (Array[3]) can be set to false by the calling application in order to disallow the new value of the cell.

**Note:** This event is the opportunity for the calling code to make a synchronising change to any related data source. In the case where the related data is Dabacon element/attribute data, the BeforeCellUpdate event should arrange for the Dabacon element/attribute to be modified appropriately. A convenient way to do this is to use the NetGridControl.DoDabaconCellUpdate(Array) function to perform the modification. Simply pass the Array to this function as an argument. If the function is unable to perform the modification for any reason the Array[3] and Array[4] values will be set to indicate the problem.

#### **BeforeRowsDeleted (Array)**

- Array[0] the rows to be deleted
- Array[1] is a Boolean. If you set it to "false" then the action is cancelled
- Array[2] is a string. If you set Array[1] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

#### **BeforeRowInsert (Array)**

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

#### **BeforeRowUpdate(Array)**

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

#### **CellChange**

- Array[0] is the tag of the row
- Array[1] is a tag of the column

Event to alert the user when the value of a cell is changed.

#### **ColumnsChanged**

This event is fired when the column layout is changed.

#### **ColumnsInGroupByMode (Array)**

- Array[0] is a Boolean. It is true if any columns are in the group by mode.

#### **CopyKeyPressed (Array)**

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.



### CutKeyPressed (Array)

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

### DeleteKeyPressed (Array)

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

### LayoutChanged

This event is fired when the grid layout is changed.

### OnPopup (Array)

- Array[0] is the x coordinate of the current cursor position
- Array[1] is the y coordinate of the current cursor position
- Array[2] contains the ID of each of the selected rows

The position of the mousedown, and an array of selected items is passed through to the calling application when the context menu is used on the grid.

### OnDragDropCopy (Array)

- Array[0] is the row tag of the copied row
- Array[1] is the drop index
- Array[2] is a Boolean. If you set it to "false" then the action is cancelled
- Array[3] is a string. If you set Array[2] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

### OnDragDropMove (Array)

- Array[0] is the row tag of the copied row
- Array[1] is the drop index
- Array[2] is a Boolean. If you set it to "false" then the action is cancelled
- Array[3] is a string. If you set Array[2] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

### PasteKeyPressed (Array)

- Array[0] is a Boolean. If you set it to "false" then the action is cancelled
- Array[1] is a string. If you set Array[0] to "false" then this value can be set to give the reason why the action has been cancelled. The message will be displayed in the cell tooltip.

## 6.6 Other Functionality Available within the Environment

Note the following additional functionality available with the Addin:

1. Selected equipment items can be dragged and dropped from the grid into the 3D View, MyData, or any other form which is a drop target. This functionality is automatically available to the Addin as part of the Common Application Framework.
2. The Grid Control can be made editable, but any data which is changed in any cell of the grid will not be written back to the database. The calling application would need to handle the changes, and write this to the database, if relevant.
3. The Addin form can be docked and floated within the environment.

## 6.7 Use of the C# Grid Control with PML

The C# Grid Control can also be used on a PML form. See the PMLGridExample code in the Samples directory.

## 6.8 AVEVA Grid Control API

The following table includes the API calls which have been made available for the Grid Control.

**Note:** That you can only use the AVEVA C# Grid Control on your own Visual Studio form if you have a design time license for Infragistics.

Name	Type	Purpose
<b>Data Source methods for populating the grid</b>		
BindToDataSource(NetDataSource)		Bind Grid to NetDataSource
NetDataSource(String TableName, Array Attributes, Array AttributeTitles, Array Elements)		Data Source constructor for database Attributes and database elements. The Grid will populate itself with database attribute values. The title of each column is represented by AttributeTitles.
NetDataSource(String TableName, Array Attributes, Array Elements)		Data Source constructor for database Attributes and database elements. The Grid will populate itself with database attribute values.
NetDataSource(String TableName, Array Attributes, Array Elements, String Tooltips)		Data Source constructor for database Attributes and database elements. The Grid will populate itself with database attribute values.
NetDataSource(String TableName, Array columns, Array of Array of rows)		Data Source constructor for column headings, and a set of row data. In this case the grid does NOT populate itself with database attribute values.
NetDataSource(String TableName, Array columns, Array of Array of rows, String Tooltips)		Data Source constructor for column headings, and a set of row data. In this case the grid does NOT populate itself with database attribute values.

Name	Type	Purpose
NetDataSource(String TableName, string PathName)		Data Source constructor for import of an Excel XLS or CSV File
<b>General methods available for the grid</b>		
allGridEvents(Boolean)		Switch Grid Events On or Off.
AllowDropInGrid(Boolean)		Allow drag/drop in the grid.
BulkEditableGrid(Boolean)		Allow/disallow cells to be bulk editable. Bulk edit mode allows multiple cells to be selected, and Fill down/Fill up operations to be performed. Copy and Paste operations will also apply to multiple cells selected in the same column.
ClearGrid()		Remove data & column headings.
ClearGridData()		Remove all of the row data from the grid.
ColumnExcelFilter(Boolean)		Allow/disallow Excel style filter on columns
ColumnSummaries(Boolean)		Allow/disallow Average, Count, etc on numeric columns
CreateValueList(String Name, Array for List)		Create a value list for the grid.
DisplayNulrefAs(String)		Nominate (or query) a text string to be used in grid cells that hold "Nulref" reference attribute data.
DisplayUnsetAs(String)		Nominate (or query) a text string to be used in grid cells that hold "unset" attribute data.
EditableGrid(Boolean)		Allow/disallow cells to be editable.  <b>Note:</b> If the user is allowed to write data to the grid, then this data does not automatically get written back to the data source (or database). The BeforeCellUpdate event of the calling code is responsible for doing this synchronisation, and is in control of whether to either allow or disallow the change (Refer to <a href="#">Add an Event to the Addin</a> for more detail). This does not enable the user to add/remove rows, just to edit the content of the existing cells.
ErrorIcon(Boolean)		Allow/disallow the display of the red icon in the cell if an attribute value or expression cannot be computed.

Name	Type	Purpose
FeedbackFailColor(String)		Nominate (or query) the colour used to highlight cells where editing fails.
FeedbackSuccessColor(String)		Nominate (or query) the colour used to highlight cells successfully edited.
FixedHeaders(Boolean)		Allow/disallow fixed headers. (Useful when scrolling).
FixedRows(Boolean)		Allow/disallow fixing of rows (Useful when scrolling).
getDataSource()	NetDataSource	Returns the data source that has been set in the grid
GridHeight()	Real	Query the height of the rows in the grid.
GridHeight(Real)		Set the height of the rows in the grid.
HeaderSort(Boolean)		Allow/disallow the user to sort columns.
HideGroupByBox(Boolean)		Hide/show the groupBy box.
loadLayoutFromXml(String)		Load a stored grid layout into the current grid instance.
OutlookGroupStyle(Boolean)		Allow/disallow the Outlook Group style feature.
PrintPreview()		Opens a Print Preview of the grid instance (which also allows printing).
ReadOnlyCellColor(String)		Nominate (or query) the background colour used to indicate cells where editing is not permitted.
RefreshTable()		For a grid displaying Dabacon data this recalculates the content of every cell to refresh to the latest database state.
RefreshVisibleRegion()		For a grid displaying Dabacon data this recalculates the content of every cell in the visible region to refresh to the latest database state.
resetCellFeedback()		Reset the cell editing feedback highlight colour and tooltip text.
ResetColumnProperties()		Reset column layout.
saveGridToExcel(string excelFile,string worksheet, string strHeader)		Save the grid data to a designated worksheet of an Excel file. Here strHeader is a user supplied string which is output to the first row in the Excel spreadsheet. The grid data is then output starting in row #2.
SaveGridToExcel(string excelFile)		Save the grid data to an Excel file. Note that this will retain all grid groupings and layout of data. The String should specify the full pathname, eg: 'C:\temp\file.xls').

Name	Type	Purpose
SaveGridToExcel(string excelFile, string worksheet)		Save the grid data to a designated worksheet of an Excel file. Note that this will retain all grid groupings and layout of data.
SaveLayoutToXml(String)		Save the grid layout (not data) to a file on the file system.
setAlternateRowColor(Red Num, Green Num, Blue Num)		Set alternate rows of the grid to a different colour. The default is: (251, 251, 255).
setAlternateRowColor(String)		Set alternate rows of the grid to a different colour.
setLabelVisibility(Boolean)		Show/hide the label which indicates the number of rows in the grid.
SingleRowSelection(Boolean)		Set grid to single row selection.
SplitGrid(Boolean)		Allow/disallow the user to split the grid horizontally and vertically.
SyntaxErrorColor(String)		Nominate (or query) the colour used to highlight cells containing syntactically invalid data.
<b>Methods on rows in the grid</b>		
addRow(Array)		Add a single row of data. If the grid is based on database elements then the first element of the array is read. If not, then the Array data represents the data in each cell of the row.
clearRowSelection()		Clear row selection.
deleteRows(Array)		Delete one or more rows by row tag.
deleteSelectedRows()		Delete selected rows.
GetFilteredInRows()	Array	Return the rows which are contained in the current filter.
GetFilteredOutRows()	Array	Return the rows which are not contained in the current filter.
getNumberRows()	Real	Get the number of rows in the grid.
GetRow(string row)	Array	Get the cells in the row with the specified row tag.
GetRows()	Array of Array (rows)	Returns an array of all the row data.
GetSelectedRows()	Array of Array (rows)	Returns array of selected rows.
GetSelectedRowTags()	Array of row tags	Get the selected row tags.
MoveSelectedRowsDown()		Move the selected rows down in the grid.
MoveSelectedRowsUp()		Move the selected rows up in the grid.

Name	Type	Purpose
RowAddDeleteGrid(Boolean)		Enable/disable row addition and deletion in the grid.
scrollSelectedRowToView(Boolean)		Scroll the selected row into view.
SelectAllRows()		Select all the rows in the grid.
SetRowColor(string rowTag, string colour)		Set the colour of the row to the named value.
SetRowColor(string row, real red, real green, real blue)		Set the colour of the row to the RGB value.
setRowTooltip(string row, string tooltip)		Set row tooltip to the specified string.
setSelectedRowTags(Array)		Programmatically select the given row tags.
<b>Methods on columns in the grid</b>		
AutoFitColumns()		Resize columns to fit widest text.
ColumnsInGroupBox()	Boolean	Returns true if there is at least one column in the GroupBy box.
ExtendLastColumn(Bool)		Extend the last column.
FixedHeaderOnColumns(string column, Boolean)		Fix the header on the column.
GetColumn(string column)	Array	Get the cells in the column with a specified tag.
GetColumnPosition(string column)	Real	Get the position of the column with the specified tag.
GetColumnName(string column)	String	Get the title of the column with the specified tag.
GetColumns()	Array of STRING	Get the columns in the grid.
GetTitles()	Array of strings	Get the titles for the grid.
GetNumberColumns()	REAL	Return the number of columns.
IsColumnVisible (string column)	Boolean	Query the visibility of the specified column.
ResetColumnFilters()		Reset all of the column filters.
ResizeAllColumns()		Resize columns to fit in available width of form.
setColumnColor(string column, string colour)		Set the column colour to a string (eg 'red')
SetColumnPosition(string, real)		Set the position of the column.
SetColumnTextAlignment(string column, string alignment)		Set the alignment of text in the column to CENTRE, CENTER, LEFT or RIGHT
SetColumnVisibility(String, Boolean)		Show/hide the specified column.

Name	Type	Purpose
SetColumnWidth(string column, real width)		Set the column to a designated width.
setNameColumnImage()		Displays standard icons in a "NAME" attribute column when database elements are used.
SortColumnAscending(String)		Sort in ascending order the specified column.
SortColumnsAscending(Array)		Sort in ascending order the specified columns.
SortedColumnCollection()	Array	Get the sorted columns,
<b>Methods on cells in the grid</b>		
DoDabaconCellUpdate(Array args)		This function is provided to support user code in the BeforeCellUpdate event when the grid is displaying Dabacon element/attribute data. The array provided as argument to the BeforeCellUpdate event can simply be passed on to this function to modify the Dabacon element/attribute in synchronisation with the cell data. If the function is unable to perform the modification for any reason the Array[3] and Array[4] values will be set to indicate the problem. For further information refer to <a href="#">Add an Event to the Addin</a> .
GetCell(string rowTag, string columnTag)	STRING	Get the content of the cell in the specified row and column.
SetCellColor(string rowTag, string columnTag, string colour)		Set the colour of a cell (eg 'red').
SetCellColor(string rowTag, string columnTag, real red, real green, real blue)		Set the colour of the cell to the RGB value.
SetCellTextAlignment(string rowTag, string columnTag, string alignment), where alignment=CENTRE, CENTER, LEFT, RIGHT		Set the alignment of text in the cell to CENTRE, CENTER, LEFT or RIGHT.
setCellValue(string rowTag, string columnTag, string value)		Set a value in the cell.
setEditableCell(string rowTag, string columnTag, Boolean editable)		Enable/disable cell editing.

## 6.9 Input Mask Characters

Masks can be set on data within a column. The mask will restrict the format and type of the data which can be entered into cells within the column. The mask can consist of the following characters:

Character	Description
#	Digit placeholder. Character must be numeric (0-9) and entry is required.
.	Decimal placeholder. The actual character used is the one specified as the decimal placeholder by the system's international settings. This character is treated as a literal for masking purposes.
,	Thousands separator. The actual character used is the one specified as the thousands separator by the system's international settings. This character is treated as a literal for masking purposes.
:	Time separator. The actual character used is the one specified as the time separator by the system's international settings. This character is treated as a literal for masking purposes.
/	Date separator. The actual character used is the one specified as the date separator by the system's international settings. This character is treated as a literal for masking purposes.
\	Treat the next character in the mask string as a literal. This allows you to include the '#', '&', 'A', and '?' characters in the mask. This character is treated as a literal for masking purposes.
&	Character placeholder. Valid values for this placeholder are ANSI characters in the following ranges: 32-126 and 128-255 (keyboard and foreign symbol characters).
>	Convert all the characters that follow to uppercase.
<	Convert all the characters that follow to lowercase.
A	Alphanumeric character placeholder. For example: a-z, A-Z, or 0-9. Character entry is required.
a	Alphanumeric character placeholder. For example: a-z, A-Z, or 0-9. Character entry is not required.
9	Digit placeholder. Character must be numeric (0-9) but entry is not required.
-	Optional minus sign to indicate negative numbers. Must appear at the beginning of the mask string.
C	Character or space placeholder. Character entry is not required. This operates exactly like the '&' placeholder, and ensures compatibility with Microsoft Access.
?	Letter placeholder. For example: a-z or A-Z. Character entry is not required.
Literal	All other symbols are displayed as literals; that is, they appear as themselves.
n	Digit placeholder. A group of n's can be used to create a numeric section where numbers are entered from right to left. Character must be numeric (0-9) but entry is not required.



Character	Description
mm, dd, yy	Combination of these three special strings can be used to define a date mask. mm for month, dd for day, yy for two digit year and yyyy for four digit year. Examples: mm/dd/yyyy, yyyy/mm/dd, mm/yy.
hh, mm, ss, tt	Combination of these three special strings can be used to define a time mask. hh for hour, mm for minute, ss for second, and tt for AP/PM. Examples: hh:mm, hh:mm tt, hh:mm:ss.



## 7 AVEVA My Data

My Data is a .NET addin that allows database elements to be collected together in collections which are persisted between modules. The API to modify My Data is also available from PML.

My Data assembly can be imported as follows

```
IMPORT 'MyDataAddin'
handle any
endhandle
USING NAMESPACE 'Aveva.Pdms.Presentation.MyDataAddin'
!!mydata=object pmlmydata()
```

### 7.1 My Data API

The following api is then available on the My Data object.

Name	Type	Purpose
PMLMyData		MyData constructor
AddElement(String elename)		Adds an orphaned element of given name
AddElements(Array elements)		Adds orphaned elements as array of strings
AddCollection(String name)		Add collection of given name
AddCollections(Array collections)		Add collections as an array of strings
AddElementToCollection(String elementName, String collectionName)		Add element of given name to collection of given name
AddElementsToCollection(Array elements, String collectionName)		Add elements as array of strings to collection of given name
RemoveElement(String elename)		Remove orphaned element of given name
RemoveElements(Array elements)		Remove orphaned elements as array of strings
RemoveAllElements		Remove all elements
RemoveCollection(String name)		Remove collection of given name

RemoveCollections(Array names)		Remove collections as an array of strings
RemoveAllCollections		Remove all collections
RemoveElementFromCollection(String elementName, String collectionName)		Remove given element from given collection
RemoveElementsFromCollection(Array elements, string collectionName)		Remove elements as array of strings from given collection
RemoveAllElementsFromCollection(String collectionName)		Remove all elements from given collection
RemoveAll()		Remove all orphaned elements and collections
Elements()	Array	Return orphaned elements as array of strings
ElementsInCollection(String collectionName)	Array	Return the elements in given collection as an array of strings
Collections()	Array	Return the collection names as an array of strings
RenameCollection(string oldName, string newName)		Rename given collection with new name

## 7.2 Using My Data from PML

From PML elements and collections can be added, removed and edited. For example:

### 7.2.1 Add an Element

```
!!mydata.addElement('/VESS1')
!!mydata.addElement('/VESS2')
!!mydata.addElement('/PUMP1')
!!mydata.addElement('/PUMP2')
```

### 7.2.2 Add an Array of Elements

```
!a=object array()
!a[1]=' /VESS1 '
!a[2]=' /VESS2 '

!b=object array()
!b[1]=' /PUMP1 '
!b[2]=' /PUMP2 '

!!mydata.addElements(!a)
!!mydata.addElements(!b)
```

### 7.2.3 Remove an Element

```
!!mydata.removeElement('/VESS1')
```

### 7.2.4 Remove Everything (Elements and Collections)

```
!!mydata.removeAll()
```

### 7.2.5 Remove an Array of Elements

```
!!mydata.addElement('/VESS1')
!!mydata.addElement('/VESS2')
!!mydata.addElement('/PUMP1')
!!mydata.addElement('/PUMP2')
!!mydata.removeElements(!a)
```

### 7.2.6 Add a Collection

```
!!mydata.addCollection('My collection1')
!!mydata.addCollection('My collection2')
```

### 7.2.7 Add an Array of Collections

```
!collections = object array()
!collections[1] = 'My collection3'
!collections[2] = 'My collection4'
!!mydata.addCollections(!collections)
```

### 7.2.8 Remove a Collection

```
!!mydata.removeCollection('My collection1')
!!mydata.removeCollection('My collection2')
```

### 7.2.9 Remove an Array of Collections

```
!collections = object array()
!collections[1] = 'My collection3'
!collections[2] = 'My collection4'
!!mydata.removeCollections(!collections)
```

### 7.2.10 Add an Element to a Collection

```
!!mydata.addCollection('My collection1')
!!mydata.addCollection('My collection2')
!!mydata.addElementToCollection('/VESS1', 'My collection1')
!!mydata.addElementToCollection('/VESS2', 'My collection2')
```

### 7.2.11 Add an Array of Elements to a Collection

```
!!mydata.addCollection('My collection1')
!!mydata.addCollection('My collection2')
!a=object array()
```

```
!a[1]=' /VESS1'
!a[2]=' /VESS2'
!b=object array()
!b[3]=' /PUMP1'
!b[4]=' /PUMP2'
!!mydata.addElementsToCollection(!a, 'My collection1')
!!mydata.addElementsToCollection(!b, 'My collection2')
```

### 7.2.12 Remove an Element from a Collection

```
!!mydata.removeElementFromCollection('/VESS1', 'My
collection1')
!!mydata.removeElementFromCollection('/VESS2', 'My
collection1')
```

### 7.2.13 Remove all Elements

```
!!mydata.removeAllElements()
```

### 7.2.14 Remove all Elements from Collection

```
!!mydata.removeAllElementsFromCollection('My collection1')
```

### 7.2.15 Remove all Collections

```
!!mydata.removeAllCollections()
```

### 7.2.16 Remove an Array of Elements from a Collection

```
!!mydata.removeElementsFromCollection(!b, 'My collection2')
```

### 7.2.17 Rename a Collection

```
!!mydata.renameCollection('collectionA', 'collectionX')
!!mydata.renameCollection('collectionB', 'collectionY')
```

### 7.2.18 Query Elements and Collections

```
q var !!mydata.elements()
q var !!mydata.collections()
```

### 7.2.19 Query Elements in a given Collection

```
q var !!mydata.elementsInCollection('My collection1')
```

## 8 PML File Browser

A standard Windows file browser window can be opened by using PML syntax.

Create a File Browser object:

```
import 'PMLFileBrowser'

using namespace 'Aveva.Pdms.Presentation'

!browser = object PMLFileBrowser('OPEN')
```

If no argument is given then the object will default to **Open** otherwise the user can specify the value **Open** or **Save**.

The user can query the object and the methods on the PMLFileBrowser instance as follows:

```
Q Var !browser

Q methods !browser
```

To show the File Browser dialog use the following syntax:

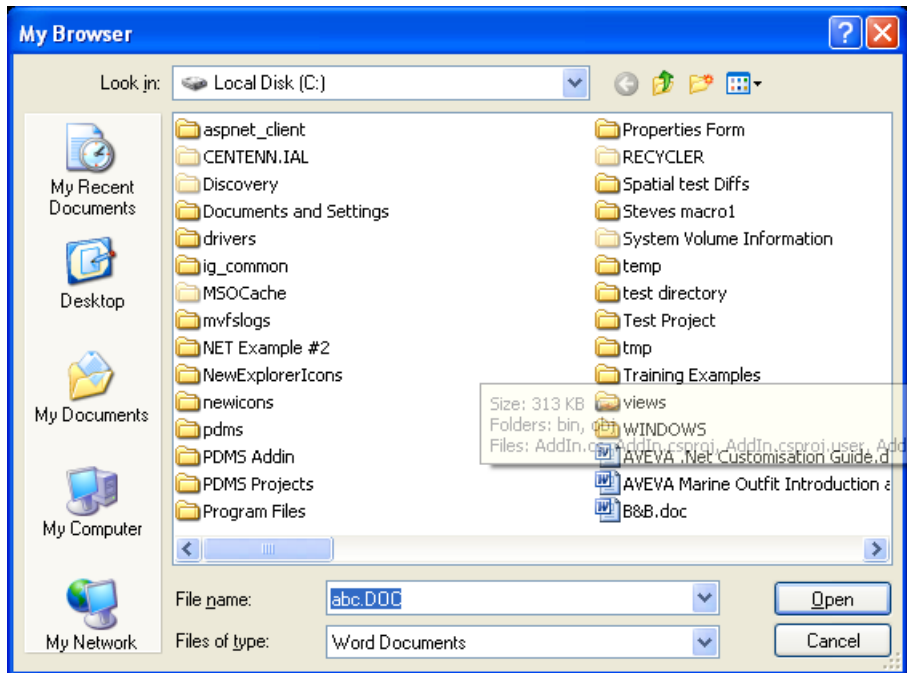
```
!browser.show(directory,seed,title,exists,filter,index)
```

Argument	Type	Purpose
Directory	String	Default directory to display in File Browser.
Seed	String	Default value to be populated in the Name input box of the File Browser.
Title	String	Value to be displayed in the File Browser window title bar.
Exists	Boolean	Check if the input file exists.
Filter	String	The type of files to open. Must be in the format of a description, followed by " " and then the file types - for example  Word Documents .DOC  or  Text files (*.txt) *.txt All files (*.*) *.*
Index	Integer	For multiple strings, the index of the filter currently selected.

For example the following syntax could be entered at the command line.

```
!browser.show('C:\','abc.DOC','My Browser',true,'Word Documents|*.DOC',1)
```

After entering the command the following window will be displayed. Notice the default values have been set as specified:



Use the following syntax to return the name of the file selected in the File Browser window.

```
q var !browser.file()
```