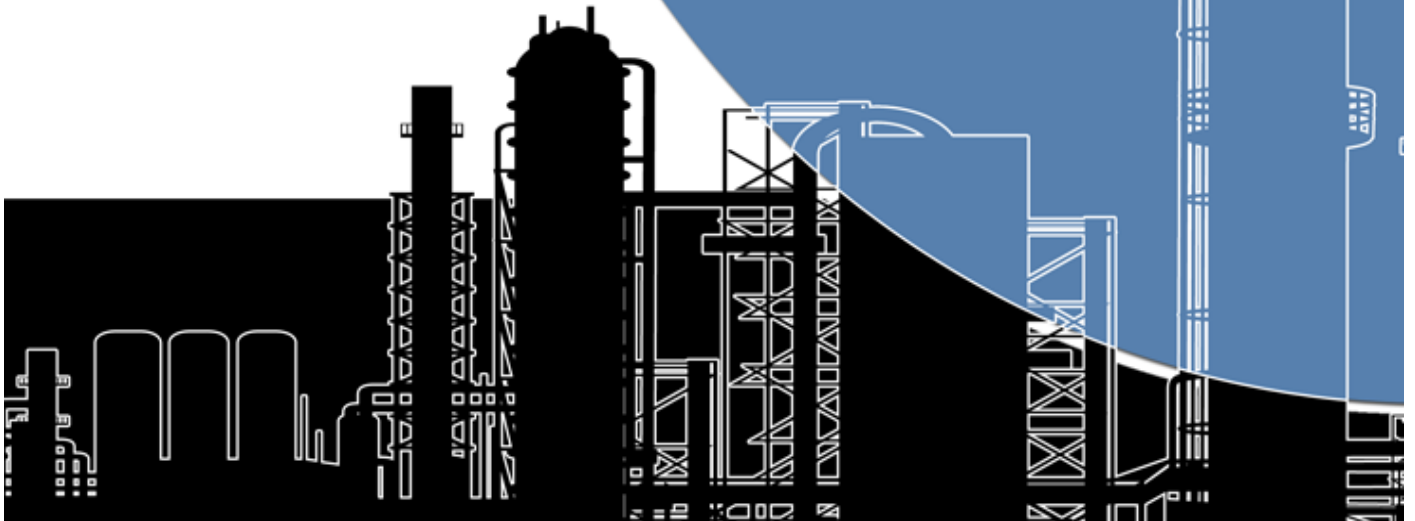




Training Guide

TM-1881
AVEVA Everything3D
PML Form Design

Copia para EEA



Copia para EE AA

Revision Log

Date	Revision	Description	Author	Reviewed	Approved
6.02.2013	0.1	General review and update	MZA		

Updates

Change highlighting will be employed for all revisions. Where new or changed information is presented section headings will be highlighted in **Yellow**.

Suggestion / Problems

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

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

Visit our website at <http://www.aveva.com>

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.

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. 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

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.



CONTENTS

1	Introduction	9
1.1	Aim	9
1.2	Objectives	9
1.3	Prerequisites	9
1.4	Course Structure	9
1.5	Using this Guide	9
1.6	Modifications to the PMLUI and PMLLIB	10
	Worked Example - Updating the environment variables	11
2	Forms	13
2.1	Forms are Global Objects	13
2.2	Dynamic Loading of Objects, Forms and Functions	13
2.3	Defining a Form	14
2.3.1	Using the .NET Framework	15
2.3.2	Built-in Methods for Forms	15
2.4	Callbacks	16
2.5	Form Gadgets	17
2.5.1	Built-in Members and Methods for Gadgets	17
2.5.2	Gadget Positioning	18
2.5.3	Docking and Anchoring Gadgets	19
2.6	Paragraph Gadgets	20
2.7	Button Gadgets	20
2.8	Text Entry Gadgets	21
2.9	Format Object	22
2.10	List Gadgets	22
2.11	Frame Gadgets	23
2.12	Textpane Gadgets	25
2.13	Option Gadgets	25
2.14	Toggle Gadgets	26
2.15	Radio Gadgets	27
2.16	Container Gadgets	28
2.17	Tooltips	28
	Exercise 1 - Working with a PML form	29
2.18	Progress Bars and Interrupting Methods	31
2.19	Open Callbacks	32
2.20	Menus	33
2.20.1	Defining a Menu Object on a Form	33
2.20.2	Defining a Bar Menu on a Form	34
2.20.3	Defining a Popup Menu on a Form	34
2.20.4	Adding Menu Objects to a Form	35
2.21	User-Defined Form Members	36
	Exercise 2 – Extending the form	37
3	PML Objects	39
3.1	Built-in PML Object Types	39
3.2	Methods Available to All PML Objects	40
3.3	The FILE Object	41
3.3.1	Using FILE Objects	41
3.3.2	Opening a FILE Object in Notepad	41
3.3.3	Using the Standard File Browser	42
	Exercise 3 – Using the FILE Object	44
4	Collections	45
4.1	COLLECT Command Syntax (PML 1 Style)	45
4.2	EVALUATE Command Syntax (PML 1 Style)	45
4.3	COLLECTION Object (PML 2 Style)	46
4.4	Evaluating the Results From a COLLECTION Object	46

Exercise 4 – Equipment Collections	47
5 View Gadgets	49
5.1 Alpha Views	49
5.2 Plot View Example	49
5.3 Volume View Example	50
Exercise 5 – Adding a Volume View to a form	52
6 Event Driven Graphics (EDG)	55
6.1 A Simple EDG Event	55
6.2 Using EDG	56
Exercise 6 – Adding EDG to Forms	58
7 PML.NET	61
7.1 Import an Assembly into E3D	61
7.2 Syntax	61
Exercise 6 - Create a File browser Object	62
7.3 Creating a PML form containing the .NET Control	62
Exercise 7 - Explorer Control on PML Form	63
7.4 Grid Control	63
7.4.1 Applying Data to the Grid	64
7.4.2 Events and Callbacks	64
Exercise 8 - A Grid Control on a PML Form	65
Exercise 9 – Developing a PML .NET form	67
8 Miscellaneous	69
8.1 Error Tracing	69
8.2 PML Encryption	69
9 Menu and Toolbar Additions (Bar Menu Modules Only)	71
9.1 Miscellaneous Notes	71
9.2 Adding a Menu/Toolbar	71
9.3 Form and Toolbar Control for Bar Menu Modules	71
9.3.1 Bar Menu and Toolbar	71
9.3.2 Define Toolbar	71
9.3.3 Adding to Menus	71
9.3.4 Adding New Menus to Form	72
9.4 Example Object Including Toolbars, Bar Menus and Menus for Bar Menu Modules	72
Worked Example – Creating a Command Bar (UIC file modification)	75
10 Tabbed Menus in Model	79
10.1 Defining a Command	79
10.2 Loading and Registering a Command	80
10.3 Attaching a Command to the UI	80
10.4 Killing a Command	80
Worked Example – Customize Tabbed Menu	81
Appendix A – Summary of Grid Control Gadget Methods	83
Appendix B – Exercise Solutions (Example Code)	89
Appendix B1 - Example c2ex1.pmlfrm	89
Appendix B2 - Example c2ex2.pmlfrm	93
Appendix B3 - Example c2ex3.pmlfrm	97
Appendix B4 - Example c2ex4.pmlfrm	98
Appendix B5 - Example !traExampleVolumeView	101
Appendix B6 - Example c2ex5.pmlfrm	104
Appendix B7 - Example c2ex6.pmlfrm	113
Appendix B8 - Example traExampleExplorer.pmlfrm	117
Appendix B9 - Example traExampleGridControl.pmlfrm	119

Appendix B10 - Example c2ex9.pmlfrm	122
Appendix B11 - Example apppml.pmlobj.....	125
Appendix C – PML training utility.....	127
Appendix C.1 - Available E3D Colours form	127
Appendix C.2 - Available E3D Icons form	127
Appendix C.3 – Training Examples form	128


Copia para EE AA

Copia para EE AA



1 Introduction

This manual is designed to give an introduction to the AVEVA Programming Macro Language. There is no intention to teach software programming but only provide instruction on how to customise Everything3D (AVEVA E3D) using Programmable Macro Language (PML) in AVEVA E3D.

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

1.1 Aim

The following points need to be understood by the trainees:

- Understand how PML can be used to customise AVEVA E3D.
- Understand how to create Forms
- Understand how to use the built-in objects of AVEVA E3D
- Understand the use of Addins to customise the environment.

1.2 Objectives

At the end of this training, the trainees will have:

- Knowledge of how Forms are created and how Form Gadgets and Form Members are defined
- Understanding of Menus and Toolbars are defined with PML
- Understanding of Collections, Basic Event Driven Graphics, Error Tracing and PML Encryption

1.3 Prerequisites

The participants must have:

- Completed an AVEVA Basic Design Course and have a familiarity with AVEVA E3D
- Completed the PML: Macros & Functions Course or have familiarity with PML.

1.4 Course Structure

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

1.5 Using this Guide

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

Menu pull downs and button press actions are indicated by **bold dark turquoise text**.

Information the user has to key-in will be in **bold red text**.

 *Additional information will be highlighted*

 *Reference to other documentation will be separate*

System prompts should be bold and italic in inverted commas i.e. '***Choose function***'

Example files or inputs will be in the courier new font with colours and styles used as before.

1.6 Modifications to the PMLUI and PMLLIB

This training course uses examples provided in a separate folder structure to the standard AVEVA E3D installation. For this reason, the additional search paths need to be configured to make these examples available.

It is possible to get AVEVA E3D to look in different places for PML and this is done by setting the environment variables to multiple paths. This allows the standard install to be kept separate from user and company customisation. This is done by updating the variable to include another path, for example:

```
set PMLUI=C:\AVEVA\Plant\Training\pmlui %pmlui%
```

```
set PMLLIB=C:\AVEVA\Plant\Training\pmlib %pmlib%
```


This will put the additional file path in front of the standard (which would have already been defined in the .bat file). This change can also be checked in a AVEVA E3D session by typing **q evar PMLUI** or **q evar PMLLIB** onto the command window.

 *If these steps have already been completed, then there is no need to repeat them again here.*

Cópia para E3D

Worked Example - Updating the environment variables

- 1 Extract the provided files into the folder **C:\AVEVA\Plant\Training**
- 2 Open the **custom_evars.bat** file in a suitable text editor
If not done already, add the following lines:
set PMLUI= C:\AVEVA\Plant\Training\pmlui %pmlui%
set PMLLIB= C:\AVEVA\Plant\Training\pmlib %pmlib%
- 3 Load **AVEVA E3D Model** module.

 *The choice of project and MDB will depend on what is available in the AVEVA E3D setup. This should be discussed with your trainer.*
- 4 Confirm that the search paths have been correctly updated by typing **q evar PMLUI** or **q evar PMLLIB** into the command window.

Cópia para EE

Copia para EE AA

2 Forms

2.1 Forms are Global Objects

A AVEVA E3D form is a PML object, stored as a **GLOBAL VARIABLE** in the system. Form gadgets (i.e. button, lists, toggles etc) are PML objects too, but can also be considered **MEMBERS** of the form object. As PML form is a global variable, once defined, the information held within a form is available to the rest of AVEVA E3D.

To find out information about a form, it can be queried as if it was an object. Before the form can be queried, it needs to be loaded. For example, show the Graphics Settings form (click the **3D View > Settings > Graphics** button to display the **Graphics Settings** form.

As the form has been shown, it has been loaded into AVEVA E3D and registered with the Forms and Menus global object (!!FMSYS). There is a method on this global object that returns the names of any visible forms. Type the following into the command window:

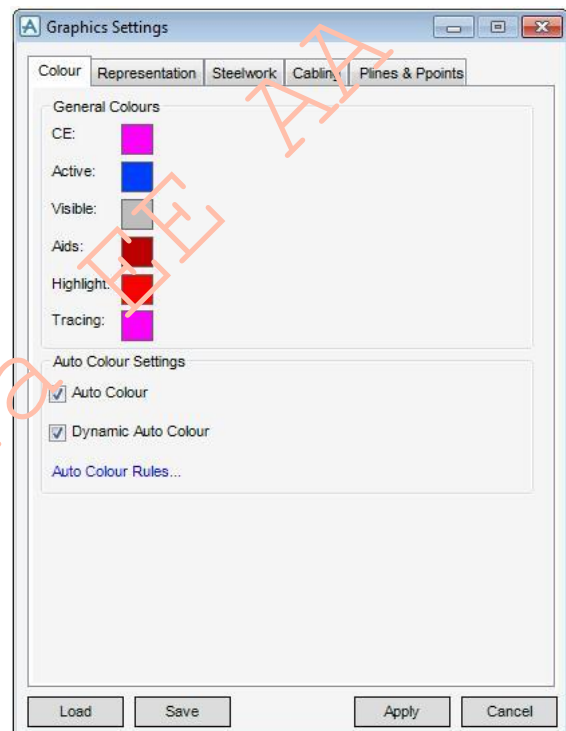
```
q var !!fmsys.shownforms ()

<ARRAY>
[1] <FORM> GPHSETTINGS
[2] <FORM> GPH3DDESIGN1
```

This shows that the name of the form is **GPHSETTINGS** (or **!!gphsettings** as it is a global variable).

i If the name was already known then form could have been shown by typing **show !!gphsettings**. The **SHOW** syntax loads and displays a form. If you wish to load the form, but not see it, you could type **load !!gphsettings**

By type **q var !!gphsettings** onto the command line, a list of form members is returned. As the form gadgets are members of the form, they will all be listed and can be investigated further.



The first gadget listed is called **APPLY** (representing the Apply button on the form). To find out information about it, type **q var !!gphsettings.apply**.

Specific information about the gadget can be queried directly e.g.:

```
q var !!gphsettings.apply.tag
q var !!gphsettings.apply.val
q var !!gphsettings.apply.active
```

i For a form to help get this information, type **show !!pmlforms**. The toggle allows shown forms to be investigated. Without it toggled, all loaded forms are available.

2.2 Dynamic Loading of Objects, Forms and Functions

When a PML object is used for the first time, AVEVA E3D loads it automatically. This is possible because of the pml.index file found in the PMLLIB search paths. As forms are PML objects, this applies to forms too. For example, if the following was typed into the command window, the two objects will be loaded:

```
!gphline = object GPHLINE()
```

show !!GPHMEASURE

i A form can also be defined, but not shown by typing **pml load form !!GPHMEASURE**

Once an object is loaded by AVEVA E3D, the definition is stored within the system. This means that if the object definition file is changed or updated whilst it is loaded inside AVEVA E3D, then the PML object will need to be reloaded. To reload PML object, it needs to be referred to by name. Type the following:

pml reload form !!GPHMEASURE
pml reload object GPHLINE

i Notice how the command is different for a form as it includes **!!** and the keyword **FORM**.

If any instances of the previous object definition are still held within AVEVA E3D, then these variables will need to be destroyed and redefined to avoid any definition clashes. For example:

pml reload object GPHLINE
!gphline.delete()
!gphline = object GPHLINE()

If a new file is created whilst AVEVA E3D is open, the file will not be mapped and AVEVA E3D will not know its location. Even if it is saved in an appropriate file path, it will need to be mapped.

To update the pml.index file and remap files, type **pml rehash**. This will remap all the files within the first file path in the PMLLIB variable. To remap all files in all of the PMLLIB file paths, type **pml rehash all**

i This command will only work if there is write access to the pml.index file.

2.3 Defining a Form

A form is defined within a **.pmlfrm** file and should be saved under the PMLLIB search path. The file will define the form, its members and any associated methods.

The file will first define the form gadgets and members within the keywords **SETUP FORM** and **EXIT**. The rest of the file will contain any form methods. These are defined within the keywords **DEFINE METHOD** and **ENDMETHOD**.

As all the form information can be contained within a single file, there is no longer the need to split a form over multiple files. This used to be a requirement of the PMLUI file structure.

Search for **gphsettings.pmlfrm** in the PMLLIB folder and open the file to investigate. Type **q evar PMLLIB** to find out what search paths are in use.

2.3.1 Using the .NET Framework

Form objects make use of the Windows .NET framework. This allows features such as the following to be used:

- Docking forms
- Anchoring gadgets (moving gadgets when resizing)
- Multicolumn lists
- Tabs on Forms

Although it is now possible to dock a form into AVEVA E3D, it is not applicable for every form. Consider the following:

- Forms in heavy use, or need to remain open can be docked.
- Any forms with an “OK” or “Cancel” button, should not dock.
- If a form has a menu bar, then these CANNOT be docked.

To declare a form as able to dock, this has to be done on the top of the form definition. By including dialog dock, the ability is declared:

For example, the follow would dock the form to the left when it is loaded for the first time:

```
setup form !!exampleForm dialog
```

To define a floating form that is able to dock, use the following line:

```
setup form !!exampleForm dialog resizable
```

To define a form of a certain size in the centre of the screen, use the following line:

```
setup form !!exampleForm document at xr 0.5 yr 0.5 size 100 100
```

If no additional details are included, the default will create Dialog, non-resizable, fixed size form. A form will always be as big as it needs to be to display the contents.

 For more examples, refer to syntax graph for the FORM object in the AVEVA E3D Software Customisation Reference Manual

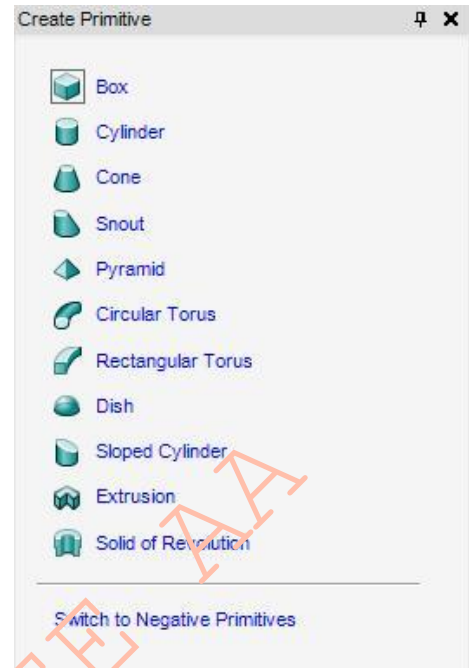
2.3.2 Built-in Methods for Forms

Although is possible to define methods within user-defined forms (discussed later), all FORM objects have built-in methods available. Try on the following on the command line:

```
!!gphsettings.show()
q var !!gphsettings.shown()
!!gphsettings.hide()
q var !!gphsettings.shown()
```

In this example, the `.show()` method is used to show the form, the `.shown()` method returns whether the form is shown and the `.hide()` method hides the form from the user.

 For more examples, refer to the FORM object in the AVEVA E3D Software Customisation Reference Manual



2.4 Callbacks

A **FORM** object has a number of **CALLBACK** members. This means that if the user interacts with a form, a stored action can be performed. Stored as strings, these calls can execute commands, run methods or call PML functions.

These actions occur when the form completes certain actions (e.g. is shown or closed). The following example PML form definition demonstrates the various callbacks on a FORM object.

Show the example form by typing `show !!traExampleCallback` into the command window.

```
setup form !!traExampleCallback
  !this.formTitle      = |Callback Example|
  !this.initCall       = |!this.init()|
  !this.firstShownCall = |!this.firstShown()|
  !this.okCall         = |!this.okCall()|
  !this.cancelCall     = |!this.cancelCall()|
  !this.quitCall       = |!this.quitCall()|
  !this.killingCall    = |!this.killCall()|

  button .ok          | OK |          OK
  button .cancel      |Cancel| at x20 CANCEL
exit

define method .traExampleCallback()
  $p Constructor Method
endmethod

define method .init()
  $p Initialise Method
endmethod

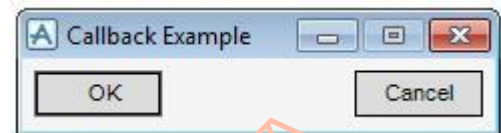
define method .firstShown()
  $p First Shown Method
endmethod

define method .okCall()
  $p OK Method
endmethod

define method .cancelCall()
  $p Cancel Method
endmethod

define method .quitCall()
  $p Quit Method
endmethod

define method .killCall()
  $p Kill Method
endmethod
```



Test the form by interacting with it (i.e. press the buttons, close it). What is printed to the command line?

The callbacks on a form object can run any command syntax, global function, local method.

In this example, the callbacks call local methods of similar names (to help recognise the methods). They could have called any methods, even the same methods.

The only limitation is that the callback must be valid, otherwise an error will occur

i Notice how the form can be referred to as `!this` within its definition. The local variable `!this` refers to the owning object and can be used in any object definition.

There are seven main event callbacks built into the FORM object:

- The CONSTRUCTOR method was called when the form loaded, recognised as it has the same name as the owning form.
- The INITCALL method is called every time the form is shown (perfect for setting default values)
- The FIRSTSHOWNCALL method is called the when the form is activated (first shown)
- The OKCALL method is called by any button gadget with the OK in its definition
- The CANCELCALL method is called by any button gadget with CANCEL in its definition
- The QUITCALL method is called by clicking the close button on the form.
- The KILLINGCALL method is called when the form is unloaded (killed or reloaded)

 *If no callbacks are defined for OKCALL, CANCELCALL and QUITCALL, the default is to hide the form.*

2.5 Form Gadgets

There are many kinds of form gadgets, each an object that will have its own MEMBERS and METHODS. When defining gadgets on a form, there are two common aims:

- To specify the area to be taken up on the form
- To define the action to be taken by the gadget if interacted with

It is the position and size of the gadget that determines the area taken up and its action is defined by its CALLBACK member.

2.5.1 Built-in Members and Methods for Gadgets

As gadgets are PML objects, there are a variety of useful members and built-in methods that can be used. Based on the previous example form, type the following onto the command line:

To grey-out the **OK** button

!!traExampleCallback.ok.active = FALSE

To reactive the **OK** button

!!traExampleCallback.ok.active = TRUE

To hide the **CANCEL** button

!!traExampleCallback.cancel.visible = FALSE

To show the **CANCEL** button again

!!traExampleCallback.cancel.visible = TRUE

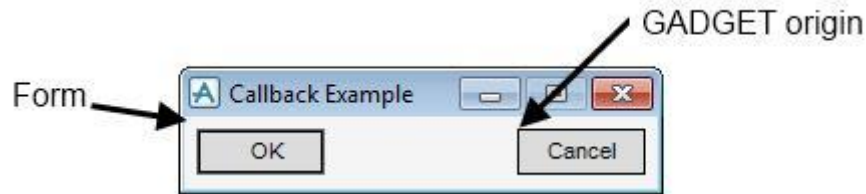
To update the tooltip on the **OK** button

!!traExampleCallback.ok.setToolTip(|This is an OK button|)

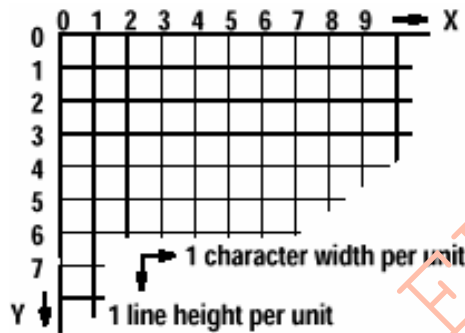
 *For further information, refer to the FORM object in the Software Customisation Reference Manual*

2.5.2 Gadget Positioning

Gadgets are positioned on a form from top left using the AT syntax. The AT syntax defines the origin of the gadget in relation to the owning object (i.e. FORM or FRAME).

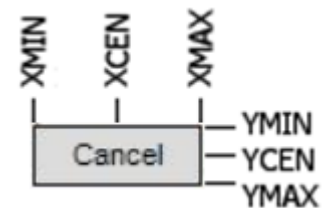


The origin of the form is the top left corner, and positions are set from here. The X positions are across the screen to the right and the Y positions are down the screen.



Gadgets can be positioned explicitly (i.e. x 10) or in relation to other objects (i.e. ymin.ok).

When referring to other objects, there 6 known positions on a gadget: **XMIN**, **XCEN**, **XMAX**, **YMIN**, **YCEN** and **YMAX**. A gadget can be thought of as an enclosing box that will enclose the geometry of the gadget and these points will sit on this enclosing box.



For example, the positions on a button are shown in the adjacent picture.

The following are some examples of the AT syntax. To position a gadget at a known position use:
at x 2 y 10

To position a CANCEL button using the XMAX and YMIN of OK button, use:
at xmax.ok + 10 ymin.ok

To position a DISMISS button in the bottom corner of a form use:
at xmax form – size ymax form

 Notice how the size of the gadget can be used by the keyword **SIZE**

If a gadget can be positioned using the AT syntax, then **<fgpos>** will appear in its syntax graph. This actually refers to another syntax graph. These common syntax graphs apply to multiple gadgets and are typically found at the beginning of the Software Customisation Reference Manual

For example, the syntax graph for a button refer to **<fgpos>**

```

>-BUTTON gname +- LINKLabel +- tagtext -----
|               +- <fgpos> -----

```

This means that the syntax found in <fgpos> can be added to the button definition.

```
>-- <fgpos> -- AT --+- val val -----
+- X val -----
+- XMIN -.
+- XCEN -|
+- XMAX --+- <fgprl> -|
+- Y val -----
+- YMIN -.
+- YCEN -|
+- YMAX --+- <fgprl> -|
-->
```

This syntax graph can also refer to another. The syntax graph **<fgprl>** helps define the relative position of the gadget. Try and find it in the reference manual.

i For further assistance reading syntax graphics and more examples, refer to the *Software Customisation Reference Manual*.

2.5.2.1 Position Gadgets Using the Path Command

The **PATH** command can be used to define logical positions for subsequent gadgets. This method has been superseded by the **AT** syntax, but is still valid and has been included for information.

After a gadget has been defined, the next gadget is positioned based on a **PATH**, **HDIST** or **VDIST** and **HALIGN** or **VALIGN**. As an example, see the picture below:

```
button .but1                $* default placement
PATH DOWN
HALIGN CENTRE
VDIST 2

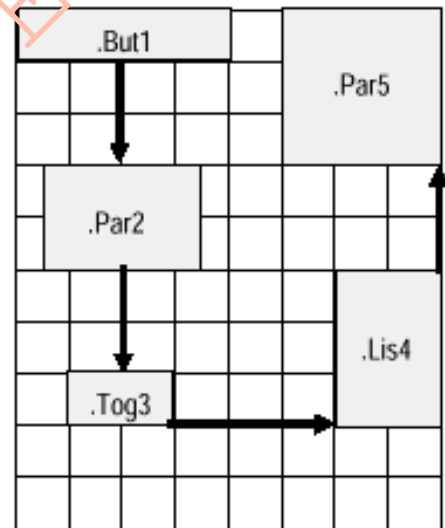
paragraph .par2 width 3 height 2 $* auto-placed
toggle .tog3                    $* auto-placed

PATH RIGHT
HDIST 3
VALIGN BOTTOM

list .lis4 width 2 height 3      $* auto-placed

PATH UP
HALIGN RIGHT

paragraph .par5 width 3 height 3 $* auto-placed
```



2.5.3 Anchoring Gadgets

After a gadget has been loaded, its size and position are fixed. The position or size can only be manually changed in the form definition. If a form can be resized, it might be necessary for gadgets to move/resize to reflect the changes to the form.

The **ANCHOR.** syntax is available if **<fganch>** is included in the syntax graph of the gadget. Once a gadget is declared as Anchored it remains so for the life of the form.

- **ANCHOR** - controls the position of an edge of the gadget relative to the corresponding edge of its container. For example, if a **DISMISS** button is anchored to the Right and Bottom, it will remain the same distance from the bottom right of the form if it is resized.

Refer AVEVA E3D Software Customisation Reference Manual for the Syntax Graphs.

Show the example form by typing **show !!traExampleDocking.**

```

setup form !!traExampleAnchor dialog resizable
    !this.formTitle = |Anchor|
    !buttpos = |xmax.fl - size ymax.fl - size|
    frame .fl panel anchor ALL width 26 height 5
        button .butt1 |Anchor L+R+T| anchor L+R+T
        button .butt2 |Anchor B+R| at $!buttpos anchor R+B

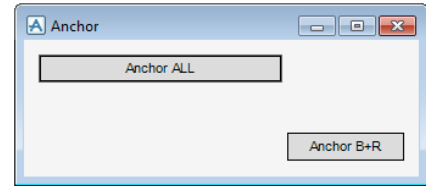
exit

define method .traExampleDocking()
    $p Constructor Method
endmethod

define method .quitCall()
    $p Quit Method
endmethod

define method .killCall()
    $p Kill Method
endmethod

```



Observe how the form behaves when it is resized. Try altering the directions which the gadgets are docked.

2.6 Paragraph Gadgets

Paragraph gadgets are simple named gadgets that allow a piece of TEXT or a PICTURE (PIXMAP) to be displayed on a form. It is a passive gadget that cannot be selected by the user so has no callback. Paragraph gadgets are typically used display information or images.

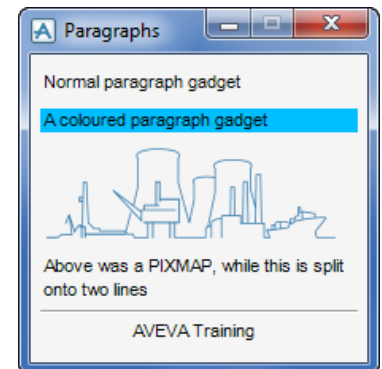
Show the example by typing **show !!traExampleParagraphs.**

```

setup form !!traExampleParagraphs
    !this.formTitle = |Paragraphs|
    para .para1 text |Normal paragraph gadget| width 25
    !t = |A coloured paragraph gadget|
    para .para2 at x0 ymax.para1 backg 157 text |$!t| width
    25
    para .para3 at x0 ymax.para2 pixmap
    /C:\AVEVA\Plant\Training\pmlib\icons\avevalogo.png width
    30 height 75
    para .para4 at x0 ymax.para3 + 0.5 text || wid 25 hei 2
exit

define method .traExampleParagraphs()
    -- Update the displayed text using CONSTRUCTOR method
    !this.para4.val = |Above was a PIXMAP, while this is
    split onto two lines|
endmethod

```



Notice how the CONSTRUCTOR method was used to set value of the last gadget. It was only given a size to reserve the space during definition.

Refer to the Reference Manual and Guide for more information about paragraph gadgets

2.7 Button Gadgets

Button gadgets are typically used to invoke an action or to display a child form. Its **CALLBACK** can call any command, method or function. If a callback and a child form are both specified, the callback command will be run before the child form is displayed.

Show the example by typing **show !!traExampleButtons**.

```

setup form !!traExampleButtons
  !this.formTitle = |Buttons|
  button .butt1 |Normal| width 8
  button .butt2 linklabel |Link Label| at xmax+2.7 wid 6
  button .butt3 |Yellow| at xmax+2.5 backg 4 width 8
  button .butt4 |Deactive| at xmin.butt1 ymax+0.2 width 8
  button .butt5 |Child Form| form !!traExampleParagraphs width 9
  button .butt6 toggle call |!this.check()| pixmap wid 72 hei 21
exit

define method .traExampleButtons()
  -- Deactivate butt3 in CONSTRUCTOR method
  !this.butt4.active = FALSE
  -- Use built in method to apply pictures
  !off = !!pml.getPathName(|leftleftarrow16.png|)
  !on = !!pml.getPathName(|leftrightarrow16.png|)
  !this.butt6.addPixmap(!off, !on)
endmethod

define method .check()
  -- Return the toggle button value to the
  command line
  !check = !this.butt6.val
  $p Value: $!check
endmethod

```



Notice how the !!PML global object was used to get the file path of the picture file. This can be used to find the file path of any file in the pml.index file.

 Refer to the Reference Manual and Guide for more information about button gadgets

2.8 Text Entry Gadgets

A text input gadget provides the user a way of entering a single value into a AVEVA E3D form. A TEXT gadget needs two aspects covered in its definition:

- **WIDTH** – determines the displayed number of characters. An optional scroll width can also be specified
- **TYPE** – determines the type of the variable created when inputting a value. This is important when PML uses the value. You may also supply a **FORMAT** object (explained below) to format the value entered (e.g. 2 d.p. number)

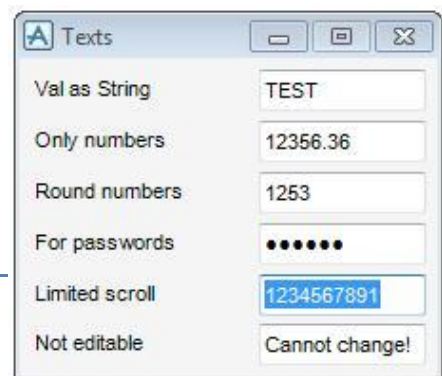
Show the example by typing **show !!traExampleTexts**.

```

setup form !!traExampleTexts
  !this.formTitle = |Texts|
  path down
  text .txt1 |Val as String | width 10 is STRING
  text .txt2 |Only numbers | width 10 is REAL format !!REALFMT
  text .txt3 |Round numbers | width 10 is REAL format !!INTEGERFMT
  text .txt4 |For passwords | width 10 NOECHO is STRING
  text .txt5 |Limited scroll | width 10 scroll 1 is
  STRING
  text .txt6 |Not editable | width 10 is STRING
exit

define method .traExampleTexts()
  !this.txt6.val = |Cannot change!|
  -- Make txt6 uneditable
  !this.txt6.setEditable(FALSE)

```



endmethod

Try and fill in the form with various values to investigate the response

- ❗ *Making a text gadget uneditable still allows the user to select its contents. A de-active text box will be full greyed out and unselectable.*

📖 *Refer to the Reference Manual and Guide for more information about text gadgets*

2.9 Format Object

A **FORMAT** object manages the information needed to convert a number (always in mm) to a STRING. It can also be used apply a format to a text gadget. Format objects are usually defined as global variables so that they are available across AVEVA E3D. For example, type the following onto the command line:

```
!!oneDP = object FORMAT()
!!oneDP.dp = 1
q var !!oneDp
```

There are four standard **FORMAT** objects which are already defined in standard AVEVA E3D:

!!distanceFmt	For distance units
!!boreFmt	For Bore Units
!!realFmt	To give a consistent level of decimal places on real numbers
!!integerFmt	To force real numbers to be integers(0 dp Rounded)

To find out more information about these **FORMAT** object, query them as global variables on the command line **q var !!boreFmt**

For example, the number of decimal places displayed using **!!realFmt** could be changed by typing **!!realFmt.dp = 6** (the default value is 2).

- ❗ *These standard format objects are used within the forms of AVEVA E3D. Changing the definition of these objects will change the way standard product behaves.*

2.10 List Gadgets

A **LIST** gadget presents an **ARRAY** of **STRING** values to the user. A single column, or multi-columns of information can be displayed. If set with a multidimensional array of values, the requirement for columns is implied and will be displayed.

Show the example by typing show **!!traExampleLists**. Test the different lists by trying to select rows.

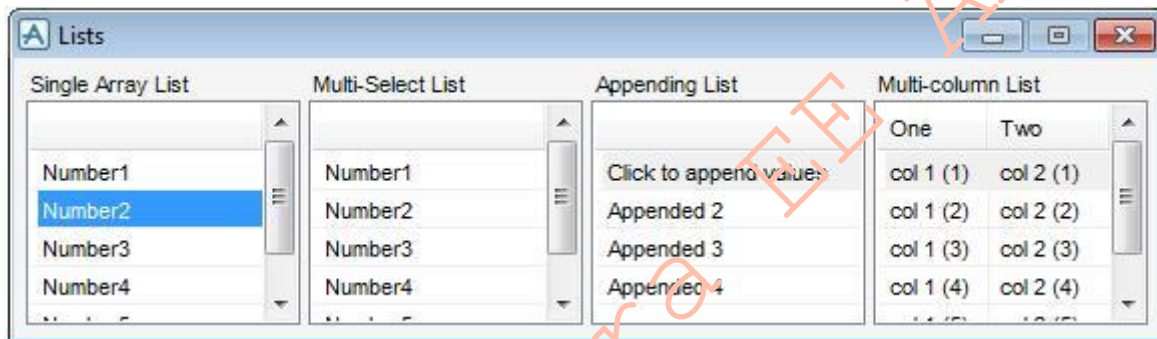
```
setup form !!traExampleLists
!this.formTitle = |Lists|
!vshap = |width 15 height 6|
list .lst1 |Single Array List| call |!this.value()| SINGLE ZEROSEL $!vshap
list .lst2 |Multi-Select List| MULTI $!vshap
list .lst3 |Appending List| call |!this.append()| $!vshap
list .lst4 |Multi-column List| $!vshap
exit

define method .traExampleLists()
do !n from 1 to 5
!values[!n] = |Number| & !n
!rtext[!n] = !n.string()
do !m from 1 to 2
!multi[!n][!m] = |col | & !m & | (| & !n & |)|
enddo
enddo
!this.lst1.dtext = !values
```

```
!this.lst1.rtext = !rtext
!this.lst2.dtext = !values
!single[1] = |Click to append values|
!this.lst3.dtext = !single
!this.lst4.setRows(!multi)
!heading = |One Two|
!this.lst4.setHeadings(!heading.split())
endmethod

define method .value()
!dtext = !this.lst1.selection('Dtext')
!rtext = !this.lst1.selection('Rtext')
$p Selected Dtext = $!<dtext>; Rtext = $!<rtext> (hidden!)
endmethod

define method .append()
!nextLine = !this.lst3.dtext.size() + 1
!val = |Appended | & !nextLine
!this.lst3.add(!val)
endmethod
```



Investigate the lists and how the gadget definition affects how they work. What values are printed the command window when value is selected.



Refer to the Reference Manual and Guide for more information about list gadgets.

2.11 Frame Gadgets

A FRAME is a cosmetic gadget used to surround a group of similar gadgets. This helps with organisation, positioning and user experience. A frame can also be declared as a FOLDUP, PANEL, TABSET or even a TOOLBAR (when defined for the main AVEVA E3D window).

Show the example by typing **show !!traExampleFrames**.

```
setup form !!traExampleFrames dialog resizable
!this.formTitle = |Frames|
frame .tabset TABSET anchor ALL wid 15 hei 5
frame .f1 |Dock| at 0 0 dock FILL
frame .fA |Dock All| dock FILL
frame .fB |Dock Right| dock RIGHT wid 10
frame .fC |Dock Bottom| dock B hei 4
exit
exit
exit
exit
frame .f2 |Foldup| at 0 0 dock FILL
button .but1 linklabel |Show Panel Frame...| at 0 0 call |!this.showD()|
frame .fD panel |Panel Frame| at x0 ymax.but1 dock FILL
frame .fE foldup |Foldup 1| at x1 ymin.fD + 1 anchor t+l+r wid 20 hei 3
button .but2 linklabel |Foldup Panel| anchor t call |!this.fold()|
exit
```



```

frame .fF foldup |Foldup 2| at xmin.fE ymax.fE anchor t+l+r wid 20 hei 3
para .para1 text |Inside a frame|
exit
    exit
exit
frame .f3 |Other| at 0 0 dock FILL
!pos = |at xcen.fG - 0.5 * size ymax|
frame .fG |Deactive| anchor t+l+r wid 21 hei 4
    button .but3 |Try and click!| at x 2 y 1 anchor t
    exit
    button .but4 toggle |Show Frame| $!pos backg 8
anchor t call |!this.showF()|
    frame .fH |Hidden Frame| at xmin.fG ymax.but4 anchor
all width.fG hei.fG
    exit
    exit
    exit
    exit

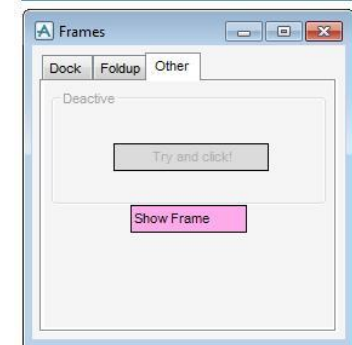
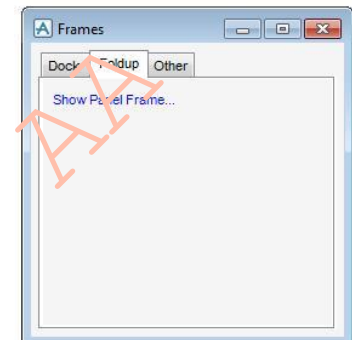
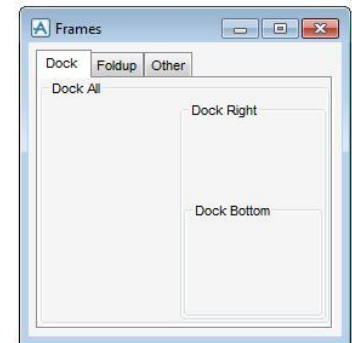
define method .traExampleFrames()
    !this.fD.visible = FALSE
    !this.fE.expanded = FALSE
    !this.fG.active = FALSE
    !this.fH.visible = FALSE
Endmethod

define method .fold()
    !this.fE.expanded = FALSE
endmethod

define method .showD()
    if !this.but1.val then
--(continued onto next page)
        !this.fD.visible = TRUE
        !this.but1.tag = |Hide Panel Frame...|
    else
        !this.fD.visible = FALSE
        !this.but1.tag = |Show Panel Frame...|
    endif
endmethod

define method .showF()
    if !this.but4.val then
        !this.fH.visible = TRUE
        !this.but4.tag = |Hide Frame|
    else
        !this.fH.visible = FALSE
        !this.but4.tag = |Show Frame|
    endif
endmethod

```



The example shows the various types of frames and the members/methods available. Notice how the frames immediately below the **TABSET** are its tabs.

i When creating a FRAME gadget, for every FRAME there must be an associated EXIT.

If insufficient **EXITs** are provided, this may cause an error and the form MAY NOT LOAD. As the error occurred inside the FORM DEFINITION, the command line will still be in form definition mode and will not function as usual. To exit this mode, type EXIT on the command line until an ERROR is received. This will mean that form definition mode has been exited and normal commands will work again.

i It is good practise to inset code inside a frame block by 2 spaces. This provides an easy way to spot missing exits and makes the code easier to read.

 Refer to the Reference Manual and Guide for more information about frame gadgets.

2.12 Textpane Gadgets

A **TEXTPANE** gadget provides an area on a form into which a user may edit multiple lines of text and cut/paste from elsewhere on the PML screen. The inputted value is stored as an **ARRAY of STRINGS** and can be set and read from the gadget.

Show the example by typing **show !!traExampleTextpanes**

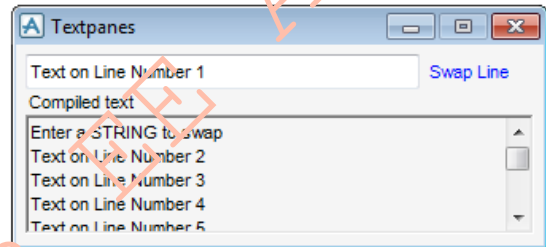
```

setup form !!traExampleTextpanes
  !this.formTitle = |Textpanes|
  !this.initCall = |!this.init()|
  text .txt1 width 30 is STRING
  button .but1 linklabel |Swap Line| at xmax.txt1 + 0.5 y 0 call |!this.swap()|
wid 7
  textpane .txtp |Compiled text| at x 0 ymax wid 40 hei 5
exit

define method .init()
  !this.txt1.val = |Enter a STRING to swap|
  do !n from 1 to 10
    !val[!n] = |Text on Line Number | & !n
  enddo
  !this.txtp.val = !val
endmethod

define method .swap()
  !lineNO = !this.txtp.curPos()
  !line = !this.txtp.line(!lineNO[1])
  !this.txtp.setLine(!lineNO[1],
!this.txt1.val)
  !this.txt1.val = !line
endmethod

```



Choose different lines inside the textpane and press the button. The method reads the cursor position inside the gadget and it will swap that line. The textpane cannot be given a callback so any interaction has to be through something else.

 *Refer to the Reference Manual and Guide for more information about textpane gadgets.*

2.13 Option Gadgets

An **OPTION** gadget offers a single choice from a list of items. It may contain either **PIXMAPS** or **STRINGS**, but not a mixture. The gadget displays the **CURRENT** choice in the list. When the user presses the option gadget, the entire set of items is shown as a drop-down list and the user can then select a new item by clicking on the option required. A **COMBO** gadget allows the user to type in a value as well as choose one from the drop down. If used in conjunction with an open callback (explained in section 2.19), it can select from the list for the user. This would be very useful when there are a long number of options.

The width of a text option gadget must be specified. A tag name is optional and is displayed to the left of the gadget. The available options are stored as an **ARRAY of STRINGS** as the gadgets **DTEXT** or **RTEXT** and can be updated by altering this array.

Show the example by typing **show !!traExampleOptions**.

```

setup form !!traExampleOptions
  !this.formTitle = |Options|
  path right
  option .opt1 |Normal| tagwidth 6 width 7
  combo .opt2 |Combo| tagwidth 6 width 7
  option .opt3 |Pixmap| tagwidth 6 pixmap width 16 height 16
exit

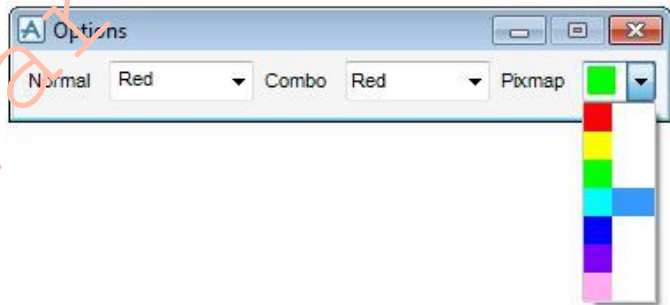
define method .traExampleOptions ()

```

```
!dtext = |Red Yellow Green Cyan Blue Violet Pink|
!this.opt1.dtext = !dtext.split()
!this.opt2.dtext = !dtext.split()
!files[1] = !!pml.getPathName(|red16.png|)
!files[2] = !!pml.getPathName(|yellow16.png|)
!files[3] = !!pml.getPathName(|green16.png|)
!files[4] = !!pml.getPathName(|cyan16.png|)
!files[5] = !!pml.getPathName(|blue16.png|)
!files[6] = !!pml.getPathName(|violet16.png|)
!files[7] = !!pml.getPathName(|pink16.png|)
!this.opt3.dtext = !files
!this.opt3.rtext = !dtext.split()
!this.opt2.callback = !!this.setColour(|Red|)
endmethod

define method .setColour(!gad is gadget, !event is STRING)
  if !event.eq('VALIDATE') then
    !userInput = !this.opt2.displayText()
    do !n index !this.opt2.dtext
      !chrs = !userInput.length()
      !test = !this.opt2.dtext[!n].upcase().substring(1, !chrs)
      if !userInput.upcase().eq(!test) then
        !this.opt2.val = !n
        break
      endif
    enddo
  endif
endmethod
```

A **COMBO** gadget allows values to be typed in as well as picked. In this example, an **OPEN CALLBACK** method is used to select the nearest available option in the list. Test the **COMBO** gadget by typing in part of the required colour and press enter.



① With a **COMBO** box, a user entered value will only be processed if an **OPEN CALLBACK** is included to complete the action

Try querying the `!rtext` of the pixmap gadget by using a built-in gadget method:

```
q var !!traExampleOptions.opt3.selection()
```

2.14 Toggle Gadgets

TOGGLE gadgets are used for independent on/off settings. This means they are used in situations where the user has two choices i.e. is it bold or not? Is it on or off? The state of the toggle is stored under its `val` member, but can also store different descriptions for selected and unselected. **NUMERIC INPUT** gadgets can be used to allow users to enter real values within a range. The user can also use the supplied toggles to alter the value by the defined step.

Show the example by typing **show !!traExampleToggles**.

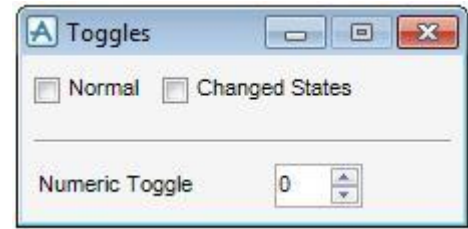
```
setup form !!traExampleToggles
!this.formTitle = |Toggles|
toggle .tog1 tagwid 5 |Normal| call !!this.state(!this.tog1)
toggle .tog2 tagwid 10 |Changed States| call !!this.state(!this.tog2) | states |N| |Y|
line .line at xmin.tog1 ymax.tog1 horiz wid 21 hei 1
```

```

    numeric .num |Numeric Toggle| at xmin.togl ymax.line range 0 10 step 1 NDP 0 wid 3
exit

define method .state(!gad is GADGET)
    if !gad.val then
        q var !gad.onvalue
    else
        q var !gad.offvalue
    endif
endmethod
endmethod

```



Click the toggles and observe the values on the command line. Notice how a **LINE** gadget has been used to divide the form

Refer to the Reference Manual and Guide for more information about toggle and numeric gadgets.

2.15 Radio Gadgets

A **RADIO GROUP** allows a user to make a single choice from a fixed number of choices. Two objects can be used to define a radio group: **RGROUP** or **RTOGGLE**. An **RGROUP** element defines the entire object and the tags contained; an **RTOGGLE** is contained within a normal **FRAME** object. **RGROUP** objects can be displayed vertically or horizontally. **RTOGGLE** objects can be arranged within a FRAME as required and can be placed alongside other gadgets

An **RGROUP** object has been deprecated and maybe withdrawn in the future. Use an **RTOGGLE** in preference for new and upgrading code.

Show the example by typing **show !!traExampleRGroups**.

```

setup form !!traExampleRGroups
    !this.formTitle = |Radio Groups|
    rgroup .vert |RGroup| FRAME vertical callback !!this.rg(!this.vert)|
        add tag |Left| select |L|
        add tag |Centre| select |C|
        add tag |Right| select |R|
    exit
    frame .rtog |RToggle| at xmax + 1 y 0
        path down
        rtoggle .left |Left| states || |L|
        rtoggle .cen |Centre| call |q var !this.cen.onvalue| at ymax - 0.1 states ||
|C|
        rtoggle .right |Right| at ymax - 0.1 states || |R|
    exit
exit

define method .traExampleRGroups()
    !this.rtog.callback = !!this.rt(!this.rtog)|
endmethod

define method .rg(!gad is GADGET)
    q var !gad.selection()
endmethod

define method .rt(!gad is GADGET)
    !rtog = !gad.rtoggle(!gad.val)
    q var !rtog.onvalue
endmethod

```



Refer to the Reference Manual and Guide for more information about radio toggle gadgets.

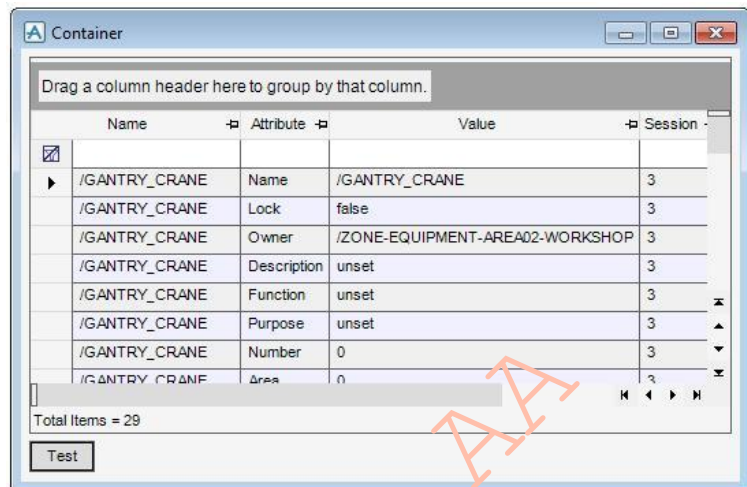
2.16 Container Gadgets

A CONTAINER gadget is a place holder for a PML .NET control. The PML .NET controls are objects developed in a .NET language, compiled into a .dll file and hosted by the PML form.

AVEVA E3D is supplied with some example controls that are used within standard product but may be used in any customisation. These include:

- A Grid Control
- A Database Explorer
- A Database Search
- A File browser

The PML .NET controls are explained further later in the guide with examples of their definition and use. Examples of container gadgets will be included then.

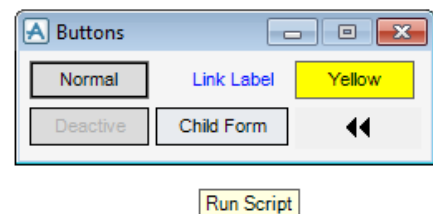


It is possible to develop customised controls using C# and this is cover by Training guide TM-2268

2.17 Tooltips

Tooltips are small help boxes which pop up when you move the mouse over an active gadget. Tooltips are typically used to provide more information to the user (for example, on a pixmap button). An example of the syntax is as follows:

```
BUTTON .B1 |SAVE| Pixmap TOOLTIP |Run Script|
```

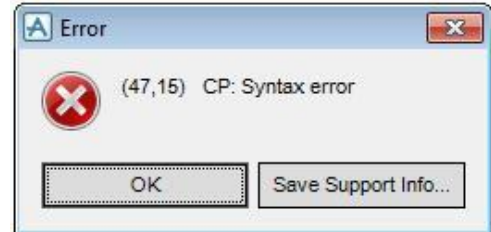


Refer to the specific gadges in the AVEVA E3D Software Customisation Reference Manual

Exercise 1 - Working with a PML form

- You have been provided with a form called !!c2ex1. To show the form, type show !!c2ex1

- The form has not been maintained and there are a number of bugs in the form. At least 3 are preventing it from being shown and at least 2 are causing the wrong information to be displayed.
- Debug the form so that it can be shown and displays the correct information.



The fixes are indicated in Appendix B

- The form demonstrates poor gadget arrangement, a problem that impacts the user experience. Work through the gadgets and improve their arrangement and sizes to improve the form.
- An example of the aligned form is shown in the below screenshot.

The screenshot shows a form titled 'Example Form - Exercise 1'. It is divided into two main sections: 'Inputs' and 'Results'.
Inputs Section:
 - 'Temperature conversion (Input)': A text box for 'Temperature' and radio buttons for '°C' (selected) and '°F'.
 - 'Temperature Range': Three text boxes for 'Minimim', 'Maximum', and 'Step Size'. A blue link 'Fill with °F to °C >>' is next to the 'Step Size' box.
 - 'Temperature Split': Two text boxes for 'Input' and 'Delimiter'. A blue link 'Split temperature >>' is next to the 'Delimiter' box.
Results Section:
 - 'Temperature conversion (Output)': A text box for 'Temperature' with a '°F' label.
 - 'Temperature Results': A table with columns 'No.', 'Celsius', '=', and 'Fahrenheit'.
 - 'Temperature Split Result': Two text boxes for 'No. of Temp' and 'Result'.

- Write a method that will run when the form is shown. This method should be used to fill default values into the input frame.
 - When the input values are set, the method should then run the available methods that fill in the output frame.
- Add a new button to the form that will fill the temperature conversion chart with Fahrenheit to Celsius values.
 - Consider how best to position the button.

A close-up of the 'Temperature Range' section of the form. It contains three text boxes: 'Minimim' with value '0', 'Maximum' with value '100', and 'Step Size' with value '25'. To the right of the 'Maximum' box is a blue link 'Fill with °C to °F >>'. To the right of the 'Step Size' box is a blue link 'Fill with °F to °C >>'.

5. • Add a method to the new button that will perform the conversion.
- A method has already been written in the form. Find the method and set the button callback.

The screenshot shows two panels. The 'Temperature Range' panel has three input fields: 'Minimum' (0), 'Maximum' (100), and 'Step Size' (25). There are two buttons: 'Fill with °C to °F >>' and 'Fill with °F to °C >>'. The 'Temperature Results' panel shows a table with 3 rows and 4 columns: 'No.', 'Fahrenheit', '=', and 'Celsius'. The data is as follows:

No.	Fahrenheit	=	Celsius
1	0	=	-17.78
2	25	=	-3.89
3	50	=	10.00

6. • There is currently no method available to split the temperature string the in lower frames.
- Write a method to do the following:
 - Split the input string based on the delimiter
 - Read the individual temperatures and convert them (this will depend on °C or °F)
 - Compile the converted values as a space separated string
 - Write the compiled string back to the form and display how many temperatures
- This method should be run when the form is shown or the button is pressed.

The screenshot shows two panels. The 'Temperature Split' panel has two input fields: 'Input' (10°C/30°F/20°C/5°F) and 'Delimiter' (/). There is a button 'Split temperature >>'. The 'Temperature Split Result' panel has two input fields: 'No. of Temp' (4) and 'Result' (50°F -1°C 68°F -15°C).

7. • Test the form and check all features work.

An example of the completed form can be found in Appendix B

The screenshot shows a completed form titled 'Example Form - Exercise 1'. It has two main sections: 'Inputs' and 'Results'. The 'Inputs' section contains three sub-panels: 'Temperature conversion (Input)' with a 'Temperature' field (0) and radio buttons for °C (selected) and °F; 'Temperature Range' with 'Minimum' (0), 'Maximum' (100), and 'Step Size' (25) fields, and buttons 'Fill with °C to °F >>' and 'Fill with °F to °C >>'; and 'Temperature Split' with 'Input' (10°C/30°C/20°C/5°C) and 'Delimiter' (/) fields, and a 'Split temperature >>' button. The 'Results' section contains two sub-panels: 'Temperature conversion (Output)' with a 'Temperature' field (32) and a radio button for °F; and 'Temperature Results' with a table showing conversions for 0, 25, and 50 degrees Celsius to Fahrenheit. The 'Temperature Split Result' panel shows 'No. of Temp' (4) and 'Result' (50°F 86°F 68°F 41°F).

No.	Celsius	=	Fahrenheit
1	0	=	32.00
2	25	=	77.00
3	50	=	122.00

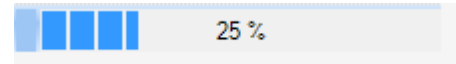
2.18 Progress Bars and Interrupting Methods

The standard AVEVA E3D global object **!!FMSYS** (Forms & Menus System) provides two pieces of functionality which can be applied when designing a form:

- An progress bar (appearing at the bottom right of the main program)
- The ability to interrupt a PML method

The progress bar is activated by passing a real number to the **.setProgress()** method. For example:

!!FMSYS.setProgress(25)



Passing an argument of zero will cause the progress bar to disappear

To identify a gadget which can be used to interrupt a method, the gadget must be passed as an argument to the **.setInterrupt()** method. This method can then check the **!!FMSYS** object to see if the gadget has been pressed through its **.interrupt()** method

Show the example by typing **show !!traExampleProgressBar**.

```
setup form !!traExampleProgressBar dialog NoAlign
  button .b1 at xmin ymax call |!this.counter()| pixmap wid 80 hei 80
  numeric .num at xmax - size ymax + 0.1 range 0 10000 step 1 ndp 0 val 0 wid 4
exit

define method .exampleProgressBar()
  !this.b1.addPixmap (!!pml.getpathname('gobutton.png'))
endmethod

define method .counter()
  !!FMSYS.setInterrupt (!!exampleProgressBar.b1)
  !this.b1.addPixmap (!!pml.getPathName('stopbutton.png'))
  !this.b1.refresh()
  do !n from 1 to 10000
    !this.num.val = !n
    !this.num.refresh()
    !percent = !n / 100
    -- Check if the method has been interrupted. Break if it has
    if (!!FMSYS.interrupt()) then
      !!alert.message(!n & | loops were completed - | & !percent.string(|D1|) &
|%)
      break
    endif
    !!FMSYS.setProgress(!percent)
  enddo
  !this.b1.addPixmap (!!pml.getPathName('gobutton.png'))
  !!FMSYS.setProgress(0)
endmethod
```

Press the start button and press it again to stop the method early



2.19 Open Callbacks

An **OPEN CALLBACK** is a way of providing change information about the gadget the user is interacting with. This means that for every interaction event, the callback will be called. Two pieces of information are supplied during an open callback:

- the gadget being interacted
- a keyword (as a STRING).

These keywords indicate the event which caused the callback. Different keywords will be generated by different gadgets under different circumstances e.g. a multi-selection gadget may have a '**SELECT**' '**UNSELECT**', as well as the more standard '**START**' '**STOP**'

AVEVA E3D will recognise an open callback if a method only has one bracket e.g. call `!this.opencall(|`

For an open callback to work, there must be an appropriate method defined. An open callback method must be able to receive two arguments (1) a gadget and (2) a keyword

e.g. define method `.opencall(!a is GADGET, !b is STRING)`

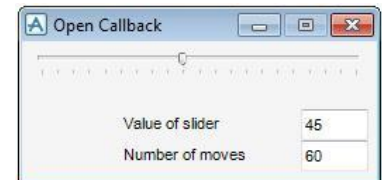
As the method is called at every significant event, the method should be written to recognise keywords. The following example has been written around the slider gadget.

Show the example by typing **show !!traExampleOpenCallback.**

```
setup form !!traExampleOpenCallback
!this.formTitle = |Open Callback|
slider .slide anchor L+R horizontal range 0 100 step 5 val 50 width 30
text .text |Value of slider| at xmax - size ymax + 1 anchor B+R width 5 is
REAL
text .moves |Number of moves| at xmax.text - size ymax anchor B+L width 5 is
REAL
member .store is REAL
exit

define method .traExampleOpenCallback ()
!this.slide.callback = !!this.slideMove(|
!this.moves.val = 0
!this.store = 0
!this.text.val = !this.slide.val
endmethod

define method .slideMove(!gad is GADGET, !val is STRING)
!this.text.val = !gad.val
!this.text.refresh()
if !val.eq(|MOVE|) then
!this.store = !this.store + 1
elseif !val.eq(|STOP|) then
!this.moves.val = !this.store
!this.store = 0
endif
endmethod
```



Investigate the form by sliding the slider. Review the methods to identify how it works. Also, revisit the `!!traExampleOptions` to review that open callback.

The `.refresh()` method has been applied to the `.text` gadget to ensure its value updates as the slider is moved. The number of moves is stored as a member of the form. This saves a new global variable from being defined and means the increasing value can be shared between the open callbacks.

i An open callback can be given to any gadget that accepts a callback. Different gadgets may generate different event keywords

2.20 Menus

Menu objects are used to provide options to a user and can be applied to a form as either a:

- Menu bar across the top of the form
- Popup menu on a gadget

In both cases, a menu object must be defined to represent the options part of the menu bar or popup menu.

2.20.1 Defining a Menu Object on a Form

Within the form definition, the form method `.newMenu()` creates a named menu object. Once the object is defined, the `.add()` method on the menu object can be used to add named menu fields. A menu field can do one of three things:

- Execute a callback
- Display a form
- Display a sub-menu

You can also add a visual separator between fields.

```
!menu = !this.newmenu(|exampleMenu|, |MAIN|)
!menu1.add(|CALLBACK|, |Query|, |q atts|)
!menu1.add(|FORM|, |Progress Bar...|, |exampleProgressBar|)
!menu1.add(|SEPARATOR|)
!menu1.add(|MENU|, |Pull-right1|, |Pull1|)
```

This creates a menu object called `exampleMenu` with 3 fields (`Query`, `Hello...` and `Pull-right1`) and a separator between the last two fields

- The `Query` field when picked will execute the **CALLBACK** command 'q att' (prints the CE attributes to the command window)
- The `Hello...` field when picked will load and display the **FORM** from the previous section `!exampleProgressBar`. By convention, the text on a menu field leading to a form ends with three dots, which you must include with the text displayed for the field.
- The **SEPARATOR**, usually a line, will appear after the previous field.
- The `Pull-right1` **MENU** field when picked will display the sub-menu `!this.Pull1` to its right. A menu field leading to a sub-menu ends with a `>` symbol (this is added automatically)

2.20.2 Defining a Bar Menu on a Form

Forms may have a bar menu gadget which appears as a row of options across the top of the form. A bar menu is defined within form definition and specifies the options the user has to choose from. There are three types that can be added:

- Choose – displays a user-defined menu object (reference by its name)
- Window – displays a list of the open forms
- Help – displays the standard help options

After the bar command, use the bar object method `.add()` to add extra options to the bar menu. As there can only be one bar menu, `!this.bar` refers to the bar menu. For example:

```
bar
!this.bar.add(|Choose|, |exampleMenu|)
!this.bar.add(|Window|, |Window|)
!this.bar.add(|Help|, |Help|)
```

 *Bar menus can only be added to forms which cannot dock (i.e. dialog docking)*

2.20.3 Defining a Popup Menu on a Form

Pop-up Menu (or Context Menu) is a MENU object displayed from a gadget by right-clicking on it. This is a useful method of providing the user with relevant functionality relating to the associated gadget.

```
!this.exampleList.setPopup(!this.exampleMenu)
```

The MENU object is applied to a gadget through the `.setPopup()` method (available on a number of gadgets). The MENU object can be applied at any time, so popup menus can change to ensure the content is always relevant.

2.20.4 Adding Menu Objects to a Form

Show the example by typing **show !!traExampleMenus**.

```

setup form !!traExampleMenus
  !this.formTitle = |Menus|
  !this.initcall = |!this.init()|
  bar
  !this.bar.add(|Choose|, |exampleBarMenu|)
  !this.bar.add(|Window|, |WINDOW|)
  !this.bar.add(|Help|, |HELP|)

  list .list callback |!this.pickList()| wid 20 hei
  12

  !menu = !this.newMenu(|exampleBarMenu|)
  !menu.add(|CALLBACK|, |Query|, |q atts|)
  !menu.add(|FORM|, |Progress Bar...|,
|exampleProgressBar|)
  !menu.add(|SEPARATOR|)
  !menu.add(|MENU|, |Pull-right1|, |Pull1|)

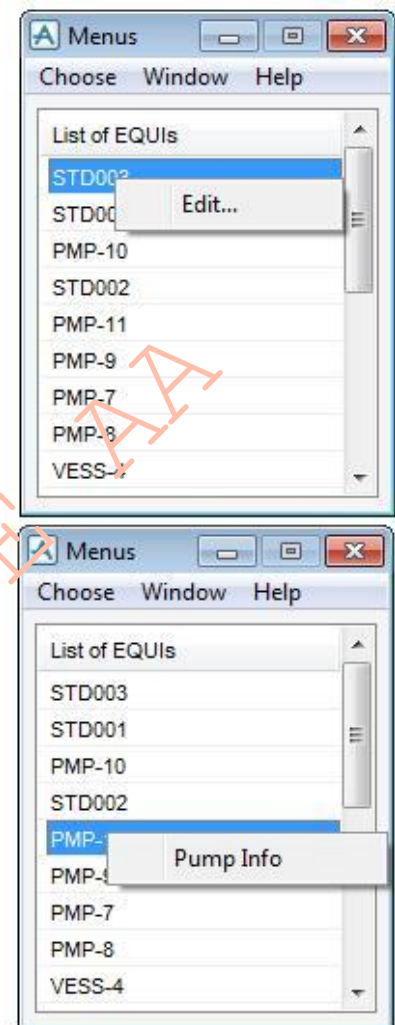
  !menu = !this.newMenu(|exampleEquiMenu|)
  !menu.add(|CALLBACK|, |Edit...|, |$p equipment
specific|)

  !menu = !this.newMenu(|examplePumpMenu|)
  !menu.add(|CALLBACK|, |Pump Info|, |$p pipe
specific|)
exit

define method .init()
  !titles = |List of EQUIs|
  !this.list.setHeadings(!titles.split(|/|))
  -- Collect the equipment from the current element
  !ce = !!ce.name
  var !equipment coll all EQUI for $!ce
  var !equipmentNames eval FLNN for all from
!equipment
  !this.list.dtext = !equipmentNames
  !this.list.rtext = !equipment
endmethod

define method .pickList()
  -- Set the popup menu based on the first letter of
name
  if
  !this.list.selection(|DTEXT|).substring(1,1).eq(|P|)
  then
    !this.list.setPopup(!this.examplePumpMenu)
  else
    !this.list.setPopup(!this.exampleEquiMenu)
  endif
endmethod

```



Notice how the context menu changes depending on the choice in the list. This allows specific functionality to be available based on the choice of the user.



Refer to the AVEVA E3D Customisation Reference Manual for the details of which gadgets permit a popup menu.

2.21 User-Defined Form Members

As a **FORM** is an **OBJECT**, it may be given user-defined **MEMBERS**. This is a useful way to store data, effectively creating global variables. The benefit of this method is that data is global without having large numbers of individual variables.

Form members replaces the previous method of USERDATA (PML 1 style data storage), and are given an object-type. These variables have the same lifetime as the form and are deleted when the form itself is unloaded.

Show the example by typing **show !!traExampleFormMembers**.

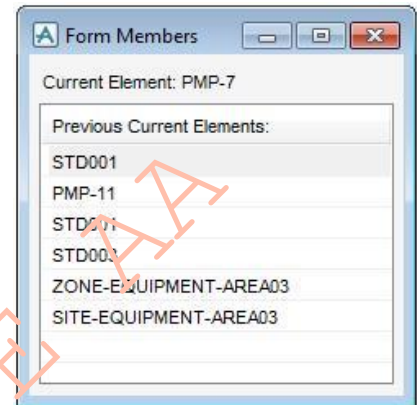
```
setup form !!traExampleFormMembers resize
  !this.formTitle = |Form Members|
  !this.initcall = !!this.init()
  track |DESICE| call !!this.update()
  para .ceName anchor t+l+r wid 20 hei 1
  list .stored || at x 0 ymax anchor all wid 25 hei 10
  member .ceRef is DBREF
  member .storage is ARRAY
exit

define method .traExampleFormMembers()
  !this.stored.callback = !!this.return()
  !headings = |Previous Current Elements:|
  !this.stored.setHeadings(!headings.split(|/|))
endmethod

define method .init()
  !this.ceRef = !!ce
  !this.ceName.val = |Current Element: |
  !!ce.attribute(|FLNN|)
endmethod

define method .update()
  !this.storage.insert(1, !this.ceRef)
  !block = object
  BLOCK(|!this.storage[!evalIndex].attribute('FLNN')|)
  !dtext = !this.storage.evaluate(!block)
  !this.stored.dtext = !dtext
  !this.init()
endmethod

define method .return()
  !!ce = !this.storage[!this.stored.val]
endmethod
```



It shows how a form can be assigned an element when shown and how this element can be updated. Every time the update button is pressed, the element is stored in the member .storage.

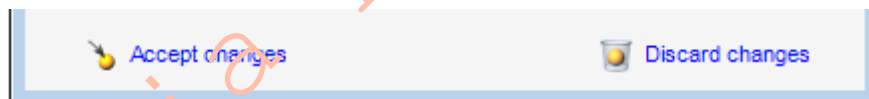
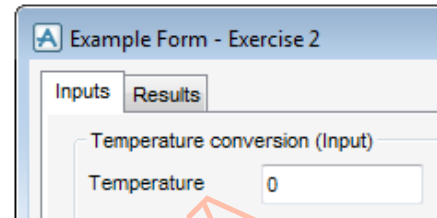
Investigate the information stored on the form members by typing:

```
q var !!traExampleFormMembers.ceRef
q var !!traExampleFormMembers.storage
```

- i** Notice how the form uses the **TRACK** syntax to follow the current element. This syntax works for other modules too. Just replace the **DESI** from **DESICE** with the module database type e.g. **PADDCE**, **CATACE** etc

Exercise 2 – Extending the form

1.
 - Save the form from the previous exercise as **c2ex2.pmlfrm** into the
 - Update form definition so the name, title and constructor method are correctly named.
 - Type **PML REHASH ALL** so AVEVA E3D finds the new file
2.
 - Upgrade the “Inputs” and “Results” frames into a single tabset
 - Consider what code is needed to define a tabset and how the existing frames will need to be updated.
 - Remove **ALL** callbacks from the existing gadgets and **ALL** buttons from the form.
 - The methods should now be called with an **OPEN CALLBACK** on the Results tab. When the Results tab is shown, all the existing methods should run.
3.
 - Add an **ARRAY** member to the form that will store the values of the input gadgets. This will provide the ability to reset the values if they are changed.
 - Write a method that will allow data to be applied to and taken from this member.
 - Consider how the code will get the values from the gadgets and how best to store them. What type of values will need to be stored?
4.
 - Add two buttons to the base of the form. One to update the stored values (Accept changes) and the other will apply the stored values to the gadgets (Discard changes)



- The buttons will be **linklabels** and should call the method to store/reset data. To the left of the buttons there should be **16x16 pixmap paragraph**.
- Consider how the gadgets will be positioned and how they will move if the form is resized.
- Using the **.addPixmap()** method, apply standard AVEVA E3D images to them.


```
.addPixmap(!pml.getPathName('accept.png'))
.addPixmap(!pml.getPathName('discard.png'))
```

 This gadget arrangement is in line with the new forms developed for AVEVA E3D

5. • As the buttons have been removed from the form, only one type of conversion can be applied to the List gadget (depending on which method the open callback calls)
- Add a **POPUP MENU** to the list gadget which will allow the other type of conversion to be completed.

Temperature Results

No.	Fahrenheit	=	Celsius
1	0	=	47.78
2	25	=	
3	50	=	10.00

Switch to °C = °F

Temperature Results

No.	Celsius	=	Fahrenheit
1	0	=	
2	25	=	
3	50	=	122.00

Switch to °F = °C

- There will need to be two new menu objects created on the form;
 1. to run the method as Celsius.
 2. to run the method as Fahrenheit.
 - After a choice has been made, the popup menu needs to be swapped so the user always sees the correct one.
 - Consider what methods will need to be called and how the popup menus can be managed.
6. • Test the form and check all features work.

An example of the completed form can be found in Appendix B

Example Form - Exercise 2

Inputs Results

Temperature conversion (Input)

Temperature 0 °C °F

Temperature Range

Minimum 0

Maximum 100 Fill with °C to °F >>

Step Size 25 Fill with °F to °C >>

Temperature Split

Input 10°C/30°C/20°C/5°C

Delimiter / Split temperature >>

Accept changes Discard changes

Example Form - Exercise 2

Inputs Results

Temperature conversion (Output)

Temperature 32 °F

Temperature Results

No.	Celsius	=	Fahrenheit
1	0	=	32.00
2	25	=	77.00
3	50	=	122.00

Temperature Split Result

No. of Temp 4

Result 50°F 86°F 68°F 41°F

Accept changes Discard changes

3 PML Objects

3.1 Built-in PML Object Types

PML objects allow values and methods to be bound together such that they can be considered as a single entity. The use of PML objects means that code can be standardised and reduced. There are a large number of standard objects which are supplied with AVEVA E3D

These include STRING, REAL, BOOLEAN, ARRAY, BORE, DIRECTION, DBREF, FORMAT, MDB, ORIENTATION, POSITION, FILE, PROJECT, SESSION, TEAM, USER, ALERT, FORM, all form Gadgets and various graphical aid objects.

Each object has a set of built in methods for setting or formatting the object contents. The following are the objects covered by the Software Customisation Reference Manual:

ARRAY	DATEFORMAT	FORMAT	REAL
BANNER	DATETIME	LOCATION	REPORT
BLOCK	DB	MACRO	SESSION
BOOLEAN	DBREF	MDB	STRING
BORE	DBSESS	OBJECT	TABLE
COLLECTION	DIRECTION	ORIENTATION	TEAM
COLUMN	EXPRESSION	POSITION	UNDOABLE
COLUMNFORMAT	FILE	PROJECT	USER

The following objects from the reference manual cover forms and menus:

ALERT	FORM	OPTION	TEXTPLANE
BAR	FRAME	PARAGRAPH	TOGGLE
BUTTON	LINE	RTOGGLE	VIEW ALPHA
COMBOBOX	LIST	SELECTOR	VIEW AREA
CONTAINER	MENU	SLIDER	VIEW PLOT
FMSYS	NUMERIC	TEXT	VIEW VOLUME

The following objects from the reference manual cover 3D geometry:

ARC	LOCATION	POINTVECTOR	PROFILE
LINE	PLANE	POSTEVENTS	RADIAL GRID
LINEARGRID	PLANTGRID	POSTUNDO	XYPOSITION



For further information about the individual objects, refer to the AVEVA E3D Software Customisation Reference Manual

3.2 Methods Available to All PML Objects

In addition to the object specific methods, there are methods that are common to all objects. The following table lists the methods available to all objects:

Name	Result	Purpose
.attribute(String)	ANY	To set or get a member of an object, providing the member name as a STRING.
.attributes()	ARRAY OF STRINGS	To get a list of the names of the members of an object as an array of STRING.
.delete()	NO RESULT	Destroy the object - make it undefined
.eq(any)	BOOLEAN	Type-dependent comparison
.lt(any)	BOOLEAN	Type-dependent comparison (converting first to STRING if all else fails)
.max(any)	ANY	Return the maximum of object and second object
.min(any)	ANY	Return the minimum of object and second object
.neq(any)	BOOLEAN	TRUE if objects do not have the same value(s)
.objectType()	STRING	Return the type of the object as a string
.set()	BOOLEAN	TRUE if the object has been given a value(s)
.string()	STRING	Convert the object to a STRING
.unset()	BOOLEAN	TRUE if the object does not have a value

 Refer to the Software Customisation Reference Manual for further details.

3.3 The FILE Object

The **FILE** object is an example of how older functionality has been replaced with a PML 2 style object. This object replaces the 'openfile' 'readfile' 'writefile' 'closefile' syntax and provides increased functionality (file path, if the file is open etc). It is now possible to read or write to a file in a single operation.

To create a file object:

```
!input = object file('C:\FileName')  
!output = object file('C:\FileName.out')
```

To open a file so it's available to be written to

```
!output.open('WRITE')
```

Available: READ, WRITE, OVERWRITE, APPEND

To close a file once finished

```
!output.close()
```

To check if the file is already open

```
q var !output.isOpen()
```

To find out when the file was last written to

```
q var !output.dtm()
```

3.3.1 Using FILE Objects

To read a line from file:

```
!line = !input.readRecord()
```

File must be open

To write a line to file:

```
!output.writeRecord(!line)
```

File must be open

To read all the input file:

```
!fileArray = !input.readFile()
```

Files are opened and Closed automatically

To write all of the data to file

```
!output.writeFile('WRITE',!fileArray)
```

Files are opened and Closed automatically

i The `.readFile()` method has a default maximum file size which it can read. This can be increased by passing the method a `REAL` argument (representing the number of lines in the file)

3.3.2 Opening a FILE Object in Notepad

The **SYSCOM** command can be used to open the file in an external program, such as Notepad. The syntax inside the string marks will be run as a windows command.

```
!file = object file(|C:\AVEVA\Plant\Training\pml\lib\forms\c2ex1.pmlfrm|)  
syscom |C:\Windows\system32\notepad.exe $!file&|
```

By using the command **START**, the file will open in the Windows default program

```
syscom |start $!file&|
```

i The "&" to the end of the line will open the file as a new process. Without it, the AVEVA E3D session will suspend until the external program is closed.

3.3.3 Using the Standard File Browser

AVEVA E3D is supplied with a standard file browser that can be used to load forms. The best way to load the form is to use the **!!fileBrowser** function it will initialise the browser (based on the arguments).

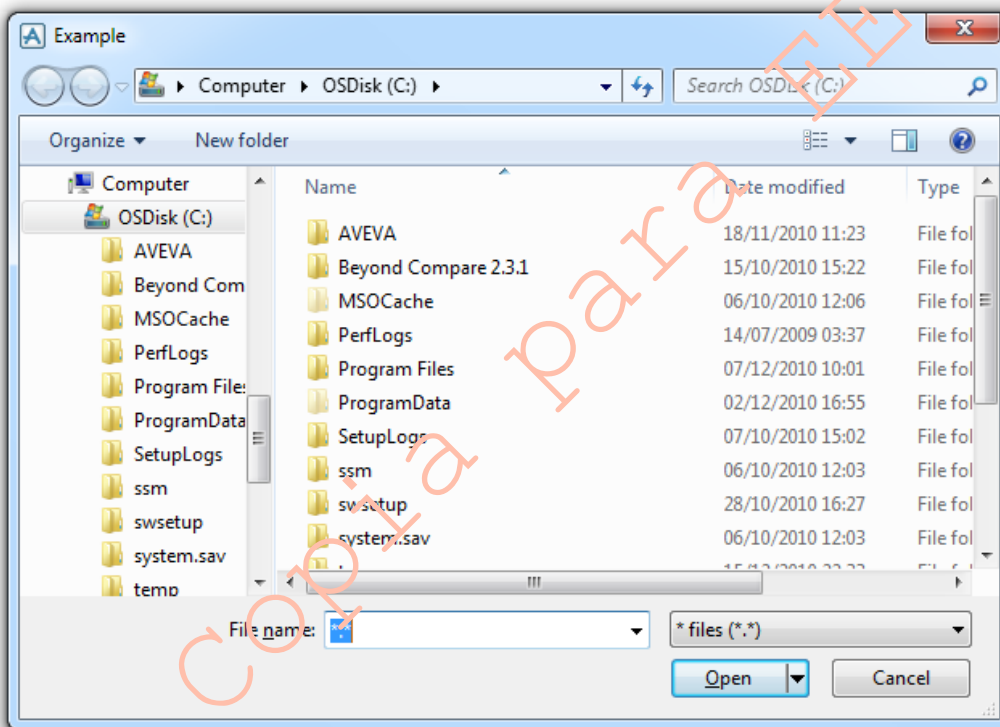
To show the browser form, type the following:

!!fileBrowser('c:\','*.*','Example',true,'q var !!fileBrowser.file')

Where the arguments are:

- Initial file path for the browser form.
- File type filter.
- Title for the browser form.
- Does the file need to exist? Boolean - used to differentiate between Open and Save
- The Callback on the action button of the browser form.

The standard windows file browser should be displayed, defaulting to the C:\



Navigate to a file and click **Open**, a file object should be printed to the command window:

<FILE> C:\AVEVA\Plant\Training\pml1lib\forms\c2ex1.pmlfrm

Once a file is chosen, it is held as the .file member of the **fileBrowser** form and is available for use.

Show the example by typing **show !!traExampleFile**. The example demonstrates how the object can be used to read a file, gain information about it and even show it.

```

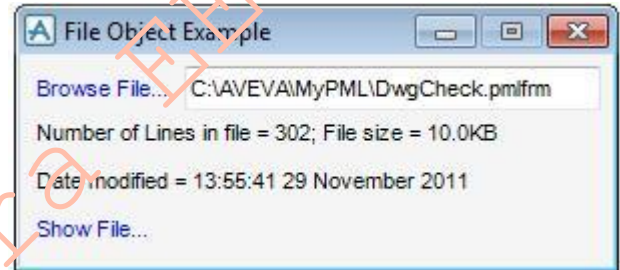
setup form !!traExampleFile
  !this.formTitle = |File Object Example|
  button .browse linklabel |Browse File...| wid 8
  text .fileName || call !!this.read(object file(!this.txt1.val), 1)| width 25
is string
  para .par1 at x 0 ymax text || width 35
  para .par2 at x 0 ymax text || width 35
  button .load linklabel |Show File...| at x 0 ymax call !!this.load()| wid 8
  member .file is FILE
exit

define method .traExampleFile()
  !this.browse.callback = !!!fileBrowser('C:\AVEVA\Plant\Training\pml\lib\forms',
  '*.pmlfrm', $
  'Load File', TRUE, '!!traExampleFile.read(!!fileBrowser.file, 2)')|
  !this.load.visible = FALSE
endmethod

define method .read(!file is file, !flag is REAL)
  !this.file = !file
  if !flag.eq(2) then
    !this.fileName.val = !file.string()
  endif
  !file.open('READ')
  !n = 0
  do
    !line = !file.readRecord()
    break if !line.unset()
    !n = !n + 1
  enddo
  !file.close()
  !this.load.visible = TRUE
  !date = !file.dtm()
  !size = (!file.size() / 1000)
  !fileSize = |File size = | & !size.string(|d1|) & |KB|
  !this.par1.val = |Number of Lines in file = | & !n & |; | & !fileSize
  !this.par2.val = |Date modified = | & !date
endmethod

define method .load()
  !file = !this.file
  syscom |start $!file&|
endmethod

```



Try showing the file and see how Windows displays it. Try updating the example so that other file types can be opened. What programs are used to display these files.

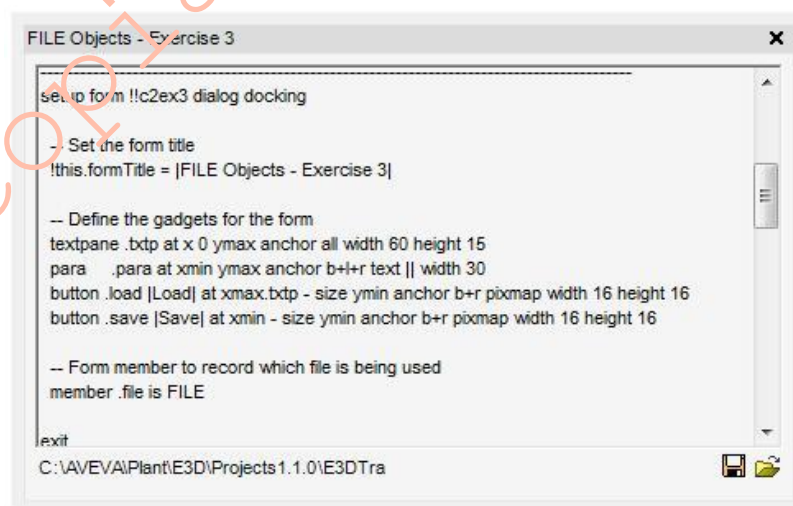
 Refer to the Reference Manual and Guide for more information about radio toggle gadgets.

Exercise 3 – Using the FILE Object

1.
 - Create a new form called **!!c2ex3** and save it to the PMLLIB area.
 - The purpose of this form will be to use the **FILE** object and a **standard file browser** to turn a **textpane** into a text editor.
2.
 - The form will require a textpane for the text to be displayed and two buttons to load and save with. Consider using a paragraph gadget to provide feedback about the file being used.



- What file types should be able to be loaded? What happens if a non-ascii file type is opened? Can the user save a blank file?
3.
 - Test the form with a range of files.
 - What other features could be added to the form to enhance the user experience? Investigate the FILE object and TEXTPANE gadget further to see if anything more can be added.



 An example of the completed form can be found in Appendix B

4 Collections

A very powerful feature of the AVEVA E3D database is the ability to collect and evaluate data according to rules. There are two available methods for collection that are valid in PML. The first is a command syntax PML 1 style and the other is with a PML **COLLECTION** object. Both are still valid, but when developing new PML, a **COLLECTION** object should be used in preference,

4.1 COLLECT Command Syntax (PML 1 Style)

The COLLECT syntax is based around three specific pieces of information:

- What element type is required?
- If specific elements are required, what aspect can be used to identify them?
- Which part of the hierarchy to look it?

If you wish to collect all the EQUI elements for the current ZONE, type the following:

```
var !equipment collect all EQUI for ZONE
q var !equipment
```

If you wish to collect all the piping components owned by a specific ERAN, type the following:

```
var !pipeComponents collect ALL with owner eq /200-B-4-B1 for SITE
q var !pipeComponents
```

if you wish to collect all the BOX primitives below the current element, type the following:

```
var !boxes collect all BOX for ce
q var !boxes
```

 You do not need to specify level of the hierarchy to search, if left out the entire MDB will be searched

 For more examples, refer to Chapter 11 of the Database Management Reference Manual and 2.3.10 Design Reference manual general commands

4.2 EVALUATE Command Syntax (PML 1 Style)

After elements have been collected through the **COLLECT** syntax, they are stored as an ARRAY of STRINGS. This array (like any array) can be processed using the **EVALUATE** syntax to provide further information

To get the names of all the elements held in **!equipment**, type the following:

```
var !equipmentNames evaluate NAME for ALL from !equipment
q var !equipmentNames
```

To get the fullnames of the elbows held within **!pipeComponents**, type the following:

```
var !elbowNames evaluate FLNN for all ELBO from !pipeComponents
q var !elbowNames
```

To get the names (without the leading slash) of the pumps in **!equipment**, type the following:

```
var !pumpNames evaluate NAMN for ALL with MATCHWILD(NAMN, [P*]) from !equipment
q var !pumpNames
```

To get the volume of all the boxes in **!boxes**, type the following:

```
var !volume eval (xlen * ylen * zlen) for ALL from !box
q var !volume
```

 Notice how the expressions are command syntax and must return a BOOLEAN. Refer to the Software Customisation Reference Manual for more examples

4.3 COLLECTION Object (PML 2 Style)

A **COLLECTION** object is an example of a PML object that has directly replaced command syntax. It is recommended that all new PML uses **COLLECTION** objects as required. One advantage of using a **COLLECTION** object is that an **ARRAY OF DBREF** objects is returned.

A **COLLECTION** object is assigned to a variable in the same way as other objects:

```
!collection = object COLLECTION()  
q var !collection  
q var !collection.methods()
```

Once assigned, the methods on the object are used to set up the parameters of the collection. To set the element type for the collection to only **EQUI** elements, type the following:

```
!collection.type(EQUI)
```

If more than one element type is required, the following could be typed:

```
!elementTypes = [EQUI BRAN SCTN]  
!collection.type(!elementTypes.split())
```

The scope of the collection must be a DBREF object and should be passed as an argument to the **.scope()** method. For example, type the following:

```
!collection.scope(!ce)
```

To filter the results, the **.filter()** method must be passed an **EXPRESSION** object. For example, to filter to members of a named branch, type the following:

```
!expression = object EXPRESSION([name of owner eq '/200-B-4-E1'])  
!collection.filter(!expression)
```

Once the collection object has been set up, the **.results()** is used to return the collected elements as an **ARRAY of DBREF** objects.

```
!results = !collection.results()  
q var !results
```

4.4 Evaluating the Results From a COLLECTION Object

After an **ARRAY of DBREF** objects has been generated from a **COLLECTION** object, the entire array can be evaluated using the **.evaluate()** method on an array object. The argument for the **.evaluate()** method must be a **BLOCK** object defined with the expression that needs evaluating.

To get an **ARRAY of STRINGS** holding the full names of the collected elements, type the following:

```
!block = object BLOCK(!results[!evalIndex].flnn))  
!resultNames = !results.evaluate(!block)  
q var !resultNames
```

Notice how the **BLOCK** object uses the local variable **!evalIndex**. This variable effectively allows the **.evaluate()** method to loop through the ARRAY. To get the positions of the collected elements, type the following:

```
!resultPos = !results.evaluate(object BLOCK(!results[!evalIndex].pos))  
q var !resultPos
```

Instead of defining the block as a separate variable, this second example shows that the object can be defined within the argument to another object.

 Notice how the evaluated ARRAY contains the correct object types generated from the evaluation

Exercise 4 – Equipment Collections

- Create a new form that will act as an **Equipment Checker**. The purpose of the form is to provide users information about **EQUI** elements from the hierarchy
 - Design the form with a **COMBO** gadget to display to the user the available EQUI elements, a **LIST** gadget to display the **NOZZ** elements owned by the chosen EQUI, a **BUTTON** to update the form and a **PARAGRAPH** as a title.
 - An example layout for the form is adjacent

- Write a method that collects all the **EQUI** elements for the **ZONE** of the **CE**. Try and write the collection part of the method in a PML 1 style.
 - What happens if no pieces of equipment are found in the **ZONE**? Call this method when the form is shown.
 - Add an **OPEN CALLBACK** to the **COMBO** gadget to validate any user entered values.
 - Add an **'Update'** button that will re-run this method if a different **ZONE** is chosen. What happens if the user is at a **SITE** when the update button is clicked?
 - Write a method that runs after the piece of equipment has been chosen. This method should collect all the **NOZZ** elements for that **EQUI** element. Try writing the collection part of the method in a PML 2 style.
 - After the method has run, set the full names of the collected **NOZZ** elements to the **LIST**. Make use of the **RTEXT** and **DTEXT** members of the gadgets (to store names and **DBREFs**)
- Add a popup menu to the **LIST** gadget to allow the users to navigate to the chosen element (set the current element).
 - What method on a **LIST** gadget retrieves the chosen **RTEXT**?

- Write a method that looks at each of the collected **NOZZ** element and checks the following:

Is it connected?	This can be decided by checking the NOZZ element's CREF attribute. If the attribute is unset or the reference is invalid, then the user will have to check that nozzle.
Is it attached?	Check the positions of the NOZZ and PIPE elements and if they are different, then the NOZZ should be checked.
Is it aligned?	Check the directions of the NOZZ and PIPE elements. If they are different, then they should be checked.

Is it sized
correctly?

Compare the diameter of the PIPE to the size of NOZZ. If they are different, then they should be checked.

- Display the results of the check in the LIST gadget using columns.
- Provide a **Refresh** button to recheck the nozzles in case the users changes anything in response to the check results. This button should rerun the check method and update the form correctly.

Nozzles	Connected?	Attached?	Aligned?	Size?
NOZZLE 1 of T...	OK	OK	OK	OK
NOZZLE 2 of T...	OK	OK	OK	OK

- Add a **TEXTPANE** to the form to allow users to enter attribute values for the chosen piece of equipment.
- The **TEXTPANE** should initially be filled with three attributes (**Description**, **Function** and **Purpose**) and the values for the chosen equipment. As the user chooses other pieces of equipment, the displayed attribute values should update.

Nozzles	Connected?	Attached?	Aligned?	Size?
NOZZLE 1 of TM...	OK	OK	OK	OK
NOZZLE 2 of TM...	OK	OK	OK	OK

03SKID1-PUMPB Attributes

Description -
Function -
Purpose -

- Write a method to read the attributes and values from the **TEXTPANE**. This will involve splitting the information entered by the user. If the user has entered valid values, the attributes should be updated.
- This method should cope with new attributes being entered by the user. Consider what should happen if an invalid attribute or value is entered.
- As the **TEXTPANE** has no callback, provide a button to call this method.

An example of the completed form can be found in Appendix B

5 View Gadgets

VIEW gadgets are named gadgets which are used to display the information from available databases. The way the information is displayed depends on the VIEW gadget type.

General View Types:

ALPHA views for displaying text output and / or allowing command input.

PLOT views for displaying non-interactive 2D plot files.

Application-specific View Types

AREA views for displaying interactive 2D graphical views.

VOLUME views for displaying interactive 3D graphical views.

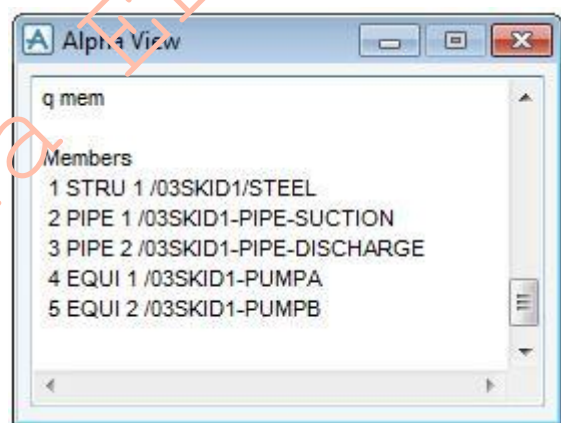
 Refer to the AVEVA E3D Customisation Reference Manual for more information

5.1 Alpha Views

An **ALPHA** view gadget is the same as used for the standard command window.

For the example, type **show !!traExampleAlphaView:**

```
setup form !!traExampleAlphaView
!this.formTitle = |Alpha View|
view .input AT X 0 Y 0 ALPHA
height 10 width 30
channel REQUESTS
channel COMMANDS
exit
exit
```



5.2 Plot View Example

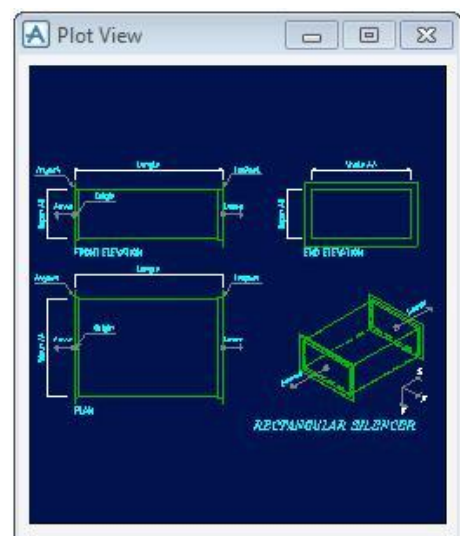
A **PLOT** view can be used to provide the user with extra information. This extra information would be contained in a .plt file and is applied to the view. The following example applies the file during the constructor method.

For the example, type **show !!traExamplePlotView:**

```
setup form !!traExamplePlotView
!this.formTitle = |Plot View|
view .plot plot width 30 height 11
CURS NOCURSOR
exit
exit

define method .traExamplePlotView()
!this.plot.borders = FALSE

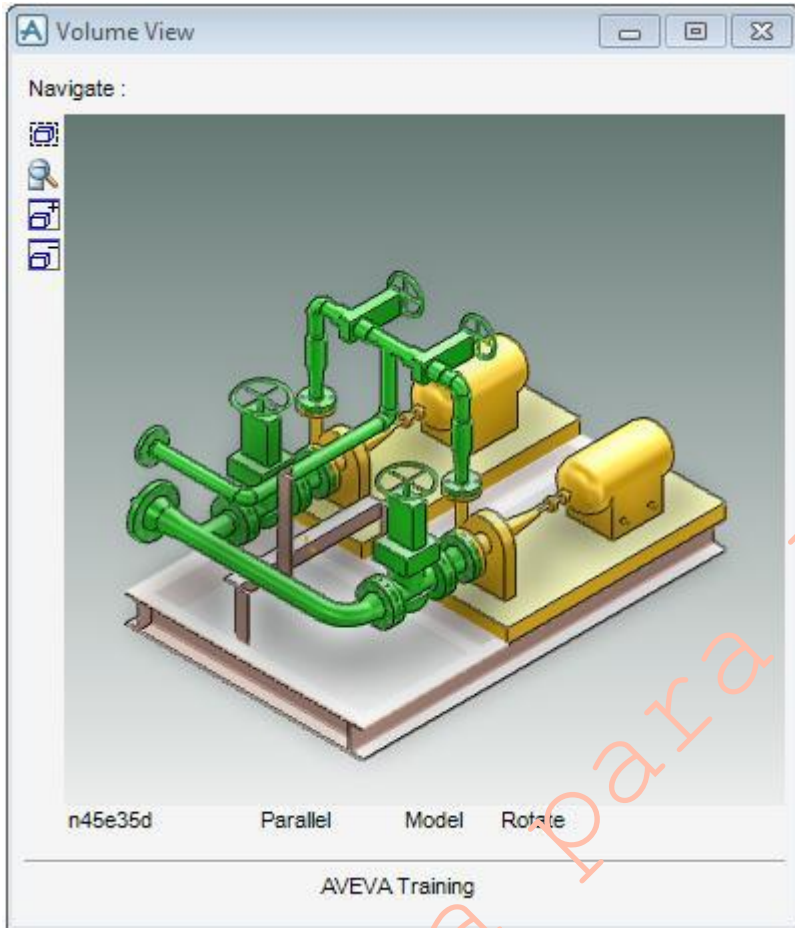
!this.plot.add(|C:\AVEVA\Plant\Training\ductABC.plt|)
endmethod
```



5.3 Volume View Example

A **VOLUME** view can be used to see the 3D data held within the AVEVA E3D database. The main AVEVA E3D window is an example of a volume view in use.

For the example, type **show !!traExampleVolumeView**:



The code that supports this example is in Appendix B

With AVEVA E3D, it is possible to assign different drawlists to different volume views. This has been demonstrated in the example by giving the volume view a different drawlist to the main view. There are two global variables that make this possible: **!!gphviews** and **!!gphdrawlists**

To investigate the **!!gphdrawlists**, type the following into the command window:

q var !!gphDrawlists

```
<GPHDRAWLISTS> GPHDRAWLISTS
DRAWLISTS <ARRAY> 1 Elements
```

q var !!gphDrawlists.drawlists

```
<ARRAY>
[1] <DRAWLIST> DRAWLIST
```

q var !!gphDrawlists.methods()

The following are some examples of the methods which are available on **!!gphDrawlists**. To find out information about all the drawlists in the global object, use:

!!gphDrawlists.listall()

To create a drawlist object in the global object, use:

!drawlist = !!gphDrawlists.createDrawlist()

q var !drawlist

To associate a drawlist from the global object with a view gadget, use:
`!!gphDrawlists.attachView(DRAWLIST, GADGET)`

Where **DRAWLIST** is a REAL represented the drawlist and **GADGET** is the volume view

Drawlists can only be attached to views that has been registered with the **!!gphViews** object. To investigate the **!!gphViews**, type the following into the command window:

q var !!gphViews

```
<GPHVIEWS> GPHVIEWS
  ACTIVEVIEW <GADGET> Unset
  SELECTEDVIEWS <ARRAY> 0 Elements
  VIEWS <ARRAY> 1 Elements
```

q var !!gphViews.methods()





To add a view to the global object, use:
`!!gphViews.add(GADGET)`

Where **GADGET** is the volume view

To set the limits of a view based on a DBREF object, use:
`!!gphViews.limits(GADGET, DBREF)`

Where **GADGET** is the volume view and **DBREF** is the element to set the limits to.

The example form also mimics the functionality of the main 3D window by providing the following four buttons:

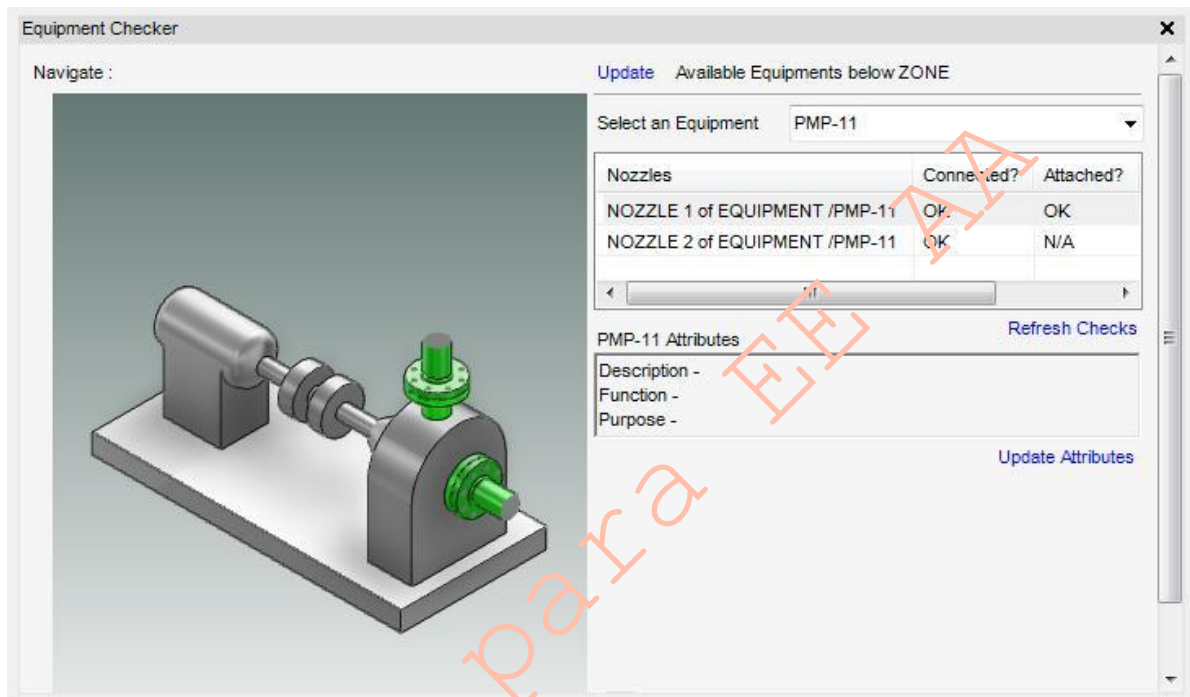
-  Set view limits to CE
-  Walk to Drawlist (set limits of the view based on drawlist)
-  Add the current element to the drawlist
-  Remove the current element from the drawlist

 *These buttons are not always necessary when defining a volume view element, but have been included in the example to demonstrate some further methods on the object.*

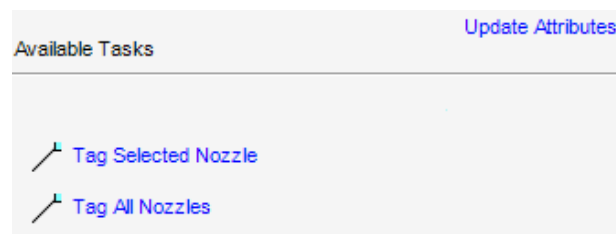
Review form definition and its method to understand how the process works.

Exercise 5 – Adding a Volume View to a form

- Add a **VOLUME VIEW** element to the form created in Exercise 4. The Volume View should have its own drawlist and should correctly display the chosen piece of equipment only
 - Look at **!!exampleVolumeView** and see which parts of the PML can be re-used. How will the method know which element has been chosen and how will the drawlist/limits be updated?
 - An example of the updated form is below (new view and given the ability to dock)

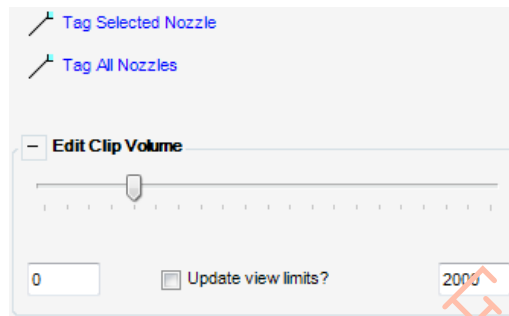


- Add an **'Available Tasks'** section to the form. Any additional functionality added to the form shall be added here.
- Add a method to the form to tag **NOZZ** elements on the chosen piece of equipment. This will allow to the user to identify which nozzle is which.
 - Provide two extra tasks: **Tag Selected Nozzle** (chosen in the list) and **Tag All Nozzles**



- When the user tags a nozzle, use the **MARK** or **AID TEXT** syntax to put the name of the nozzle into the volume view. The method should keep track of which nozzles are tagged so that they are only tagged once (use a form member?). This will also manage the state of the toggle so that it is always correct.

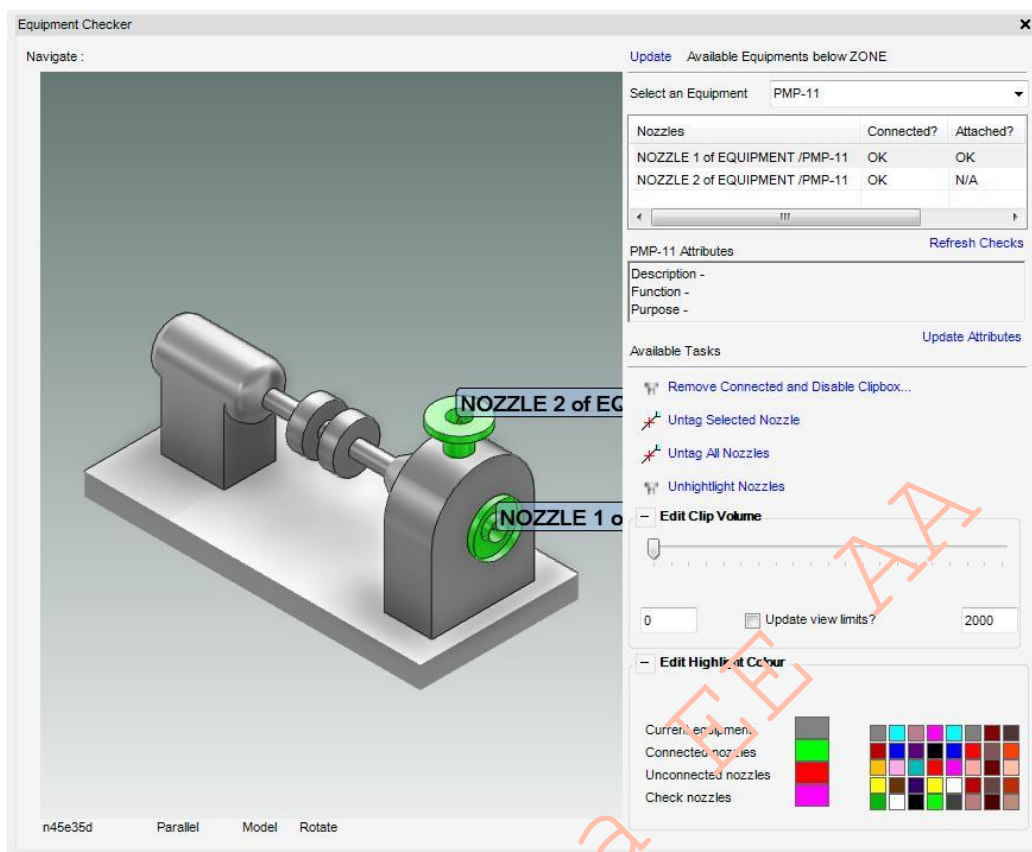
4. • Update the 'Add Connected' method to include a clipbox. This will limit how much the user sees. A clipbox is added to a Volume View by using a **GPHCLIPBOX** object. The **VIEW** member of a **GPHCLIPBOX** object holds the name of the Volume View you wish to clip.
 - The **GPHCLIPBOX** object should be stored as a member of the form. This is so it can be resized as different elements are chosen. To find out more about the **GPHCLIPBOX** object, open **gphclipbox.pmlobj**
 - When the user turns the clipbox on, they should then see a hidden fold-up panel which holds a slider. This slider will allow the user to dynamically change the size of the clipbox. Make use of the **.refresh()** method to update the volume view during the slide



- Add a method to update the range of the slider based on the input into two text boxes. This will be in case the equipment is too large/small for the default value.
 - Add a method that will update the limits of the view when the clipbox is resized. This is to stop the clipbox becoming larger than the limits of the view.
5. • Add a method to the form that will add colour to the elements of the drawlist to indicate the results of the nozzle check. Colour can be added through the **.highlight()** method on a **DRAWLIST** object.
 - Three different colours should be added. A colour for unconnected nozzles, a colour for nozzles that need checking and a colour for connected nozzles which pass the checks. You may also wish to highlight the chosen piece of equipment.
 - When the user turns on the highlighting, show a hidden frame which allows users to change the colours. Four buttons shall display the current colours. On pressing one of those buttons, a hidden panel frame shall be shown which will allow the users to update the colour. Think about how a DO loop could be used to create this grid of buttons.



- The methods will need to manage which elements are coloured, which colours should be used and the visibility of the gadgets on the form
6. • To complete the form, test the functionality on a range of equipment elements. Address any errors that occur during this process.



 An example of the completed form can be found in Appendix B

6 Event Driven Graphics (EDG)

The EDG has been developed to allow a common interface for appware developers to use when setting up the graphic canvas for graphical selection. This method is relatively simple and easily extendible. This will allow the developer to concentrate on the development of their own application without the need to know the underlying mechanism (core implementation) of the EDG interaction handlers and system.

The system handles all the underlying maintenance of the current events stacked e.g. associated forms, initialisation sequences, close sequences, etc.

The current implementation of the system has mainly been developed for interaction with the 3D graphic views in the Design module. However, the interface can be used with any of the modules that use the same executable and the standard 3D graphic views. It is intended that EDG will supersede the old ID@ syntax. When setting up an EDG event, the following should be considered:

- What items will the user be picking?
- How many picks are required and in what order?
- What happens when the user makes a pick?
- What happens when the user has finished picking?

These aspects are controlled by methods on the objects associated with EDG. The main object is an **EDGPACKET** object. Once setup, this object is added to the **!!EDGCtrl** object that will activate the event.

❗ For more information about EDG, refer to the EDG interface manual

❗ In E3D 2.1 using EDG will disable the CIE tools (PowerWheel, PowerCompass, Canvas Commands)

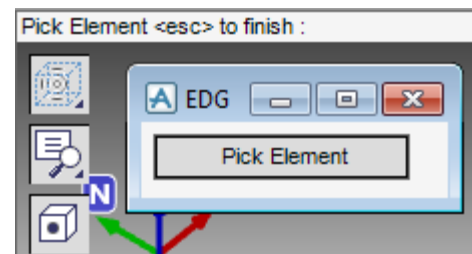
6.1 A Simple EDG Event

To demonstrate a simple EDG setup, show the example by typing **show !!traExampleSimpleEDG**:

```
setup form !!traExampleSimpleEDG
  !this.formTitle = |EDG|
  button .but1 | Pick Element | call
  |!this.pick()|
exit

define method .pick()
  -- Define event packet object
  !packet = object EDGPACKET()
  -- Define object as a predefined standard element pick
  !packet.elementPick(|Pick Element <esc> to finish|)
  -- Query the item member of the returned information from the pick
  !packet.action = |!!traExampleSimpleEDG.process(!this.return[1])|
  -- Set what happens when the user presses esc
  !packet.close = |$p Finished|
  -- Add the event packet to the global EDG control object
  !!EDGCtrl.add(!packet)
endmethod

define method .process(!item is ANY)
  q var !item
endmethod
```



The example sets up a predefined element pick event by running the `.elementPick()` method on a **EDGPACKET** object. The subsequent methods specify what happens when a pick is made and what happens when the event is finished.

- ① Notice how the action method of the **EDGPACKET** object references the form method explicitly. It displays the returned information (described in EDG interface manual)

6.2 Using EDG

The following example shows how EDG can be applied to a form to provide picking functionality for users. To show the example, type **show !!traExampleEDG**:

```
setup form !!traExampleEDG
!this.formTitle = |Pick Equipment|
!this.initcall = !!this.init()|
!this.quitcall = !!this.clear()|
button .pick |Start Pick| call !!this.init()|
textpane .txt1 |Picked Equipments| at x 0 ymax width 25 height 10
member .storage is ARRAY
exit
define method .init()
!this.clear()
!packet = object EDGPACKET()
!packet.elementPick(|Pick Equipments <esc> to add to list|)
!packet.description = |Identify Equipment|
!packet.action =
!!traExampleEDG.identify(!this.return[1].item)|
!packet.escape = !!traExampleEDG.setInfo()|
!!EDGCtrl.add(!packet)
endmethod

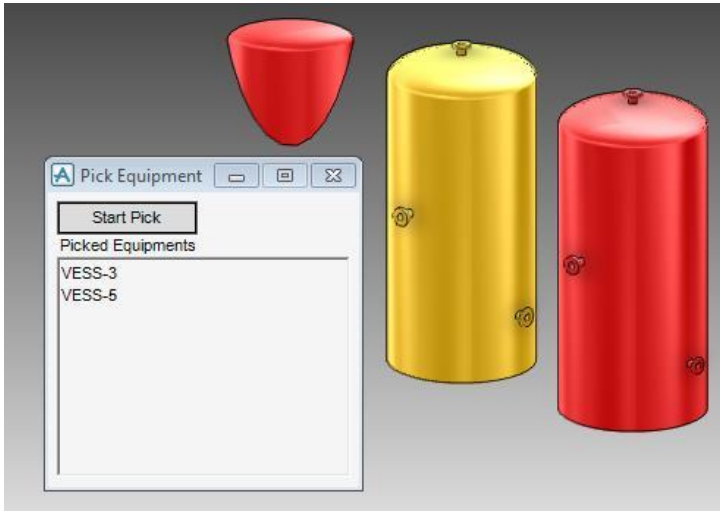
define method .identify(!pick is DBREF)
do
if !pick.type.eq(|EQUI|) or !pick.type.eq(|WORL|)
then
break
else
!pick = !pick.owner
endif
enddo
if !pick.type.eq(|EQUI|) then
if !this.storage.findfirst(!pick).unset() then
!!gphDrawlists.drawlists[1].highlight(!pick, 314)
!this.storage.append(!pick)
else
!this.storage.remove(!this.storage.findfirst(!pick))
!!gphDrawlists.drawlists[1].unhighlight(!pick)
endif
endif
endmethod

define method .setInfo()
!block = object BLOCK(|!this.storage[!evalIndex].flnn|)
!names = !this.storage.evaluate(!block)
!this.txt1.val = !names
endmethod

define method .clear()
do !element values !this.storage
!!gphDrawlists.drawlists[1].unhighlight(!element)
enddo
!!edgCtrl.remove(|Identify Equipment|)
```




```
!this.txt1.clear()
!this.storage.clear()
endmethod
```

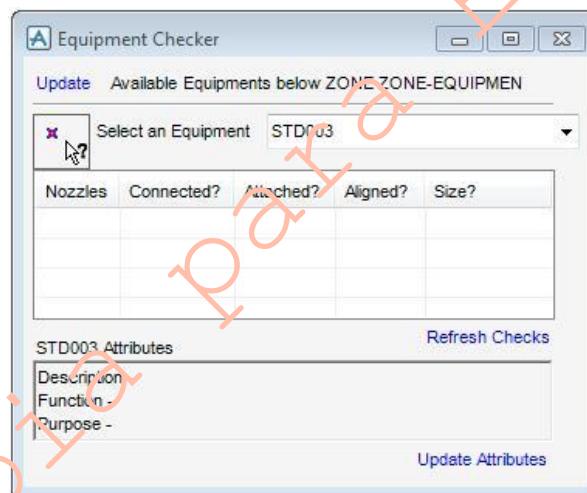


This example shows how a method can take the returned information and use it. In this case, the picked element type is checked whether it is a piece of equipment. If it is not, the method loops up the hierarchy until one is found (or the world is reached). The EQUI is then highlighted if it's new or unhighlighted if already chosen. There is a different method for the close action. For this reason, the picked elements are collected a form member. The escape method (press ESC keyboard button) applies the form member to the textpane gadget.

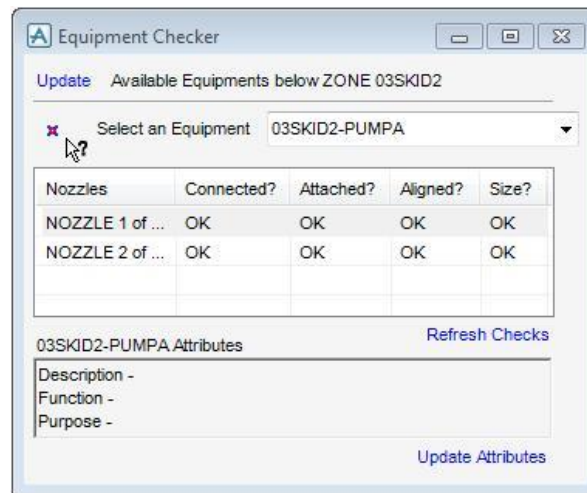
Notice how the form uses the .clear() method to tidy up

Exercise 6 – Adding EDG to Forms

1.
 - Add a pixmap button to the form created in Exercise 4 that will allow users to pick which equipment they want from the main 3D window. Once picked, that piece of equipment (if available) should be selected in the **OPTION** gadget
 - The button should initialise an EDG event. This EDG event should run a method on picking an element that does the following:
 - Turn the EDG event off after something has been chosen
 - Identify the type of element chosen
 - If an EQUI has been chosen, OK
 - Otherwise, loop up the hierarchy to find an EQUI.
 - If an EQUI cannot be found, report to the user and re-initialise the EDG
 - Find the chosen EQUI in the OPTION gadget and set it to the correct equipment
 - Run the nozzle collection method



2. • Extend the method to allow **NOZZ** elements to be chosen (as well as EQUIs). When a nozzle is chosen, highlight it in the **LIST**. Consider what happens if a nozzle is chosen which is not in the list, but is owned by a piece of equipment in the **OPTION** gadget.



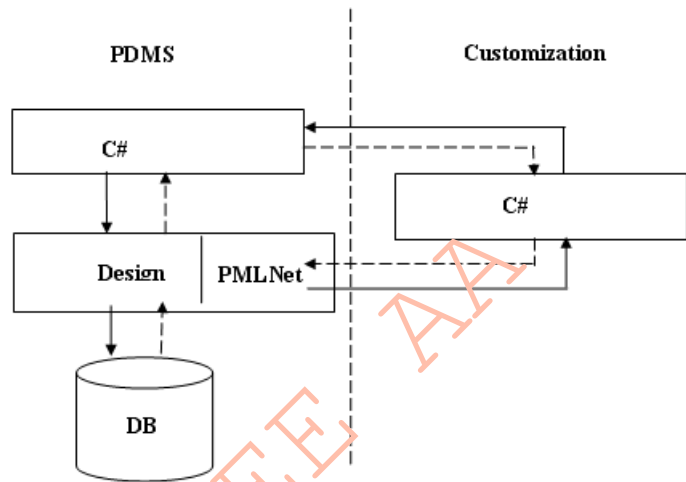
Copia para EE AA

7 PML.NET

Using AVEVA E3D, it is possible to customise the product using .NET assemblies through PML .NET controls and objects. PML .NET allows you to instantiate and invokes 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. The PML proxy objects behave just like any other PML object.

PML callable assemblies are loaded by AVEVA E3D 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 and once 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.



It is possible to add container objects to a conventional PML form creating a hybrid of PML & .NET. The container is treated as a normal form gadget (i.e. syntax graph) but can be filled with a .NET object

7.1 Import an Assembly into AVEVA E3D

Before a .NET control can be used in a form, it first has to be imported into AVEVA E3D. The required .dll file has to be loaded and the namespace specified. For example:

```
import 'GridControl'
using namespace 'Aveva.Core.Presentation'
```

i Trying to import a .dll file which has already loaded or cannot be found will result in an error. Error handling should be considered to prevent the error messages being displayed to users.

7.2 Syntax

To define your first .NET object, type out the following into the Command Window:

```
import 'GridControl'
using namespace 'Aveva.Core.Presentation'
!netObj = object NETGRIDCONTROL()
q var !netObj
q var !netObj.methods()
```

i Notice the .methods() method. This is a new global method available on all objects. It returns all of the methods

Exercise 6 - Create a File browser Object

1. • Create a File Browser object, type the following onto the command line.

```
import 'PMLFileBrowser'
using namespace 'Aveva.Core.Presentation'
!browser = object PMLFileBrowser('OPEN')
```

 The arguments are OPEN and SAVE. If no argument is given, then it will default to OPEN.

2. • Query the object and the methods of the **PMLFILEBROWSER** instance.

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

3. • To display the browser form, type the following:

```
!browser.show('C:\','abc.doc','Example',true, 'Word Docs|*.doc',1)
q var !browser.file()
```

- The arguments to the .show method set up the browser, they are as follows:
 1. Start directory (String)
 2. Initial file name (String)
 3. Title for the browser (String)
 4. Should the file exist (Boolean)
 5. File filter (String) - type of files to open. A description, followed by "|" and then the file types – for example: Word Documents|.DOC|Text files (*.txt)|*.txt|All files (*.*)|*.*
 6. Index (integer) – for multiple strings, the index of the filter currently selected

7.3 Creating a PML form containing the .NET Control

Once the .dll file has been loaded and the namespace specified the object is available for use. For example a user may specify a **CONTAINER** within a form definition to hold the .NET control and a form member to define it:

```
container .netFrame PMLNETCONTROL 'NET' dock fill
member .netControl is NETGRIDCONTROL
```

After this, the **constructor method** would define the .NET object and apply it to the container gadget

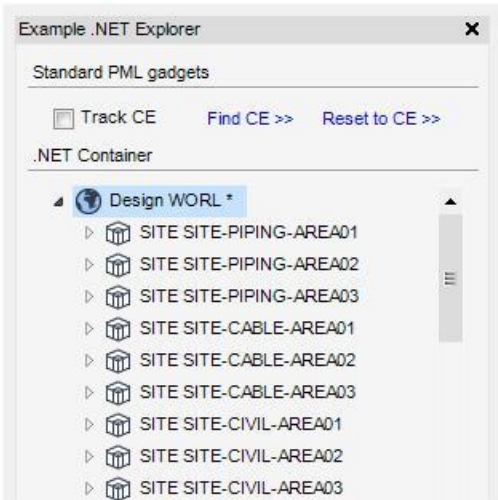
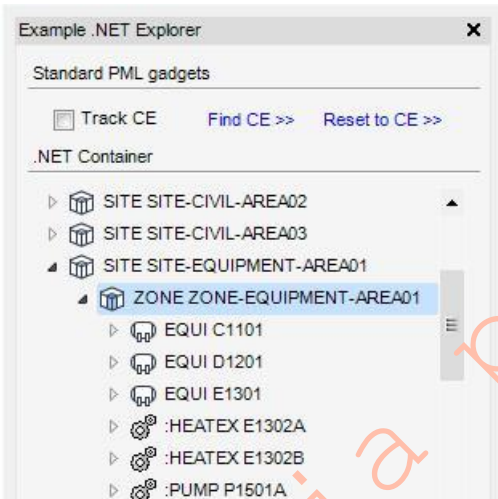
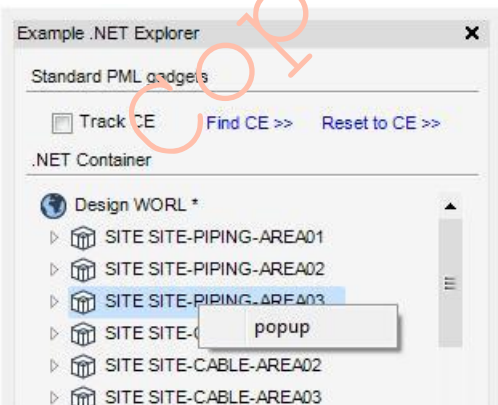
```
using namespace 'Aveva.Core.Presentation'
!this.netControl = object NETGRIDCONTROL()
!this.netFrame.control = !this.netControl.handle()
```

The user is now able to specify event methods to control the .NET gadget:

```
!this.netControl.addeventhandler('OnPopup', !this, 'rightClickExplorer')
```

This allows the forms **.rightClickExplorer()** method to be run when the .NET object is right clicked.

Exercise 7 - Explorer Control on PML Form

1. 
 - Type the following on the AVEVA E3D Command line:
show !!traExampleExplorer
 - You will see a form which uses a container gadget to display a .NET explorer control
 - The example code for this form can be found in Appendix B.
2. 
 - Experiment with the "Track CE" toggle and the "Find CE" functionality
 - The PML gadgets are invoking methods on the C# control.*
 - If you "Reset to CE", what happens?
3. 
 - A popup menu has been programmed on the explorer control.
 - The current method prints the name of the element to the command window.
 - Extend this method to increase the functionality of the form. For example set the current element, show attributes, provide member information

7.4 Grid Control

Another example of an available .NET object is the **Grid Control** gadget. Despite appearing similar to a **LIST** gadget, the Grid Control gadget has a large number of methods available to it allowing it to behave more like a spreadsheet. Some of the advantages are as follows:

- Data in the grid can be selected, sorted and filtered.

- Data in the grid can be exported to/imported from a Microsoft Excel file
- Grid contents can be previewed and printed directly
- The colour of rows, columns or cells can be set (including icons)
- The values of entire rows/columns can be extracted/set
- The contents of the grid can be filled based on DB elements and their attributes

7.4.1 Applying Data to the Grid

Data is applied to a Grid Control gadget through the use of a **NETDATASOURCE** object. While defining this object, the required data is collected and formatted for the grid. The data is applied to the grid by using its **.bindToDataSource()** method. For example:

```
var !coll collect all EQUI
!attribs = |NAME TYPE OWNER AREA|
!data = object NetDataSource('Example Grid, !attribs.split(), !coll)
!this.exampleGrid.BindToDataSource(!data)
```



In this example, the NetDataSource object is collecting information from the database. There are other methods available. Refer to the .NET Customisation User Guide, available within the 12.0 manual

7.4.2 Events and Callbacks

The following events are available on the grid:

- 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
- 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 key of the cell
 - Array[3] is a Boolean. If you set it to "false" then the new value will NOT be allowed
- AfterSelectChange(Array)
 - The array contains the ID of each of the selected rows
 - AfterCellUpdate
 - No arguments are passed.



For more information, refer to Appendix A or the .NET Customisation User Guide, available within the AVEVA E3D manuals

Exercise 8 - A Grid Control on a PML Form

1. • Type the following on the AVEVA E3D Command window:

show !!traExampleGridControl

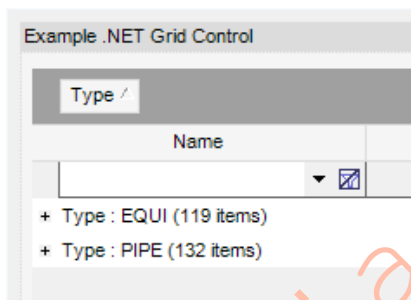
Example .NET Grid Control

Drag a column header here to group by that column.

Name	Type	Owner	Description
/C1101	EQUI	/ZONE-EQUIPMENT-AREA01	COLUMN
/D1201	EQUI	/ZONE-EQUIPMENT-AREA01	Reflux Drum
/E1301	EQUI	/ZONE-EQUIPMENT-AREA01	Reboiler
/P1501A	EQUI	/ZONE-EQUIPMENT-AREA01	Reflux Pump - Duty
/P1501B	EQUI	/ZONE-EQUIPMENT-AREA01	Reflux Pump - Standby
/P1502A	EQUI	/ZONE-EQUIPMENT-AREA01	Overhead Product Pump - Duty
/P1502B	EQUI	/ZONE-EQUIPMENT-AREA01	Overhead Product Pump - Standby
/E1302A	EQUI	/ZONE-EQUIPMENT-AREA01	Reflux Condenser A
/E1302B	EQUI	/ZONE-EQUIPMENT-AREA01	Reflux Condenser B

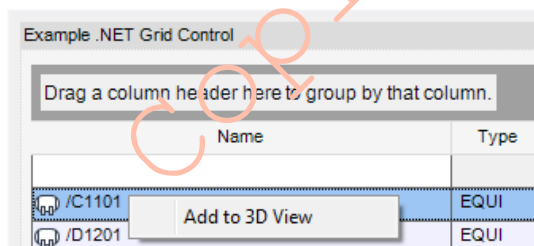
- ① The grid has been populated with a set of attributes (the headings), and a set of model items (Equipment and Pipes in this example). The grid control populates itself with the attribute data.

- 2.



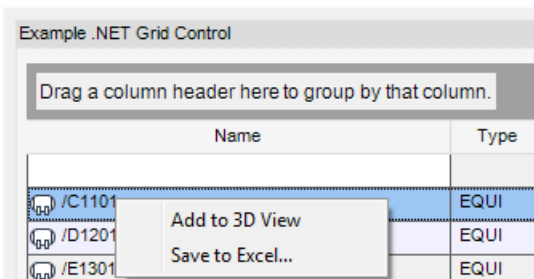
- Grid Behaviour:
- Drag and drop the "TYPE" heading into the grouping area (see picture)
- Drag and Drop selected items into the 3D view (or "My Data")
- Investigate the filters on each column

- 3.



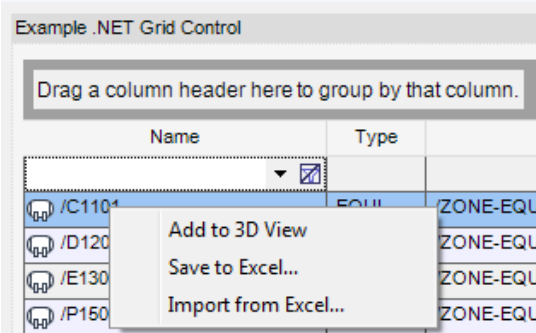
- Pop-up Menu:
- Make a selection and open the pop-up menu. The menu doesn't do anything yet!
- You need to edit the PML form to enable the Pop-up menu to run a callback function.

- 4.



- Save Data to an Excel file:
- Add to the popup menu the ability to save the grid data to an Excel spreadsheet.
- Make use of the File Browser object to enable the user to specify the Excel file on the file system.

5.



- Import Data from an Excel file:
- Add to the popup menu the ability to import data from an excel spreadsheet into the grid.
- ① *Note that the imported data will replace the data which is already there (not append to it) and the data is completely defined by the spreadsheet.*
- The first row in the Excel file will represent the column headings. The other rows will be data.

6. • Event: afterSelectChange

- Add an “afterSelectChange” event
- Add a method to print out the name of the new selection whenever you change the selection in the grid.

7. • Grid Control API:

- Try out some of the other events and methods available on the Grid Control.



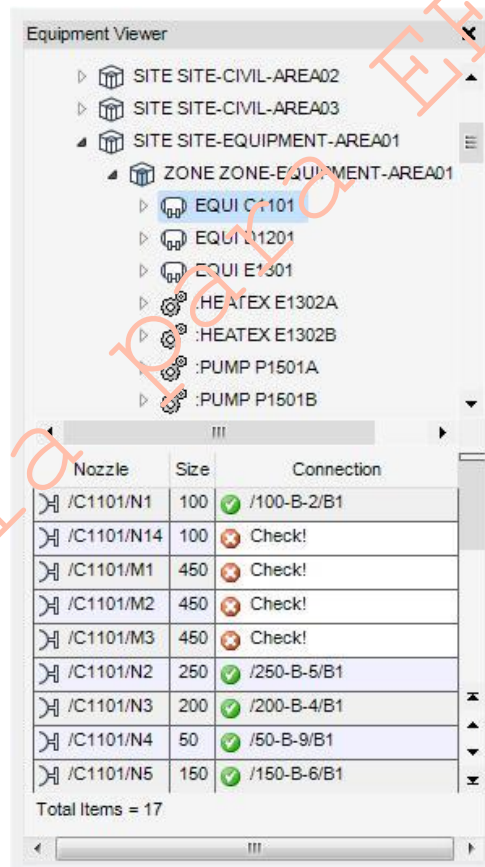
For details of the methods available on a grid control gadget, refer to Appendix A or the .NET Customisation User Guide, available within the AVEVA E3D manual.

Copia para E3D

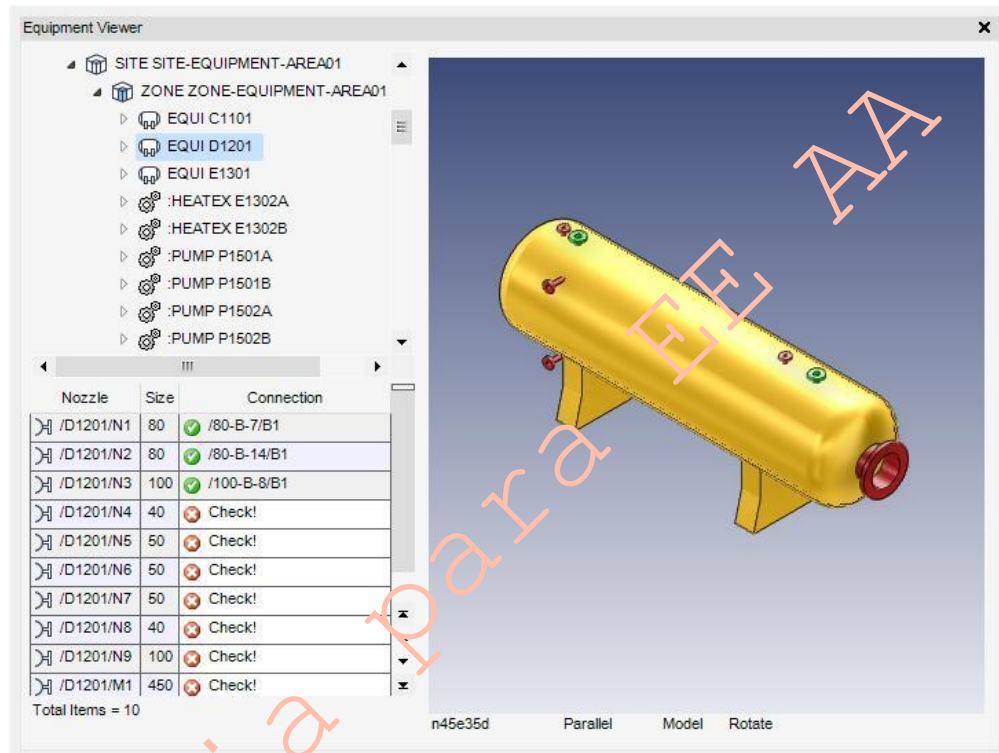
Exercise 9 – Developing a PML .NET form

- Create a new form called **!!c2ex9** and save it to the PMLLIB area.
 - The purpose of this form will be to use the PML .NET controls to create an Equipment Viewer form. This form will mimic the functionality of exercise 5, but will benefit from the increased user experience the .NET controls bring.
 - Provide a **.NET Explorer** which displays the **ZONE /EQUIP** and below from the example Stabilizer plant.
 - Provide a **.NET Grid Control** gadget. This gadget will display the names of the nozzles owned by the chosen, as well as their size and connection information. The contents of the **Grid** should refresh every time a piece of equipment is chosen.
 - When displaying the connections of the **NOZZ** elements, make it more obvious to the user which ones are connected. This could be done by changing colours, display text or displaying icons.

 Refer to the *.NET customisation Guide for suitable methods.*



2.
 - Extend the form by adding a volume view gadget to the form
 - Give the volume view its own drawlist.
 - Consider the management of this drawlist (when should it be created, modified, removed)
 - What should be displayed in the drawlist?
 - There are many methods available on a drawlist object. Provide a way for the user to see the results of the connection check within the view
 - Test the form on a range of elements.



8 Miscellaneous

The following sections describe some other useful PML actions.

8.1 Error Tracing

It is possible to list all the commands that AVEVA E3D runs when a PML operation is carried out. This list can either be printed to the command window, or written to an external file.

Type \$r109 /C:\AVEVA\Plant\Training\log.log onto the command line

Where C:\AVEVA\Plant\Training\log.log is the output file name (it shall be created if it does not exist)

Now every PML action will be written to the external file. The file will list every line of code and action carried out. Lines read will be indicated by a line number in square brackets. Lines not read will be indicated by a line number between round brackets. Entry and exit points between methods, functions and objects are indicated as well as any errors.

Type \$r110 onto the command line to print the same lines to the command window.

Printing to the screen should only be done when a small number of lines are expected. The command window may run out of lines to display the information

Once you are finished with error tracing, type \$r to the command line. If the file is not closed, information will continue to be added to it

This will need to be typed before the output file can be opened

To find out more information about the \$r command, type \$HR into the command window.

8.2 PML Encryption

It is now possible to encrypt any files you create before sharing them using PML Publisher. Once encrypted, the files can still be used in any compatible AVEVA program, but they are not easily read through a normal text editor. Encrypted files may be used without additional licenses, but the encryption utility described below is separately distributed and licensed.

Once encrypted, PML cannot be decrypted. A reference copy of all PML should be kept safe

The default encryption is implemented using the Microsoft Base Cryptographic Provider, which is included in, among other operating systems, Windows 2000 and Windows XP. There is a limit to the strength of this encryption and is not secure against all possible attacks. AVEVA may release improved algorithms with future releases of the product. If this happens, encrypted files would require re-encrypting.

Move the pmlencrypt.exe to the root folder where the files for encryption exist. Write a local .bat file containing the following line, where -pmlib is an instruction to encrypt PML files, ForEncryption is a folder containing the source file and Encrypted is a folder where the encrypted files shall be put

pmlencrypt -pmlib ForEncryption Encrypted

If the example form !!nameCE was encrypted then the following file would be created:

```
--<004>-- Published PML 1.0 >--
return error 99 'Unable to decrypt file in this software version'
$** abdfel9b3008494b6399edda08b66004
$** MR+zhtg-egE2Ig9IiHSVmdPo08ChKexa7wbfcyODTbfjTFWU02pK3v4sXq5i
$** TKW3dEFRJCd60uSzaLXdc5fvLeOKqXO71uF1Z1vEsIOOHvq8viAwiys4rGXg
$** XLgFFVG7mpsmnFtrQDN3o51aiAgicFS6u08C7r8IaxUTUQA0dXeBmlp4TLXc
$** 9KR5LtAIugLrC9a7NxbF+0Hn-c5tOhUAEBG
```



Reading an encrypted file is slower than reading a decrypted one. Making use of the Buffer argument can help. Refer to the PML Publisher User Guide for more information

Copia para EE AA

9 Menu and Toolbar Additions (Bar Menu Modules Only)

9.1 Miscellaneous Notes

To find out the menus currently contained within a model, use `q var !!appcatmain` for Paragon.

More examples can be found in directory `pmlib/design/objects`. All the add-in application objects begin with the letters `app****.pmlobj` e.g. `appaccess.pmlobj`.

9.2 Adding a Menu/Toolbar

For a file created in `%PMLUI%\DES\ADDINS` directory, the name doesn't matter, except for identification. The file will contain similar information to the old applications file in `%PMLUI%`. This file is used for menu additions, new toolbars and standard application switching

9.3 Form and Toolbar Control for Bar Menu Modules

Forms are controlled using the `APPFORMCNTRL` object. Forms can be registered to:

- Appear only in certain applications
- Be reshown when AVEVA E3D restarts (serialisation)
- `!!appFormCntrl.registerForm(!form is FORM)`

Toolbars are controlled using the `APPTBARCNTRL` object: to avoid having too many toolbars on screen, each is enabled only when the user is in the right application. Users can add more forms/toolbars to these control objects, and those items will be shown to the user.

9.3.1 Bar Menu and Toolbar

```
define method .barMenu()
  !bmenu = object APPBARMENU()
  !bmenu.add('Position', 'POSITION')
  !bmenu.add('Orientate', 'ORI')
  !bmenu.add('Connect', 'CONN')
  !!appMenuCntrl.addBarMenu(!bmenu, 'EQUI')
endmethod
```

9.3.2 Define Toolbar

```
define method .toolbars()
  frame .equiToolbar toolbar 'Equipment
Toolbar'
  -- define buttons for toolbar
  exit

!!appTbarCntrl.addToolbar('equiToolbar',
'EQUI')
endmethod
```

9.3.3 Adding to Menus

```
define method .createMenu()
  !menu = object APPMENU('sysCrt')
  !menu.add('SEPARATOR')
  !menu.add('FORM', 'Equipment...',
'CDEQUI', 'equiEquipment')
```



```
!menu.add('FORM', 'Sub-Equipment...',
  'CDSUBEQ', 'equiSubEquipment')
!menu.add('FORM', 'Primitives... ',
  'CDPRIM', 'equiPrimitives')
!menu.add('FORM', 'Standard... ',
  'equCreateStd', 'equiStandard')
!!appMenuCntrl.addMenu(!menu, 'EQUI')
endmethod
```

9.3.4 Adding New Menus to Form

```
define method .menus()
  !menu = object APPMENU('POSITION')
  !menu.add('CALLBACK', |Explicitly
(AT)...|, |!!pos3DExplicit()|,
'Explicitly')
  !menu.add('CALLBACK', |Relatively
(BY)...|, |CALLIBD X3DPOSR|,
'Relatively')
  !menu.add('MENU', |Move|, 'MOVE',
'Move')
  !menu.add('MENU', |Drag|, 'DRAG',
'Drag')
  !menu.add('MENU', |Plane Move|,
'PLMOVE', 'PlaneMove')
  !menu.add('SEPARATOR')
  !menu.add('MENU', |Equipment Point|,
'PPEAT', 'EquipmentPoint')
  !!appMenuCntrl.addMenu(!menu, 'EQUI')
...
endmethod
```

9.4 Example Object Including Toolbars, Bar Menus and Menus for Bar Menu Modules

This is only an example showing adding toolbars, bar menus and menus. The effect of this object will be to add a new toolbar and main menu. It will also add some extra menu options to the EQUI application, including some under the Utilities menu. It will demonstrate how user-defined forms/functions/callbacks can be added to the AVEVA E3D menu system.

i The methods `.modifyForm()` and `.modifyMenus()` run automatically, so can be used to call the objects methods.

Type out the following and save it as `appcompa.pmlobj`

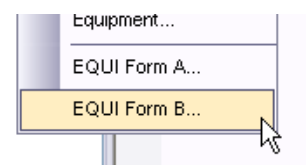
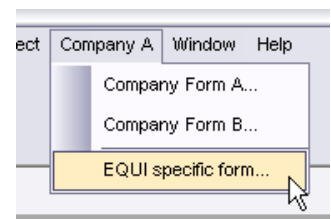
```
define object APPCOMPA
endobject

-----
-- Method:      .modifyForm()
-- Description: Standard method
--              Makes modifications to the main form
-----

define method .modifyForm()
  !this.toolbars()
endmethod

-----
-- Method:      .modifyMenus()
-- Description: Standard Method
--              Runs all other menu creation methods
-----

define method .modifyMenus()
  !this.barMenu()
  !this.menus()
  !this.specificMenus()
```




```

!this.UtilsMenu()
endmethod

-----
-- Method:      .UtilsMenu()
-- Description: Adds to the Utilities menu in the EQUI application
-----
define method .UtilsMenu()
    !menu = object APPMENU('SYSUTIL')
    !menu.add('SEPARATOR')
    !menu.add('CALLBACK', |EQUI Form A...|, |$p Form A|)
    !menu.add('CALLBACK', |EQUI Form B...|, |$p Form B|)
    !!appMenuCntrl.addMenu(!menu, 'EQUI')
endmethod

-----
-- Method:      .toolbars()
-- Description: Creates toolbar for all applications
-----
define method .toolbars()
    frame .MyFormToolbar toolbar 'CompanyA Toolbar'
        !iconSize = !!comSysOpt.iconSize()
        !Pixmap1 = !!pml.getPathName('createpipe-' & !iconSize & '.png')
        !Pixmap2 = !!pml.getPathName('exclamation-' & !iconSize & '.png')
        button .but1 'Show Form A' tooltip 'Show Company Form A' $
            pixmap /$!<Pixmap1> width $!iconSize height $!iconSize callback |$p Form A|
        button .but2 'Show Form B' tooltip 'Show Company Form B' $
            pixmap /$!<Pixmap2> width $!iconSize height $!iconSize callback |$p Form B|
    exit
    !!appTbarCntrl.addToolbar('CompAToolbar', 'ALL')
endmethod

-----
-- Method:      .barMenu()
-- Description: Adds options to the bar menu for all applications
-----
define method .barMenu()
    !bmenu = object APPBARMENU()
    !bmenu.add(|Company A|, 'CompAMenu')
    !!appMenuCntrl.addBarMenu(!bmenu, 'ALL')
endmethod

-----
-- Method:      .menus()
-- Description: Creates menus to the bar for all applications
-----
define method .menus()
    !menu = object APPMENU('CompAMenu')
    !menu.add('CALLBACK', |Company Form A...|, |$p Form A|)
    !menu.add('CALLBACK', |Company Form B...|, |$p Form B|)
    !!appMenuCntrl.addMenu(!menu, 'ALL')
endmethod

-----
-- Method:      .menus()
-- Description: Creates menus to the bar for specific applications
-----
define method .specificMenus()
    !menu = object APPMENU('CompAMenu')
    !menu.add('SEPARATOR')
    !menu.add('CALLBACK', |EQUI specific form...|, |$p EQUI specific|)
    !!appMenuCntrl.addMenu(!menu, 'EQUI')
endmethod

```

To call the above object, copy out the following into a new file called **COMP**A (no file extension) and save it into %PMLUI%\des\addins\. Update %PMLUI% in AVEVA E3D.bat if not done already.

```

name:          COMP
title:         Company A app
showOnMenu:    FALSE
object:        APPCOMP

```

Where: Name is the unique id for user and addin code; Title is the text description of the add-in; ShowOnMenu determines whether the add-in has an entry on the Applications menu, i.e. whether the add-in defines an application (false by default). If it were an application addin like MDS this would be TRUE; Object is the object file that is used to define the menu/toolbar additions.

These are some optional lines that can be included. **Callback** (typically CALLE XEQUI Macro to call when the menu is selected) **Startup** (typically \$m/%PMLUI%/des/equi/init Macro to run on startup of the equi app (for first time)) **ModuleStartup** (the callback run when the AVEVA E3D module first starts) **StartupModify** (Name of application to modify and the callback run when an application is first started, separated by a colon. e.g. EQUI:!!equiAppStartupCall()) **SwitchModify** (Name of application to modify and the callback to run when the application is switched to, separated by a colon. e.g. PIPE:!!pipeAppSwitchCall())

To add a startup macro to more than one application, add more entries into the add-in file typically:-

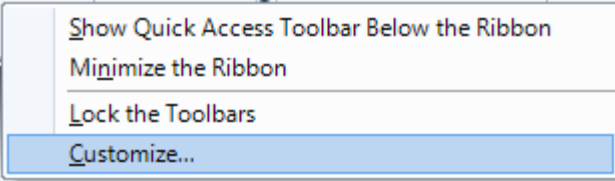
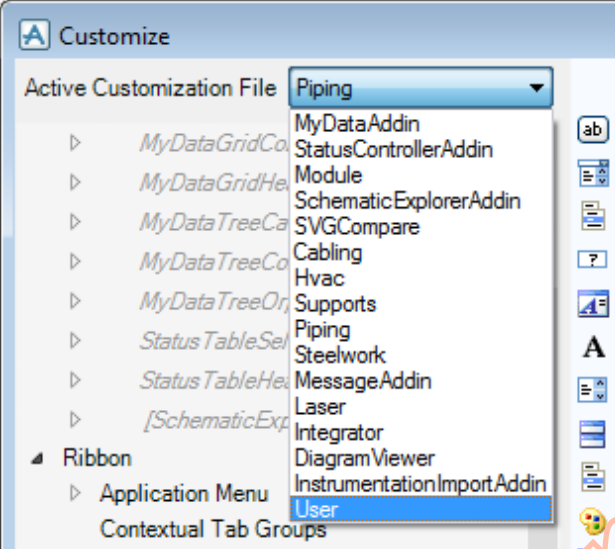
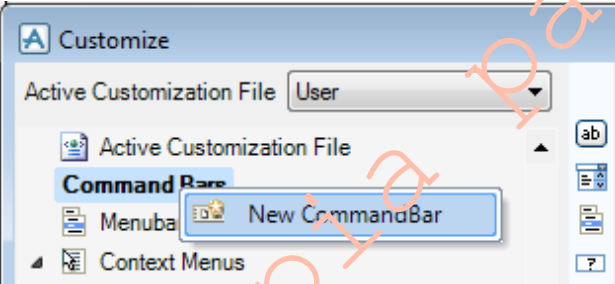
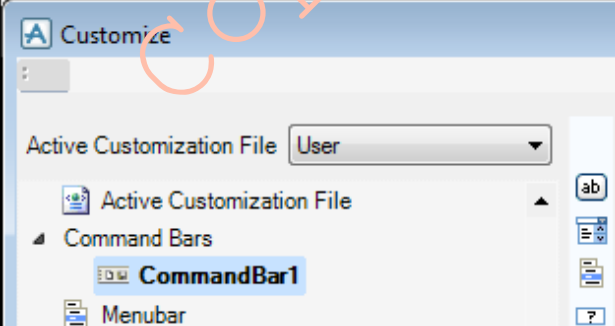
```
switchModify:EQUI:!!equiAppStartupCall()
switchModify:PIPE:!!equiAppStartupCall()
```

The following keys are only used for applications, i.e. add-ins that have a menu option on the Applications menu and can be switched to.

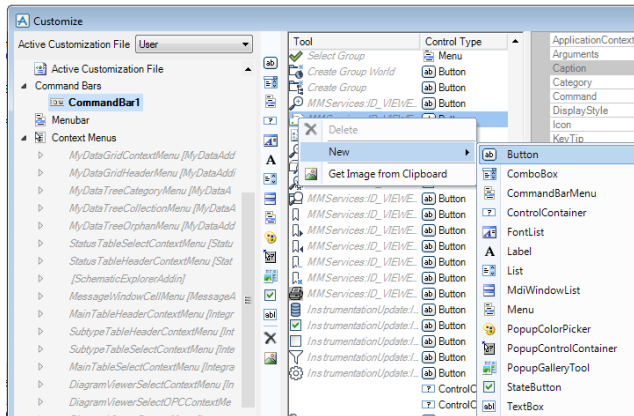
Menu	Entry for application on the applications menu (the title is used if this isn't specified)
Directory	Name of the application directory under %PMLUI%\module
Synonym	Synonym created for the directory (only used if the directory is specified)
Group	Group number (applications with the same group number are put into a submenu together)
GroupName	Description of submenu to appear on main menu

Copia para

Worked Example – Creating a Command Bar (UIC file modification)

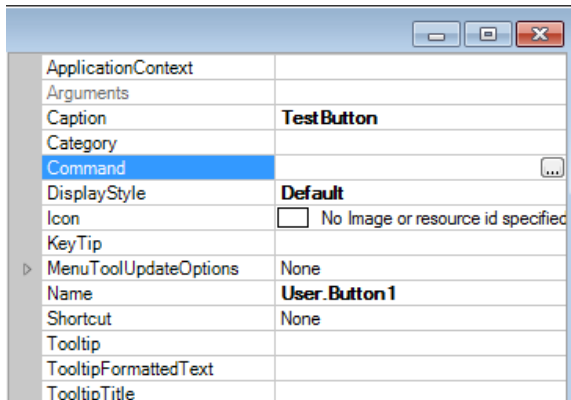
- 1
 
 - Open the Customisation Form. Do this by right clicking on an empty part of the toolbar and choose Customize...
- 2
 
 - On the Active Customization File, choose User.
 - This will save any changes to a specific user file (file path given on the right hand side of the form)
- 3
 
 - Right click on the Command Bars heading in bottom window, and click on New CommandBar
- 4
 
 - You will now see a preview of the command bar in the top left of the form (when the new CommandBar is selected)

5



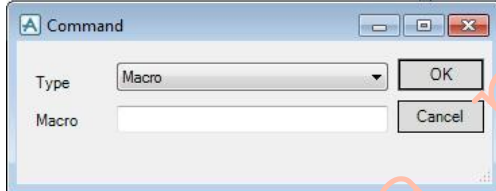
- Right click in the middle window in the form
- Create a **New > Button** from the menu

6



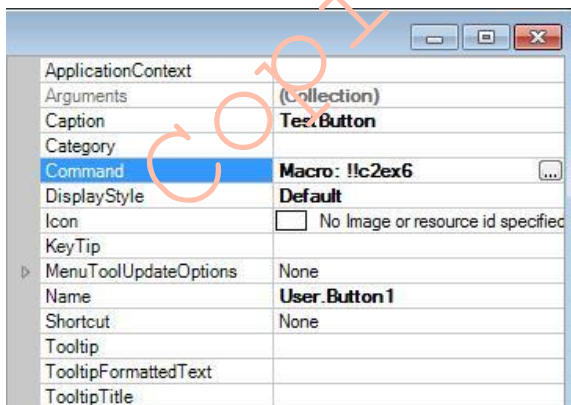
- With the new button selected in the middle window, look to the right of the form
- Select the line titled **Command** and click the small “...” button which appears on the right. This will let us set the command.

7



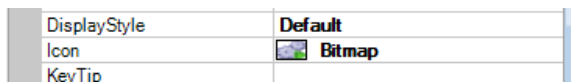
- A command can be a core command, command class or a PML command.
- Choose Macro, and type show !!c2ex6 (or equivalent form name)
- Close this form by pressing OK.

8



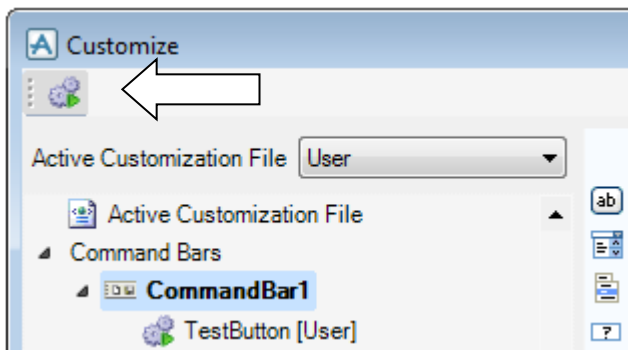
- The associated command is now set.
- Select the line titled Icon. Click the “...” button that appears to the right.
- Browse for the file *.png
- Click **Open**

9



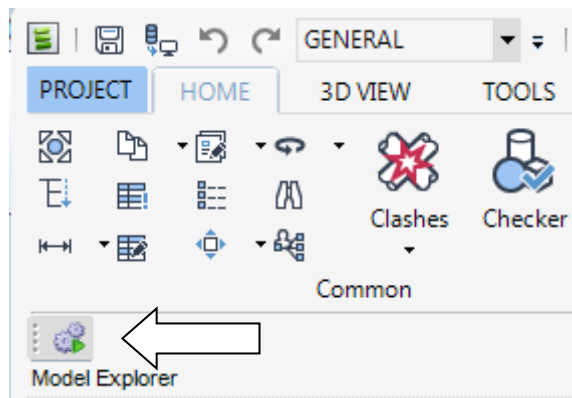
- The icon is now displayed in the right hand window.
- Set the tooltip to 'Show Nozzle Checker'
- Drag&Drop TestButton in to CommandBar1

10



- Select the new button in the middle window.
- Drag and Drop the new button beneath the new CommandBar object to make the association.
- Select the CommandBar object and notice the button is displayed in the preview.

11



- Apply and OK the form
- A new Command Bar will be displayed in the main application
- Test the button

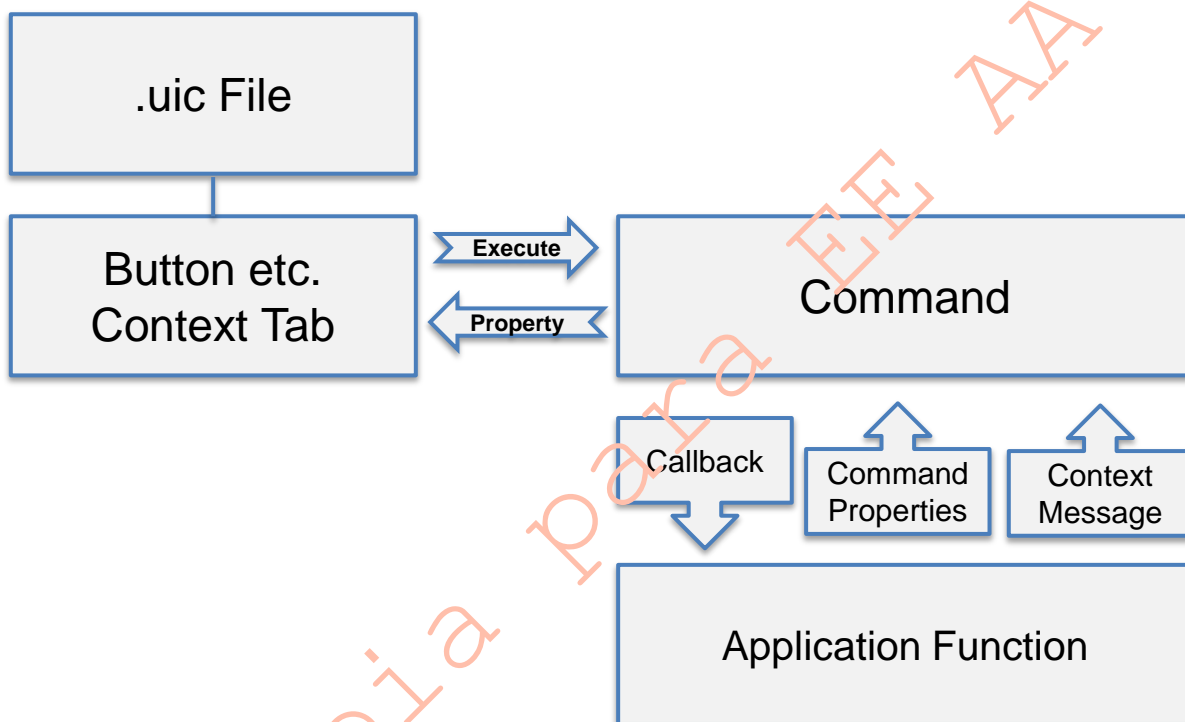
Copia para EE AA

10 Tabbed Menus in Model

The Tabs of an application user interface are defined in customisation files (see .NET Customisation User Guide) and built from tools associated with commands or by buttons that make direct calls to PML macros.

Note: Direct calls from tabbed menu buttons to PML macros is not recommended because there are several restrictions when working in this way. It is not possible to control the active state or the visibility of buttons that are not linked to command objects. Tools other than buttons, such as state buttons (toggles), combo boxes and tabs must be linked to the application by command objects in order to control their state.

Commands can be implemented as PML command objects or core command objects. The diagram shows the Command Object linking tools defined in user interface configuration (uic) file with application functions.



PML commands are similar in many ways to PML forms. Forms and commands are a type of global variable. This means that a form or command cannot have the same name as any other global variable or any other form or command. A form or command definition is also the definition of an object, so a form or command cannot have the same name as any other object type.

10.1 Defining a Command

A PML COMMAND is a PML 2 object defined in a file with extension .pmlcmd located in a directory somewhere under %PMLLIB% and with the same name as the command. It is defined in a similar way to a FORM using the SETUP syntax. A command has a number of properties - checked, enabled, value, visible and callbacks - execute, list, refresh, state. Properties are used to get/set the command state and callbacks may be added to handle command events like the execute event which is raised when the attached tool is clicked.

A command is defined using the SETUP syntax as follows:

```

setup command !!avevaPmlTraining
exit
-- Constructor
define method .avevaPmlTraining()

```



```
!this.key = | AVEVA.DesignGeneral.CommandPMLTraining|
!this.Execute = |execute|
Endmethod
```

The key and command members may be defined within the command setup. It is recommended that the command key and callbacks are defined in the constructor method.

The command key is used to uniquely identify the command when it is registered with the command manager. Similarly, any callbacks should be set before the command is registered (Note: once the command has been registered the key or callbacks cannot be changed).

The Execute callback is called when the attached tool is clicked and optionally can be passed a single STRING argument !args[0] from the tool.

In its simplest form a button requires a simple callback to be executed.

```
define method .execute(!args is ARRAY)
    show !!form
endmethod
```

10.2 Loading and Registering a Command

Once a command has been defined (and added to pml.index) it can be loaded using the PML LOAD syntax as follows:

PML LOAD COMMAND !!cmdbutton

which creates the command global instance. The command can then be registered with the Command Manager as follows

!!cmdbutton.register()

The key set in the constructor is !this.key and is used to uniquely identify the command. Once registered, it will be available in the application's Customize tool below this key in the Command window (see .NET Customisation Guide)

A function !!loadUserCommands() in %PMLLIB% is called when entering Model. User PML commands can be loaded and registered in this function. Add-in applications will typically load commands within the application itself in initialisation callbacks.

10.3 Attaching a Command to the UI

In the application's Customisation tool (see .NET Customisation Guide) the command will appear allowing it to be attached to one or more tools

Clicking on the attached tool will call the command's execute() callback passing any arguments in the !args[] array.

10.4 Killing a Command

Commands can be destroyed in the same way as a form using the KILL command as follows

KILL !!cmdbutton

This will delete the command global instance and, if the command has been registered, unregister the command from the command manager. (Note: commands like forms are not persisted across module switches).

Worked Example – Customize Tabbed Menu

- 1 Create command file (avevapmltraining.pmlcmd) and store in PMLLIB directory

```
setup command !!avevaPmlTraining

exit

define method .avevaPmlTraining()

    !this.key      = 'AVEVA.DesignGeneral.CommandPMLTraining'

    !this.execute = 'execute'

endmethod

define method .execute(!dummyArgs is ARRAY)

    show !!traexampleviewer

endmethod
```

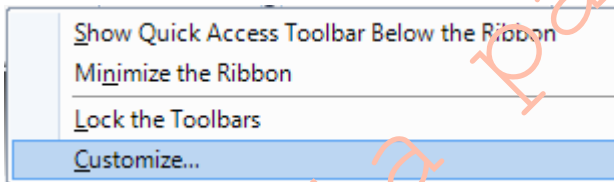
Use command **pml rehash all** Make sure, traexampleviewer.pmlfrm is existing in PMLLIB

- 2 Load and register command:

PML LOAD COMMAND !!avevaPmlTraining

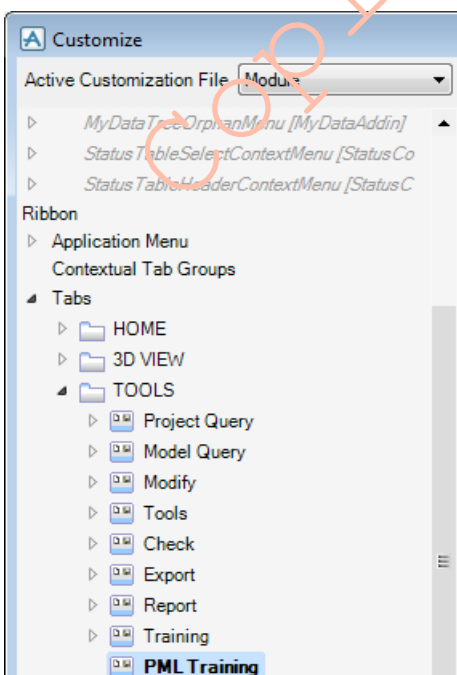
!!avevaPmlTraining.register()

- 3



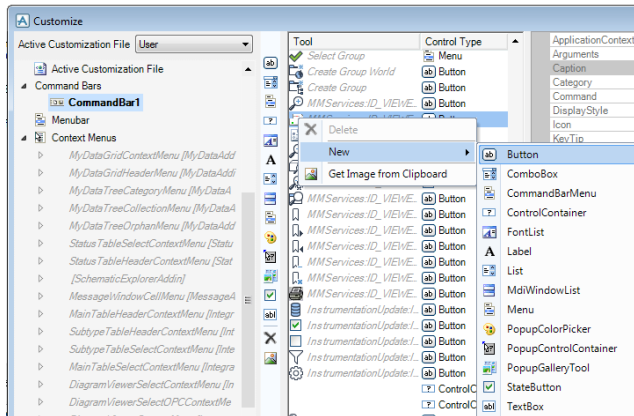
- Open the Customisation Form. Do this by right clicking on an empty part of the toolbar and choose Customize...

- 3



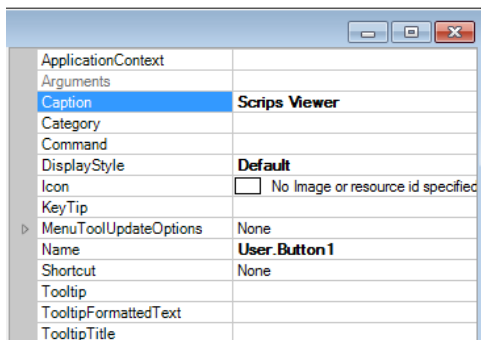
- On the Active Customization File, choose Module.
- Right click on the Ribbon -> Tabs -> Tools heading in bottom window, and click on New Tab. Change name to **PML Training**

6



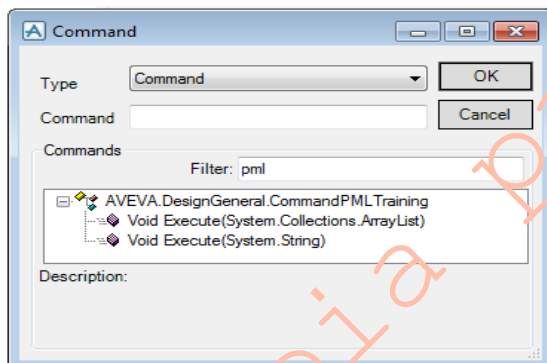
- Right click in the middle window in the form
- Create a **New > Button** from the menu

7



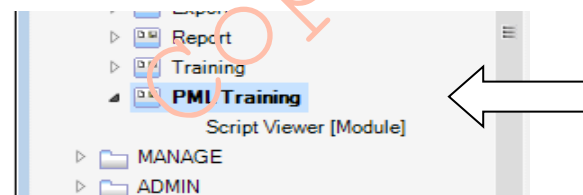
- With the new button, selected in the middle window, look to the right of the form
- Select the line titled **Command** and click the small “...” button which appears on the right. This will let us set the command.

8



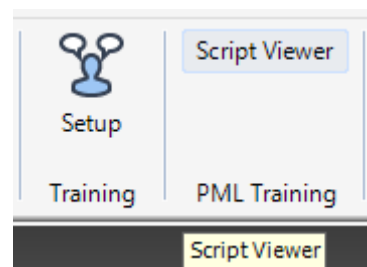
- Type filter **pml**
- Choose
AVEVA.DesignGeneral.CommandPMLTraining
- Close this form by pressing OK.

11



- Select the new button in the middle window.
- Drag and Drop the new button beneath the new PML Training object to make the association.

12



- Apply and OK the form
- A new menu item will be displayed in the main application.
- Press Script Viewer for test
- Modify by defining icon (choose any available icon file)
- Add in to menu calls to icon and colour viewers (see Appendix C)

Appendix A – Summary of Grid Control Gadget Methods

The following table shows all the current API calls to the grid control gadget. These can be used in a PML form or on the AVEVA E3D command line



For the latest API, refer to the .NET Customisation User Guide your current AVEVA E3D install

Name	Type	Purpose
Grid Population		
NetDataSource(String TableName, Array Attributes, Array Items)		Data Source constructor for database Attributes and database items. The Grid will populate itself with database attribute values.
NetDataSource(String TableName, Array Attributes, Array AttributeTitles, Array Items)		Data Source constructor for database Attributes and database items. The Grid will populate itself with database attribute values. The attribute titles can be added.
NetDataSource(String TableName, Array columns, Array of Array of rows)		Data Source constructor for column headings, and a set of rows. In this case the grid does NOT populate itself with database attribute values.
NetDataSource(String TableName, Array Attributes, Array Items, String Tooltips)		Data Source constructor for database Attributes and database items. The Grid will 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 rows. In this case the grid does NOT populate itself with databases attribute values.
NetDataSource(String TableName, string PathName)		Data Source constructor for import of an Excel XLS or CSV File
BindToDataSource(NetDataSource)		Bind Grid to NetDataSource
Grid Settings		
getDataSource()	NetDataSource	Returns the data source that has been set in the grid
SaveGridToExcel(String)		Save All Data to Excel File. Note that this will retain all grid groupings and layout of data. The String should specify the full pathname, e.g.: 'C:\AVEVA\Plant\Training\file.xls'.
SaveGridToExcel(String, Worksheet)		Save All Data to a Worksheet in an Excel File. Note that this will retain all grid groupings and layout of data.
FixedHeaders(Boolean)		Allow/disallow fixed headers. (Used when scrolling).
FixedRows(Boolean)		Allow/disallow fixing of rows. (Used when scrolling).
OutlookGroupStyle(Boolean)		Allow/disallow Outlook group facility
HideGroupByBox(Boolean)		Hide the groupBy box.....this will leave the Outlook style grouping in place and hence disallow the user from changing the grouping

Name	Type	Purpose
Grid Settings		
setGroupByColumn(colNum, Boolean)		Set/unset column in OutlookGroupStyle
ColumnExcelFilter(Boolean)		Allow/disallow Excel style filter on columns
ColumnSummaries(Boolean)		Allow/disallow Average, Count, etc... on numeric columns
HeaderSort(Boolean)		Allow alphabetic column sorting
ErrorIcon(Boolean)		Allow/disallow the display of the red icon in the cell if a data value is not available
SingleRowSelection(Boolean)		Set grid to single row selection
allGridEvents(Boolean)		Switch Grid Events On or Off
ClearGrid()		Remove data & column headings.
RefreshTable()		For a grid displaying Dabacon data this recalculates the content of every cell to refresh to the latest database state.
setAlternateRowColor(String)		Set alternate rows to a different colour
setAlternateRowColor(Red Num, Green Num, Blue Num)		Set alternate rows to a different colour. The default is: (251, 251, 255)
FeedbackSuccessColor(String)		Nominate (or query) the colour used to highlight cells successfully edited.
FeedbackFailColor(String)		Nominate (or query) the colour used to highlight cells where editing fails
SyntaxErrorColor(String)		Nominate (or query) the colour used to highlight cells containing syntax errors
ReadOnlyCellColor(String)		Nominate (or query) the colour used to highlight cells that cannot be edited.
resetCellFeedback()		Reset the cell editing feedback highlight colour and tooltip text.
DisplayUnsetAs(String)		Nominate (or query) a text string to be used in grid cells that hold "unset" attribute data.
DisplayNulrefAs(String)		Nominate (or query) a text string to be used in grid cells that hold "nulref" attribute data.
EditableGrid(Boolean)		Allow/disallow cells to be editable. Note that if the user is allowed to write data to the grid, then this data does not get written back to the database. The calling code is in control as to whether to either allow or disallow the change. This does not enable the user to add/remove rows, just to edit the content of the existing rows.
BulkEditableGrid(Boolean)		Allow/disallow cells to be bulk editable. Allows multiple cells to be selected, and Fill down/Fill up operations to be performed. Copy and Paste operations are also permitted to apply to multiple cells selected in the same column. The note about synchronising cell data with the data source as mentioned above for EditableGrid also applies to BulkEditableGrid.

Name	Type	Purpose
Grid Settings		
setLabelVisibility(Boolean)		Show/Hide label
CreateValueList(Name, Array for List)		Create a value list for the grid
ResetColumnProperties()		Reset column layout
SaveLayoutToXml(String)		Save the grid layout (not data) to a file on the file system.
loadGridFromXml(String)		Load a stored layout on the file system (not data) into a grid instance.
PrintPreview()		Opens an Infragistics style Print Preview Form (which allows printing)
saveGridToExcel(string excel file, string worksheet, string strHeader)		Where strHeader is a user supplied string which is outputted to the first row in the spreadsheet. The rest starts at row 2.
Rows		
GetSelectedRows()	Array of Array (rows)	Returns array of selected rows
GetSelectedRowTags()	Array of row tags	Get the selected row tags
GetRows()	Array of Array (rows)	Returns array of rows
GetRow(NUM)	Array	Get the cells in the Row number=NUM
GetRow(row tag)	Array	Get the cells in the Row with the specified row tag
GetRowTag(NUM)	STRING	Get the Tag ID of the row
setEditableRow(Integer, Boolean)		Allow/disallow a designated row to be editable
setRowColor(NUM, String)		Set row colour to a string (e.g. 'red')
setRowTooltip(row, String)		Set row tooltip to the specified string
setSelectedRowTags(Array rows)		Programmatically select the given row tags
selectRow(NUM)		Programmatically select the row=NUM
getNumberRows()	REAL	Get the number of rows in the grid
clearRowSelection()		Clear row selection
addRow(Array data)		Add a <u>single</u> row of data. If DB Element grid then only first array data is read. If not, then the Array data represents the data in each cell of the row,
deleteSelectedRows()		Delete selected rows
deleteRows(Array data)		Delete one or more rows given by the "Tags" in the array.
deleteRows(Array data)		Delete one or more rows given by the "Tags" in the array.
scrollSelectedRowToView(Boolean)		Scroll the selected row into view
setRowVisibility(double rowNum, Boolean)		Turn on/off the visibility of the specified row

Name	Type	Purpose
Rows		
IsRowVisible(double rowNum)		Query the visibility of the specified row
Columns		
SetColumnMask(NUM, STRING(MASK))		Set the Mask on column number=NUM (See valid masks in table below)
getColumnMask(NUM)	STRING	Get the mask on the specified column
setEditableColumn(Integer, Boolean)		Allow/disallow a designated column to be editable
GetNumberColumns()	REAL	Return the number of columns
AssignValueListToColumn(Name, Column NUM)		Assign the value list to a column
GetColumns()	Array of STRING	Get the columns in the grid
GetColumn(NUM)	Array	Get the cells in the Column number=NUM
setColumnColor(NUM, String)		Set column colour to a string (e.g. 'red')
setColumnImage(NUM, String)		Set all cell images in column to the pathname of the specified icon
setColumnVisibility(colNum, Bool)		Show/Hide column
getColumnKey(NUM)	STRING	Get the Key of the column
setNameColumnImage()		Displays standard icons in a "NAME" attribute column when database items are used.
AutoFitColumns()		Resize columns to fit widest text
SetColumnWidth(Column NUM, Column Width)		Set column number to a set width
ExtendLastColumn(Bool)		Extend the last column, or not
ResizeAllColumns()		Resize columns to fit in available width of form
IsColumnVisible(double colNum)		Query the visibility of the specified column.
Cells		
GetCell(ROW, COL)	STRING	Get the content of cell(ROW,COL) by row and column number
GetCell(row tag, column tag)	STRING	Get the content of cell(ROW,COL) by row tag and column tag
setCellColor(row, col, String)		Set cell colour to a string (e.g. 'red')
setEditableCell(Row NUM, Col NUM, Boolean)		Allow/disallow a designated cell to be editable
setCellTooltip(row, col, string)		Set cell tooltip to the specified string
setCellImage(row, col, String)		Set cell image to the pathname of the specified icon
setCellValue(rowNum, ColNum, STRING)		Programmatically set a cell value to a string

Name	Type	Purpose
Cells		
AssignValueListToCell(Name, Row NUM, Column NUM)		Assign the value list to a cell
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.

Input masks 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.
&	STRING
>	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.

Character	Description
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.
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.

Copia para EE AA

Appendix B – Exercise Solutions (Example Code)

This appendix contains examples of code that provide solutions to each of the exercises.

- ① *Local variables have been used to shorten lines to avoid “word wrap” and improve clarity. Look for the definition of these variables within the examples and where/how they are used.*

Appendix B1 - Example c2ex1.pmlfrm

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex1.pmlfrm
--
-- Description:
-- Example form solution to Exercise 1
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-- Form definition
--
setup form !!c2ex1 dialog resizeable

-- Set the form title and the initialisation call
!this.formTitle = |Example Form - Exercise 1|
!this.initcall = |!this.init()|

-- Set the default path for gadget layout
path down

-- Input frame (LHS)
frame .inputFrame |Inputs| at x 0 anchor T+L+B

-- Top Frame - Temperature Input
frame .tempInputFrame |Temperature conversion (Input)| at x 1
text .tempInput |Temperature| call |!this.temperatureConvert()| width 10 is REAL
frame .tempChoiceFrame panel at xmax.tempInput ymin.tempInputFrame + 0.5
rToggle .celsius |°C| tagwid 3 at xmax.tempInput + 3 ymin.tempInput
rToggle .fahrenheit |°F| tagwid 3 at xmax.celsius ymin.tempInput
exit
exit

-- Middle Frame - Temperature Range
frame .tempRangeFrame |Temperature Range| at x 1 width.tempInputFrame
text .minimum |Minimum| call |!this.check(1)| at x 1 width 10 is REAL
text .maximum |Maximum| call |!this.check(2)| at x 1 width 10 is REAL
!format = |format !!INTEGERFMT|
text .step |Step Size| call |!this.check(0)| at x 1 width 10 is REAL $!format
!bPos = |xmax.step + 1 ymin.step|
button .fillFahrenheit linklabel |Fill with °F to °C >>| at $!bPos wid 12
!bPos = |xmin.fillFahrenheit ymin.fillFahrenheit - size|
button .fillCelsius linklabel |Fill with °C to °F >>| at $!bPos wid 12
exit

-- Lower Frame - Temperature Split
frame .stringSplitFrame |Temperture Split| anchor L+B+R at x 1 width.tempInputFrame
text .stringInput |Input| width 24 is STRING
text .delimiter |Delimiter| width 10 is STRING
!bPos = |xmax.delimiter + 1 ymin.delimiter|
button .split linklabel |Split temperature >>| at $!bPos wid 12
exit
exit

-- (Continues on next page)
```

Fix1: U not valid

Fix2: REAL with one L

```
-- Results Frame (RHS)
frame .results |Results| at xmax + 1 y 0 anchor ALL

-- Top frame - Temperature output
!fSize = |width.tempInputFrame height.tempInputFrame|
frame .tempOutputFrame |Temperature conversion (Output)| anchor L+T+R at x 1 $!fSize
text .tempOutput |Temperature| call || width 10 is REAL
para .unit at xmax.tempOutput ymin.tempInput text || width 5
exit

-- Middle frame - Temperature results
!fSize = |width.tempRangeFrame height.tempRangeFrame|
frame .tempRangeOutFrame |Temperature Results| anchor L+T+R at x1 $!fSize
list .tempList dock fill width 1 height 1
exit

-- Lower frame - Split temperature
!fPos = |x1 ymin.stringSplitFrame|
!fSize = |width.stringSplitFrame height.stringSplitFrame|
frame .stringSplitOutFrame |Temperature Split Result| anchor L+B+R at $!fPos $!fSize
text .number |No. of Temp| width 10 is REAL format !!INTEGERFMT
text .result |Result| width.stringInput is STRING
exit
exit
exit

-- Constructor Method - Set the callbacks on the form
define method .c2ex1()
!this.tempChoiceFrame.callback = |!this.temperatureConvert()|
!this.fillFahrenheit.callback = |!this.fill('Fahrenheit')|
!this.fillCelsius.callback = |!this.fill('Celsius')|
!this.split.callback = |!this.split()|
endmethod

-- Initialisation Method - Set the initial gadget values and run the methods
define method .init()
!this.tempInput.val = 0
!this.tempChoiceFrame.val = 1
!this.minimum.val = 0
!this.maximum.val = 100
!this.step.val = 25
!this.stringInput.val = |10°C/30°C/20°C/5°C|
!this.delimiter.val = | / |
!this.temperatureConvert()
!this.fill(|Celsius|)
!this.split()
endmethod

-- Method .celsiusToFahrenheit(REAL)REAL - Convert the number to Fahrenheit
define method .celsiusToFahrenheit(!celsius is REAL) is REAL
!fahrenheit = !celsius * 1.5 + 32
return !fahrenheit
endmethod

-- Method .fahrenheitToCelsius(REAL)REAL - Convert the number to Celsius
define method .fahrenheitToCelsius(!fahrenheit is REAL) is REAL
!celsius = (!fahrenheit - 32) / 1.8
return !celsius
endmethod

-- Method .temperatureConvert() - Convert input temperature based on radio button
define method .temperatureConvert()
if !this.tempChoiceFrame.val.eq(1) then
!this.tempOutput.val = !this.CelsiusToFahrenheit(!this.tempInput.val)
!this.unit.val = |°F|
elseif !this.tempChoiceFrame.val.eq(2) then
!this.tempOutput.val = !this.FahrenheitToCelsius(!this.tempInput.val)
!this.unit.val = |°C|
endif
endmethod

-- (Continues on next page)
```

Fix3: Width+height at the end of

Fix4: Incorrect constructor name

A new init method applies default values to the form

Fix5: F and C the wrong way around

```

-----
-- Method .fill(STRING) - Fill the temperature conversion list based on argument --
-----
define method .fill(!whichTemperature is STRING)
!n = 0
-- Loop through the supplied values and build array for the list
do !temperature from !this.minimum.val to !this.maximum.val by !this.step.val
!n = !n + 1
!tempArray[!n][1] = !n.string()
!tempArray[!n][2] = !temperature.string()
!tempArray[!n][3] = |=|
if !whichTemperature.eq(|Celsius|) then
!fahrenheit = !this.celsiusToFahrenheit(!temperature)
!tempArray[!n][4] = STRING(!fahrenheit,!!realFMT)
else
!celsius = !this.fahrenheitToCelsius(!temperature)
!tempArray[!n][4] = STRING(!celsius,!!realFMT)
endif
enddo

-- Set the list headings based on the argument
if !whichTemperature.eq(|Celsius|) then
!headings = |No. Celsius = Fahrenheit|
else
!headings = |No. Fahrenheit = Celsius|
endif
-- Display the collected data
!this.tempList.setHeadings(!headings.split())
!this.tempList.setRows(!tempArray)
endmethod

-----
-- Method .check(REAL) - Based on the type of check, ensure the entered values are ok --
-----
define method .check(!flag is REAL)

-- Check the step value, if = 0 - then make it 1
if !flag.eq(0) then
if !this.step.val.eq(0) then
!this.step.val = 1
endif

-- For others....
elseif !flag.eq(1).or(!flag.eq(2)) then
-- Check that text boxes have values entered in them
if !this.maximum.val.unset() then
!this.maximum.val = !this.minimum.val + !this.step.val
elseif !this.minimum.val.unset() then
!this.minimum.val = !this.maximum.val - !this.step.val
endif
-- Check that the values aren't lower than absolute zero.
if (!this.minimum.val.lt(-273)) then
!this.minimum.val = -273
if (!this.maximum.val.leq(!this.minimum.val)) then
!this.maximum.val = -272
endif
elseif (!this.maximum.val.lt(-273)) then
!this.minimum.val = -273
!this.maximum.val = -272
endif
-- Check that the maximum value is larger than the minimum value
if (!this.maximum.val.leq(!this.minimum.val)) then
if !flag.eq(1) then
!this.maximum.val = !this.minimum.val + !this.step.val
elseif !flag.eq(2) then
!this.minimum.val = !this.maximum.val - !this.step.val
endif
endif
endif
endif
endmethod

-- (Continues on next page)

```

```

-----
-- Method .split() - Split the user entered string, convert and concatenate --
-----
define method .split() -----
  -- Split the string
  !split = !this.stringInput.val.trim().split(!this.delimiter.val)
  !this.number.val = !split.size()
  !result = ||
  -- Loop through the split
  do !n index !split
    -- If it's Celsius....convert to Fahrenheit
    if !split[!n].substring(!split[!n].length()).upcase().eq(|C|) then
      !temp = !split[!n].substring(1, !split[!n].length() - 2).real()
      !result = !result & !this.celsiusToFahrenheit(!temp).string(!INTEGERFMT) & |°F |
    else
      !temp = !split[!n].substring(1, !split[!n].length() - 2).real()
      !result = !result & !this.fahrenheitToCelsius(!temp).string(!INTEGERFMT) & |°C |
    endif
  enddo
  -- Return the concatenated string back to the form
  !this.result.val = !result
endmethod
-----
-- End of Form definition and methods --
-----
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----

```

New method to split the string

Copia para EEAA

Appendix B2 - Example c2ex2.pmlfrm

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex2.pmlfrm
--
-- Description:
-- Example form solution to Exercise 2
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex2 dialog resizable

-- Set the form title and the initialisation call
!this.formTitle = |Example Form - Exercise 2|
!this.initcall = |!this.init()|
-- Set the default path for gadget layout
path down
frame .tabset TABSET anchor all

-- Input frame (LHS)
frame .inputFrame |Inputs| at x 0 y 0 anchor all

-- Top Frame - Temperature Input
frame .tempInputFrame |Temperature conversion (Input)| at x 1
text .tempInput |Temperature| call |!this.temperatureConvert()| width 10 is REAL
frame .tempChoiceFrame panel at xmax.tempInput ymin.tempInput
rToggle .celsius |°C| tagwid 3 at xmax.tempInput + 3 ymin.tempInput
rToggle .fahrenheit |°F| tagwid 3 at xmax.celsius ymin.tempInput
exit
exit

-- Middle Frame - Temperature Range
frame .tempRangeFrame |Temperature Range| at x 1 width.tempInputFrame
text .minimum |Minimum| call |!this.check(1)| at x 1 width 10 is REAL
text .maximum |Maximum| call |!this.check(2)| at x 1 width 10 is REAL

!format = |format !!INTEGERFMT|
text .step |Step Size| call |!this.check(0)| at x 1 width 10 is REAL $!format
!bPos = |xmax.step + 1 ymin.step|

button .fillFahrenheit linklabel |Fill with °F to °C >>| at $!bPos wid 12
!bPos = |xmin.fillFahrenheit ymin.fillFahrenheit - size|
button .fillCelsius linklabel |Fill with °C to °F >>| at $!bPos wid 12
exit

-- Lower Frame - Temperature Split
frame .stringSplitFrame |Temperture Split| anchor L+B+R at x 1 width.tempInputFrame
text .stringInput |Input| width 24 is STRING
text .delimiter |Delimiter| width 10 is STRING
!bPos = |xmax.delimiter + 1 ymin.delimiter|
button .split linklabel |Split temperature >>| at $!bPos wid 12
exit
exit

-- Results Frame (RHS)
frame .results |Results| at x0 y 0 anchor ALL

-- Top frame - Temperature ouput
!fSize = |width.tempInputFrame height.tempInputFrame|
frame .tempOutputFrame |Temperature conversion (Output)| anchor L+T+R at x 1 $!fSize
text .tempOutput |Temperature| call || width 10 is REAL
para .unit at xmax.tempOutput ymin.tempInput text || width 5
exit

-- Middle frame - Temperature results
!fSize = |width.tempRangeFrame height.tempRangeFrame|
frame .tempRangeOutFrame |Temperature Results| anchor L+T+R at x1 $!fSize
list .tempList dock fill width 1 height 1
exit

--(Continues on next page
```

Tabset addd


```

-- Lower frame - Split temperature
!fPos = |x1 ymin.stringSplitFrame|
!fSize = |width.stringSplitFrame height.stringSplitFrame|
frame .stringSplitOutFrame |Temperature Split Result| anchor L+B+R at $!fPos $!fSize
text .number |No. of Temp| width 10 is REAL format !!INTEGERFMT
text .result |Result | width.stringInput is STRING
exit

exit
exit

!pixmap = |pixmap width 16 height 16|
!link = |linkLabel width 10|
para .paAccept at xmin.tempInputFrame ymax+0.5 anchor L+B $!pixmap
button .lkAccept |Accept changes| at xmax.paAccept+1 ymin anchor L+B $!link
button .lkDiscard |Discard changes | at xmax.tempInputFrame - size ymin anchor R+B $!link
para .paDiscard at xmin.lkDiscard - 1.5 * size ymin anchor R+B $!pixmap

-- Form member to store the form values
member .data is ARRAY

-- MENU objects to be used as context menus on the list gadget
!menu = !this.newMenu(|celsiusMenu|)
!menu.add(|Callback|, |Switch to °F = °C|, |!this.fill('Fahrenheit')|)
!menu = !this.newMenu(|fahrenheitMenu|)
!menu.add(|Callback|, |Switch to °C = °F|, |!this.fill('Celsius')|)

exit

-- Constructor Method - Set the callbacks on the form
define method .c2ex2()
!this.results.callback = |!this.tabCall(|
!this.paAccept.addPixmap(!pml.getPathName('accept.png'))
!this.paDiscard.addPixmap(!pml.getPathName('discard.png'))
!this.lkAccept.callback = |!this.setData(1)|
!this.lkDiscard.callback = |!this.setData(2)|
endmethod

-- Initialisation Method - Set the initial gadget values and run the methods
define method .init()
!this.tempInput.val = 0
!this.tempChoiceFrame.val = 1
!this.minimum.val = 0
!this.maximum.val = 100
!this.step.val = 25
!this.stringInput.val = |10°C/30°C/20°C/5°C|
!this.delimiter.val = |||
!this.setData(1)
endmethod

-- Method .tabCall(GADGET, STRING) - Open callback on Results tab
define method .tabCall(!gadget is GADGET, !type is STRING)
-- If the results tab is shown, then run the form methods
if !type.eq(|SHOWN|) then
!this.temperatureConvert()
!this.fill(|Celsius|)
!this.split()
endif
endmethod

-- Method .celsiusToFahrenheit(REAL)REAL - Convert the number to Fahrenheit
define method .celsiusToFahrenheit(!celsius is REAL) is REAL
!fahrenheit = !celsius * 1.8 + 32
return !fahrenheit
endmethod

-- Method .fahrenheitToCelsius(REAL)REAL - Convert the number to Celsius
define method .fahrenheitToCelsius(!fahrenheit is REAL) is REAL
!celsius = (!fahrenheit - 32) / 1.8
return !celsius
endmethod
--(Continues on next page)

```

New menu objects

New Accept/Discard buttons

Open Callback on results tab

```

-----
-- Method .temperatureConvert() - Convert input temperature based on radio button --
-----
define method .temperatureConvert()
  if !this.tempChoiceFrame.val.eq(1) then
    !this.tempOutput.val = !this.CelsiusToFahrenheit(!this.tempInput.val)
    !this.unit.val = |°F|
  elseif !this.tempChoiceFrame.val.eq(2) then
    !this.tempOutput.val = !this.FahrenheitToCelsius(!this.tempInput.val)
    !this.unit.val = |°C|
  endif
endmethod

-----
-- Method .fill(STRING) - Fill the temperature conversion list based on argument --
-----
define method .fill(!whichTemperature is STRING)
  !n = 0
  -- Loop through the supplied values and build array for the list
  do !temperature from !this.minimum.val to !this.maximum.val by !this.step.val
    !n = !n + 1
    !tempArray[!n][1] = !n.string()
    !tempArray[!n][2] = !temperature.string()
    !tempArray[!n][3] = |=|
    if !whichTemperature.eq(|Celsius|) then
      !fahrenheit = !this.celsiusToFahrenheit(!temperature)
      !tempArray[!n][4] = STRING(!fahrenheit,!!realFMT)
    else
      !celsius = !this.fahrenheitToCelsius(!temperature)
      !tempArray[!n][4] = STRING(!celsius,!!realFMT)
    endif
  enddo
  -- Set the list headings based on the argument
  if !whichTemperature.eq(|Celsius|) then
    !headings = |No. Celsius = Fahrenheit|
    !this.tempList.setPopup(!this.celsiusMenu)
  else
    !headings = |No. Fahrenheit = Celsius|
    !this.tempList.setPopup(!this.fahrenheitMenu)
  endif
  -- Display the collected data
  !this.tempList.setHeadings(!headings.split())
  !this.tempList.setRows(!tempArray)
endmethod

-----
-- Method .check(REAL) - Based on the type of check, ensure the entered values are ok --
-----
define method .check(!flag is REAL)
  -- Check the step value, if = 0 - then make it 1
  if !flag.eq(0) then
    if !this.step.val.eq(0) then
      !this.step.val = 1
    endif
  endif

  -- For others....
  elseif !flag.eq(1).or(!flag.eq(2)) then
    -- Check that text boxes have values entered in them
    if !this.maximum.val.unset() then
      !this.maximum.val = !this.minimum.val + !this.step.val
    elseif !this.minimum.val.unset() then
      !this.minimum.val = !this.maximum.val - !this.step.val
    endif
    -- Check that the values aren't lower than absolute zero.
    if (!this.minimum.val.lt(-273)) then
      !this.minimum.val = -273
      if (!this.maximum.val.leq(!this.minimum.val)) then
        !this.maximum.val = -272
      endif
    elseif (!this.maximum.val.lt(-273)) then
      !this.minimum.val = -273
      !this.maximum.val = -272
    endif
    -- Check that the maximum value is larger than the minimum value
    if (!this.maximum.val.leq(!this.minimum.val)) then
      if !flag.eq(1) then
        -- (Continues on next page)
      endif
    endif
  endif
endmethod

```

```

        !this.maximum.val = !this.minimum.val + !this.step.val
    elseif !flag.eq(2) then
        !this.minimum.val = !this.maximum.val - !this.step.val
    endif
endif
endif
endmethod

-----
-- Method .split() - Split the user entered string, convert and concatenate --
-----
define method .split()
    -- Split the string
    !split = !this.stringInput.val.trim().split(!this.delimiter.val)
    !this.number.val = !split.size()
    !result = ""
    -- Loop through the split
    do !n index !split
        -- If it's Celsius...convert to Fahrenheit
        if !split[!n].substring(!split[!n].length() - 2).uppercase().eq("C") then
            !temp = !split[!n].substring(1, !split[!n].length() - 2).real()
            !result = !result & !this.celsiusToFahrenheit(!temp).string(!INTEGERFMT) & "°F |"
        else
            !temp = !split[!n].substring(1, !split[!n].length() - 2).real()
            !result = !result & !this.fahrenheitToCelsius(!temp).string(!INTEGERFMT) & "°C |"
        endif
    enddo
    -- Return the concatenated string back to the form
    !this.result.val = !result
endmethod

-----
-- Method .setData(REAL) - Store or retrieve the gadget values from the storage array --
-----
define method .setData(!flag is REAL)
    -- Either save the values, or bring them back (array of strings)
    if !flag.eq(1) then
        !this.data[1] = !this.tempInput.val
        !this.data[2] = !this.tempChoiceFrame.val
        !this.data[3] = !this.minimum.val
        !this.data[4] = !this.maximum.val
        !this.data[5] = !this.step.val
        !this.data[6] = !this.stringInput.val
        !this.data[7] = !this.delimiter.val
    else
        if !this.data.size().eq(7) then
            !this.tempInput.val = !this.data[1]
            !this.tempChoiceFrame.val = !this.data[2]
            !this.minimum.val = !this.data[3]
            !this.maximum.val = !this.data[4]
            !this.step.val = !this.data[5]
            !this.stringInput.val = !this.data[6]
            !this.delimiter.val = !this.data[7]
            !this.temperatureConvert()
            !this.fill("Celsius")
            !this.split()
        endif
    endif
endmethod

-----
-- End of Form definition and methods --
-----
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----

```

Method to transfer to and from
the .data member

Appendix B3 - Example c2ex3.pmlfrm

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex3.pmlfrm
--
-- Description:
-- Example form solution to Exercise 3
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex3 dialog docking

-- Set the form title
!this.formTitle = |FILE Objects - Exercise 3|

-- Define the gadgets for the form
textpane .txtp at x 0 ymax anchor all width 60 height 15
para .para at xmin ymax anchor b+l+r text || width 30
button .load |Load| at xmax.txtp - size ymin anchor b+r pixmap width 16 height 16
button .save |Save| at xmin - size ymin anchor b+r pixmap width 16 height 16

-- Form member to record which file is being used
member .file is FILE

exit

-- Constructor Method - Set the images and callbacks on the form
--
define method .c2ex3()
-- Apply the images to the pixmap buttons
!this.load.addPixmap(!!pml.getPathName(|openbrowser.png|))
!this.save.addPixmap(!!pml.getPathName(|saveview16.png|))
-- Apply tooltips to the icon buttons for clarity.
!this.load.setToolTip(|Load text file|)
!this.save.setToolTip(|Save text file|)
-- Set the callback to run the standard function !!filebrowser
!dir = 'C:\AVEVA\Plant\Training'
!this.load.callback = ||!fileBrowser('| & !dir & |', '', 'Load text file', $
TRUE, '!!c2ex3.load(!!fileBrowser.file)')|
!this.save.callback = ||!fileBrowser('| & !dir & |', '', 'Save text file', $
FALSE, '!!c2ex3.save(!!fileBrowser.file)')|
endmethod

-- Method .load(FILE) - Read the chosen file into the textpane
--
define method .load(!file is FILE)
-- Store and read the file
!this.file = !file
!this.txtp.val = !this.file.readFile()
-- Display the filename
!this.para.val = !this.file.string()
endmethod

-- Method .save(FILE) - Save the chosen file from the textpane
--
define method .save(!file is file)
!this.file = !file
!this.file.writeFile('OVER', !this.txtp.val)
!this.para.val = !this.file.string()
endmethod

-- End of Form definition and methods
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-----
```

Appendix B4 - Example c2ex4.pmlfrm

```

--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex4.pmlfrm
--
-- Description:
-- Example form solution to Exercise 4
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex4 dialog resiz

-- Set the form title and initialisation call
!this.formTitle = |Equipment Checker|
!this.initcall = |!this.init()|

-- Create a MENU object for use as a context menu on the list gadget
!this.newMenu(|popupMenu|)
!popupCall = '!!CE = !this.nozzleList.selection().dbref()'
!this.popupMenu.add('CALLBACK', 'Go to Nozzle', !popupCall)

-- Define the gadgets at the top of the form
button .update linklabel |Update| call |!this.init()| width 5
para .title at xmax ymin anchor t+l+r text |Available Equipments below | width 34
line .line at xmin.update ymax anchor t+l+r horizontal width 12

!buttPos = |xmax - size ymax anchor b+r|
!anch1 = |anchor t+l+r |
!anch2 = |anchor b+l+r|

-- Define the gadgets to choose the equipments from
combo .equip at xmin.update ymax+0.2 |Select an Equipment| tagwidth 15 $!anch1 width 24
list .nozzleList at xmin.update ymax+0.2 anchor a+l width 40 length 5
button .refresh linklabel |Refresh Checks| at $!buttPos width 9.6

-- Define the gadgets to allow the attributes to be altered
!tag = |Equipment Attributes|
textpane .atta |$!tag| at xmin.nozzleList ymax-0.5 $!anch2 width 42 height 3.5
button .updateAtta linklabel |Update Attributes| at $!buttPos width 10.3

exit

-- Constructor Method - Set the images and callbacks on the form
--
define method .c2ex4()
-- Set the titles and context menu on the list
!title = |Nozzles/Connected?/Attached?/Aligned?/Size?|
!this.nozzleList.setHeadings(!title.split(|/|))
!this.nozzleList.setpopup(!this.popupMenu)
-- Pre-fill-in some attributes to be edited
!info[1] = 'Description - Unset'
!info[2] = 'Function - Unset'
!info[3] = 'Purpose - Unset'
!this.atta.val = !info
-- Set the gadget callbacks
!this.equip.callback = |!this.setEquip(|
!this.refresh.callback = |!this.checkNozzles()|
!this.updateAtta.callback = |!this.updateAtt(2)|
endmethod

-- Initialisation Method - Collect all equi for level and display nozzles
--
define method .init()
-- If at SITE level, collect for the site otherwise do it for ZONE
if !!ce.type.eq(|SITE|) then
!level = |SITE|
else
!level = |ZONE|
endif
--(Continues on next page)

```

```
-- Collect the equipment using PML 1 style collection syntax
VAR !coll COLL ALL EQUIP FOR $!level
handle ANY
    !!alert.error(|Make sure you are at an EQUI, ZONE or SITE element|)
elsehandle NONE
    -- If no equipment are found, and we're at a ZONE try collecting for SITE
    if !coll.size().eq(0) and !level.eq(|ZONE|) then
        !!alert.message(|No equipment found in current ZONE, so whole SITE will be searched|)
        VAR !coll COLL ALL EQUIP FOR SITE
        !level = |SITE|
    endif
    -- Update the title paragraph gadget
    !this.title.val = |Available Equipments below | & !level
    -- Evaluate the fullnames of the equis and display the information
    VAR !name EVAL FLNN FOR ALL FROM !coll
    !this.equip.dtext = !name
    !this.equip.rtext = !coll
    -- Collect the nozzles
    !this.collectNozzles()
endhandle
endmethod

-- Method .collectNozzles() - For the selected EQUI, collect all the nozzles
-----
define method .collectNozzles()
    -- Get the selected EQUI element
    !equipRef = !this.equip.selection().dbref()
    if !equipRef.unset() then
        -- If the element is unset, then clear the nozzle list
        !this.nozzleList.clear()
    else
        -- Perform a PML2 style collection for the nozzles
        !nozzColl = object COLLECTION()
        !nozzColl.type('NOZZ')
        !nozzColl.scope(!equipRef)
        !results = !nozzColl.results()
        !this.nozzleList.dtext = !results.evaluate(object block ('!results[!evalIndex].flnn'))
        !this.nozzleList.rtext = !results.evaluate(object block ('!results[!evalIndex].string()'))
        -- Check the nozzles and update the results
        !this.checkNozzles()
        !this.updateAtt(1)
    endif
endmethod

-- Method .setEquip(GADGET, STRING) - Open callback on combo gadget to check user entry --
-----
define method .setEquip(!gad is gadget, !event is STRING)
    -- If the user has typed into the gadget
    if !event.eq('VALIDATE') then
        !userInput = !this.equip.displayText()
        -- Loop through available equipments and choice the nearest match
        do !n index !this.equip.dtext
            !chrs = !userInput.length()
            !test = !this.equip.dtext[!n].upcase().substring(1, !chrs)
            if !userInput.upcase().eq(!test) then
                !this.equip.val = !n
                break
            endif
        enddo
    endif
    -- Recollect the nozzles on the newly selected equipment
    !this.collectNozzles()
endmethod

-- Method .checkNozzles() - Check the collected nozzles and display the results
-----
define method .checkNozzles()
    if !this.nozzleList.rtext.set() then
        -- Loop through the collected nozzles
        do !n index !this.nozzleList.rtext
            !nozz = !this.nozzleList.rtext[!n].dbref()
            -- set the default result
            do !m from 2 to 4
                !result[!m] = |N/A|
            enddo
            -- Check 1: is the connection valid
        enddo
    endif
endmethod
-- (Continues on next page)
```

```

if !nozz.cref.unset().or(!nozz.cref.badref()) then
    !result[1] = |Check|
else
    !result[1] = |OK|
    !end = |t|
    if !nozz.cref.href.eq(!nozz) then
        !end = |h|
    endif
    -- Check 2: is the nozzle and the same position as the attached
    if !nozz.pos.wrt( /* ).eq(!nozz.cref.attribute(!end & |pos|).wrt( /* )) then
        !result[2] = |OK|
    endif
    -- Check 3: is the nozzle and the same direction as the attached
    if !nozz.pdir[1].wrt( /* ).eq(!nozz.cref.attribute(!end & |dir|).wrt( /* )) then
        !result[3] = |OK|
    endif
    -- Check 4: is the nozzle and the same size as the attached
    if !nozz.cpar[1].eq(!nozz.cref.attribute(!end & |bore|).real()) then
        !result[4] = |OK|
    endif
endif
-- Record the check results for this nozzle
!nozzleInfo[!n][1] = !nozz.flnn
!nozzleInfo[!n].appendArray(!result)
enddo
-- Apply the results back to the list gadget
!rtext = !this.nozzleList.rtext
!this.nozzleList.setRows(!nozzleInfo)
!this.nozzleList.rtext = !rtext
endif
endmethod

-----
-- Method .updateAtt(REAL) - Check the collected nozzles and display the results --
-----

define method .updateAtt(!flag is REAL)
    -- Get the selected piece of equipment and the available attributes
    !equip = !this.equip.selection().dbref()
    !availAtts = !equip.attributes()
    -- Update the textpane title
    !this.atta.tag = !equip.flnn & ' Attributes'
    !rows = !this.atta.val
    -- Loop through the rows of the textpane to extract and use the information
    !out = ARRAY()
    do !n index !rows
        -- Split the line on a "-"
        !split = !rows[!n].split('-')
        !attrib = !split[1].trim()
        -- Evaluate the available attributes so they are the same string length as the entered
        !block = object block (!availAtts[!evalIndex].upcase().substring(1, !attrib.length()))
        !avail = !availAtts.evaluate(!block)
        -- If the attribute can be found, then use it
        if !avail.findfirst(!attrib.upcase()).set() then
            if !flag.eq(1) then
                -- If the flag = 1 then update the textpane with the current attribute value
                !value = !equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())])
                !out.append(!attrib & | - | & !value)
            elseif !flag.eq(2) then
                -- If the flag = 2 then assign the value to the attribute
                !val = !split[2].trim()
                !equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())]).assign(!val)
                !out.append(!attrib & | - | & !val)
            endif
        endif
    enddo
    -- Display the updated attribute list
    !this.atta.val = !out
endmethod

-----
-- End of Form definition and methods --
-----

-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----

```


Appendix B5 - Example !!traExampleVolumeView

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File:          traExampleVolumeView.pmlfrm
--
-- Description:
--   Form to demonstrate an VOLUME view gadget
--
--   Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!traExampleVolumeView

-- Set the form title
!this.formTitle = |Volume View|

-- Set the callbacks on the form
!this.initcall = |!this.init()|
!this.firstShownCall = |!this.firstShown()|
!this.killingCall = |!this.close()|

-- Define a context menu for the volume view gadget
!this.newMenu(|pop1|)
!this.pop1.add('CALLBACK', 'Limits CE', '!this.limitsCE()')
-- Set the standard icon size
!pix = |pixmap wid 16 hei 16|

-- Define a prompt paragraph gadget. This will provide feedback to the user
para .prompt text |Navigate : | width 46 lines 1

-- Define the four buttons down the L.H.S of the form
button .but1 at xmin ymax call |!this.limitsCE()| $!pix
button .but2 at xmin ymax call |!this.walkDrawlist()| $!pix
button .but3 at xmin ymax call |!this.setDrawlist(1, FALSE)| $!pix
button .but4 at xmin ymax call |!this.setDrawlist(2, FALSE)| $!pix

-- Define the volume view element
view .volumeView at xmax.but1 ymax.prompt volume

-- Set the size and the initial viewing direction
width 45 height 15
limits auto
isometric 3

-- Set the required "INMODES". These are required if the user is to pick from the view
inmode create _navi type |DES_NAVIGATE| $* Navigate
inmode create _pick type |DES_PICK| $* Standard Element Pick
inmode create _pAny type |DES_PICK_ANY| $* Pick Any (Element, Ppoint, Pline)
inmode create _pLine type |DES_PICK_PLINE| $* Plines
inmode create _point type |DES_PICK_POINT| $* Ppoints
inmode create _screen type |DES_3D_LINE| $* Screen Position
inmode create _pickxgeom type |DES_PICK_XGEOM| $* Laser Model rays
inmode create _pickDetail type |DES_PICK_DETAIL| $* Default (Navigation)
inmode create _default type |DES_NAVIGATE| $* Default (Navigation)

exit

-- Define the footer
line .line at xmin.but1 ymax+0.5 anchor b+l+r horizontal width 47.5
para .footer at xcen - 0.5 * size ymax anchor b text |AVEVA Training| width 11

-- A form member to remember the drawlist associated with the view
member .drawlist is REAL

exit

--(Continues on next page)
```

```

-----
-- Constructor Method - Setup the form when it is loaded
-----
define method .traExampleVolumeView()

    -- Apply the icons to the four buttons
    !this.but1.addPixmap(!pml.getPathName(|autoce16.png|))
    !this.but2.addPixmap(!pml.getPathName(|ng_zoomtodrawlist.png|))
    !this.but3.addPixmap(!pml.getPathName(|addce16.png|))
    !this.but4.addPixmap(!pml.getPathName(|removece16.png|))

    -- As icons are being used, set tooltips to help users
    !this.but1.setToolTip(|Limits CE|)
    !this.but2.setToolTip(|Walk to Drawlist|)
    !this.but3.setToolTip(|Add to Drawlist|)
    !this.but4.setToolTip(|Remove from Drawlist|)

    -- Apply some default settings to the view element
    !this.volumeView.borders = FALSE
    !this.volumeView.background = |darkslate|
    !this.volumeView.shaded = TRUE
    !this.volumeView.projection = |PARALLEL|
    !this.volumeView.radius = 100
    !this.volumeView.range = 500.0
    !this.volumeView.eyemode = FALSE
    !this.volumeView.step = 25
    !this.volumeView.setpopup(!this.pop1)

    -- Set Defaults for various INMODEs
    !this.volumeView.navi.cursor = 'Pointer'
    !this.volumeView.navi.prompt = 'Navigate : '
    !this.volumeView.pick.cursor = 'Pointer'
    !this.volumeView.pick.prompt = 'Pick Element : '
    !this.volumeView.pAny.cursor = 'Pointer'
    !this.volumeView.pAny.prompt = 'Pick Any : '
    !this.volumeView.pLine.cursor = 'Pointer'
    !this.volumeView.pLine.prompt = 'Pick Pline : '
    !this.volumeView.point.cursor = 'Pick'
    !this.volumeView.point.prompt = 'Point Ppoint : '
    !this.volumeView.screen.cursor = 'Crosshair'
    !this.volumeView.screen.prompt = 'Pick 3D Position : '
    !this.volumeView.default.cursor = 'Pointer'
    !this.volumeView.default.prompt = 'Navigate : '

    -- Create local drawlist within the global object
    !this.drawlist = !!gphDrawlists.createDrawList()
endmethod

-----
-- First Shown Method - Register the view gadget and attach the drawlist
-----
define method .firstShown()
    -- Add 3D view to view system
    !!gphViews.add(!this.volumeView)
    -- Add local drawlist add to 3D view
    !!gphDrawlists.attachView(!this.drawlist, !this.volumeView)
endmethod

-----
-- Initialisation Method - Update the drawlist and active the view
-----
define method .init()
    -- Update the drawlist
    !this.setDrawlist(1, TRUE)
    -- Active the volume view
    !this.volumeView.active = TRUE
    -- Turn on holes drawn
    !!gphDrawlists.drawlists[!this.drawlist].holes(TRUE)
endmethod

-----
-- Killing Call Method - Detach the drawlist and then delete it
-----
define method .close()
    !!gphDrawlists.detachView(!this.volumeView)
    !!gphDrawlists.deleteDrawlist(!this.drawlist)
endmethod

-- (Continues on next page)

```

```

-----
-- Method .walkDrawlist() - Set the view limits based on the drawlist --
-----
define method .walkDrawlist()
  -- Get the drawlist associated with the view
  !drawlist = !!gphDrawlists.drawlist(!this.drawlist)
  -- Derive a volume object from the members of the drawlist
  !volume = object volume(!drawlist.members())
  -- Derive a limits array in the expected format
  !limits[1] = !volume.from.east
  !limits[2] = !volume.to.east
  !limits[3] = !volume.from.north
  !limits[4] = !volume.to.north
  !limits[5] = !volume.from.up
  !limits[6] = !volume.to.up
  -- Apply the limits to the volume view
  !this.volumeView.limits = !limits
  handle ANY
    -- Incase the drawlist is empty or has no volume
  endhandle
endmethod

-----
-- Method .limitsCE() - Set the view limits based on the current element --
-----
define method .limitsCE()
  -- Set the Views limits based on the chosen element
  !!gphViews.limits(!this.volumeView, !!CE)
endmethod

-----
-- Method .setDrawlist(REAL, BOOLEAN) - Update the drawlist --
-----
define method .setDrawlist(!flag is REAL, !reset is BOOLEAN)
  -- Get the drawlist associated with the view
  !drawlist = !!gphDrawlists.drawlist(!this.drawlist)
  -- Clear the drawlist is a reset is requested
  if !reset then
    !drawlist.removeall()
  endif
  -- Add/Remove CE
  if !flag.eq(1) then
    !drawlist.add(!!CE)
  elseif !flag.eq(2) then
    !drawlist.remove(!!CE)
  endif
  -- Update the limits of the view based on the drawlist
  !this.walkDrawlist()
endmethod

-----
-- End of Form definition and methods --
-----
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----

```

Appendix B6 - Example c2ex5.pmlfrm

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex5.pmlfrm
--
-- Description:
-- Example form solution to Exercise 5
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex5 dialog docking

-- Set the form title and callbacks
!this.formTitle = |Equipment Checker|
!this.initcall = |!this.init()|
!this.firstShownCall = |!this.firstShown()|
!this.killingCall = |!this.close()|
!this.quitcall = |!this.clear()|

-- Create a MENU object for use as a context menu on the list gadget
!this.newMenu(|popupMenu|)
!this.popupMenu.add('CALLBACK', 'Go to Nozzle', '!!CE = !this.nozzleList.selection().dbref()')
-- Define the prompt and view gadgets
para .prompt text |Navigate : | width 46 lines 1
frame .view panel at x 0.5 ymax.prompt anchor all wid 1 hei 1
view .volumeView at x 0.5 ymax.prompt VOLUME
width 40 height 28
border off
shading on
isometric 3
exit
exit

-- Define the gadgets at the top of the form
button .update linklabel |Update| at xmax ymin.prompt anchor t+r call |!this.init()| width 5
para .title at xmax ymin anchor t+r text |Available Equipments below | width 34
line .line at xmin.update ymax anchor t+r horizontal width 42

!buttPos = |xmax - size ymax anchor b+r|
!anch1 = |anchor t+l+r|
!anch2 = |anchor b+l+r|
-- Define the gadgets to choose the equipments from
combo .equip at xmin.update ymax+0.2 |Select an Equipment| tagwidth 15 anchor t+r width 24
list .nozzleList at xmin.update ymax+0.2 anchor t+r width 40 length 5
button .refresh linklabel |Refresh Checks| at $!buttPos anchor t+r width 9.6

-- Define the gadgets to allow the attributes to be altered
!tag = |Equipment Attributes|
textpane .atta |$!tag| at xmin.nozzleList ymax-0.5 anchor t+r width 42 height 3.5
button .updateAtta linklabel |Update Attributes| at $!buttPos anchor t+r width 10.3
para .taskTitle at xmin.update ymax.atta + 0.5 anchor t+r text |Available Tasks|
line .taskLine at xmin.update ymax.taskTitle anchor t+r horiz width 42

!anch = |anchor t+r|
para .clipPix at xmin + 1 ymax.taskLine + 0.2 anchor t+r pixmap wid 16 hei 16
button .clipBut linklabel at xmax + 1 ymin |Add Connected and Enable Clipbox...| $!anch
para .tagNozzPix at xmin.clipPix ymax + 0.2 anchor t+r pixmap wid 16 hei 16
button .tagNozzBut linklabel at xmax + 1 ymin |Tag Selected Nozzle| $!anch
para .tagNozzlesPix at xmin.clipPix ymax + 0.2 anchor t+r pixmap wid 16 hei 16
button .tagNozzlesBut linklabel at xmax + 1 ymin |Tag All Nozzles| $!anch
para .hlNozzPix at xmin.clipPix ymax + 0.2 anchor t+r pixmap wid 16 hei 16
button .hlNozzBut linklabel at xmax + 1 ymin |Highlight Nozzles| $!anch

frame .limitslide foldup |Edit Clip Volume| at xmin.atta ymax.hlNozzPix + 0.5 $!anch width 41
slider .slide anchor l+r range 0 +2000 step 100 val 0 width 40 height 2.5
text .minslide at xmin.limitslide+0.2 ymin.limitslide+2.5 width 5 is REAL
text .maxslide at xmax.limitslide-size ymin.limitslide+2.5 width 5 is REAL
toggle .incl |Update view limits?| at xmax.minslide + 5 ymin.limitslide+2.5
exit

!fPos = |xmin.atta ymax.limitslide+0.1|
frame .colour foldup |Edit Highlight Colour| at $!fPos $!anch width .limitslide
```

```

para .para1      at xmin.colour + 0.5 ymin.colour + 1.2 text |Current equipment |
button .butt1 |  | at xmax.para1 - 3 ymin.para1 pixmap wid 25 hei 15
para .para2      at xmin.colour + 0.5 ymax.butt1      text |Connected nozzles |
button .butt2 |  | at xmax.para2 - 3 ymax.butt1 pixmap wid 25 hei 15
para .para3      at xmin.colour + 0.5 ymax.butt2      text |Unconnected nozzles|
button .butt3 |  | at xmax.para3 - 3 ymax.butt2 pixmap wid 25 hei 15
para .para4      at xmin.colour + 0.5 ymax.butt3      text |Check nozzles      |
button .butt4 |  | at xmax.para3 - 3 ymax.butt3 pixmap wid 25 hei 15

frame .colourpick panel at xmax.butt1 + 2.5 ymin.colour + 0.5 width 10
!val = 0
do !I from 1 to 8
  do !J from 0 to 4
    !val = !val + 1
    !no = !I & |000| & !J
    !x = (!I * 2) - 1
    !y = (!J * 0.6) + 0.75
    button .butt$!no |  | pixmap at x$!x ymin.colourpick + $!y backg $!val width 10 aspect 1
    !this.butt$!<no>.callback = !|this.colourpick($!val)|
  enddo
enddo
exit
member .clipbox is GPHCLIPBOX
member .drawlist is REAL
member .limits is ARRAY
member .col      is REAL
member .tagged is ARRAY
member .highlight is ARRAY
exit
-----
-- Constructor Method - Set the images and callbacks on the form --
-----
define method .c2ex5()
-- Set the titles and context menu on the list
!title = |Nozzles/Connected?/Attached?/Aligned?/Size?|
!this.nozzleList.setHeadings(!title.split(|/|))
!this.nozzleList.setpopup(!this.popupMenu)
-- Pre-fill-in some attributes to be edited
!info[1] = 'Description - Unset'
!info[2] = 'Function - Unset'
!info[3] = 'Purpose - Unset'
!this.atta.val = !info
-- Set the gadget callbacks
!this.equip.callback = !|this.setEquip(|
!this.nozzleList.callback = !|this.check(|
!this.refresh.callback = !|this.checkNozzles(|
!this.updateAtta.callback = !|this.updateAtt(2)|

!this.slide.callback = !|this.slide(|
!this.minslide.callback = !|this.updateRange(|
!this.maxslide.callback = !|this.updateRange(|

!this.butt1.callback = !|this.colourpick('A' )|
!this.butt2.callback = !|this.colourpick('B' )|
!this.butt3.callback = !|this.colourpick('C' )|
!this.butt4.callback = !|this.colourpick('D' )|

!this.clipBut      .callback = !|this.enableClip(|
!this.tagNozzBut   .callback = !|this.tag(!this.tagNozzBut)|
!this.tagNozzlesBut.callback = !|this.tag(!this.tagNozzlesBut)|
!this.hlNozzBut    .callback = !|this.highlight(|

!this.volumeView.borders = FALSE
!this.volumeView.background = |darkslate|
!this.volumeView.shaded = TRUE
!this.volumeView.projection = |PARALLEL|
!this.volumeView.radius = 100
!this.volumeView.range = 500.0
!this.volumeView.eyemode = FALSE
!this.volumeView.step = 25

!this.minslide.val = 0
!this.maxslide.val = 2000

!this.drawlist = !!gphDrawlists.createDrawList()
!this.clipPix.addPixmap(!|pml.getPathName('clipce16.png')|)
!this.tagNozzPix.AddPixmap(!|PML.GetPathName('draft_lab_01.png')|)

```

```

!this.tagNozzlesPix.AddPixmap(!PML.GetPathName('draft_lab_01.png'))
!this.hlNozzPix.AddPixmap(!PML.GetPathName('nozzle-16.png'))

!this.colourpick.visible = FALSE
!this.limitslide.visible = FALSE
!this.limitslide.expanded = FALSE
!this.colour.visible = FALSE
!this.colour.expanded = FALSE

!this.clipbox = object GPHCLIPBOX()
!this.clipbox.view = !this.volumeView
!this.clipbox.capOn()

!this.butt1.background = 1
!this.butt2.background = 20
!this.butt3.background = 18
!this.butt4.background = 23

endmethod
-----
-- First Shown Method - Collect all equi for level and display nozzles --
-----
define method .firstShown()
-- Add 3D view to view system
!!gphViews.add(!this.volumeView)
-- Add local drawlist add to 3D view
!!gphDrawlists.attachView(!this.drawlist, !this.volumeView)
endmethod
-----
-- Killing Call Method - Collect all equi for level and display nozzles --
-----
define method .close()
!!gphDrawlists.detachView(!this.volumeView)
!!gphDrawlists.deleteDrawlist(!this.drawlist)
endmethod
-----
-- Initialisation Method - Collect all equi for level and display nozzles --
-----
define method .init()
-- Activate the view for use
!this.volumeView.active = TRUE
-- If at SITE level, collect for the site otherwise do it for ZONE
if !!ce.type.eq(|SITE|) then
!level = |SITE|
else
!level = |ZONE|
endif
-- Collect the equipment using PML 1 style collection syntax
VAR !coll COLL ALL EQUIP FOR $!level
handle ANY
!!alert.error(|Make sure you are at an EQUI, ZONE or SITE element|)
elsehandle NONE
-- If no equipment are found, and we're at a ZONE try collecting for SITE
if !coll.size().eq(0) and !level.eq(|ZONE|) then
!!alert.message(|No equipment found in current ZONE, so whole SITE will be searched|)
VAR !coll COLL ALL EQUIP FOR SITE
!level = |SITE|
endif
-- Update the title paragraph gadget
!this.title.val = |Available Equipments below | & !level
-- Evaluate the fullnames of the equis and display the information
VAR !name EVAL FLNN FOR ALL FROM !coll
!this.equip.dtext = !name
!this.equip.rtext = !coll
-- Collect the nozzles
!this.collectNozzles()
endhandle
endmethod

-----
-- Method .collectNozzles() - For the selected EQUI, collect all the nozzles --
-----
define method .collectNozzles()
-- Get the selected EQUI element
!equipRef = !this.equip.selection().dbref()
if !equipRef.unset() then

```

```

-- If the element is unset, then clear the nozzle list
!this.nozzleList.clear()
else
-- Perform a PML2 style collection for the nozzles
!nozzColl = object COLLECTION()
!nozzColl.type('NOZZ')
!nozzColl.scope(!equipRef)
!results = !nozzColl.results()
!this.nozzleList.dtext = !results.evaluate(object block ('!results[!evalIndex].flnn'))
!this.nozzleList.rtext = !results.evaluate(object block ('!results[!evalIndex].string()'))
-- Check the nozzles and update the results
!this.checkNozzles()
!this.clear()
!this.updateAtt(1)
!this.setDrawlist()
!this.enableClip()
!this.highlight()
!this.tagged.clear()
do !n index !this.nozzleList.dtext
!this.tagged[!n] = FALSE
enddo
endif
endmethod

-----
-- Method .setEquip(GADGET, STRING) - Open callback on combo gadget to check user entry --
-----
define method .setEquip(!gad is gadget, !event is STRING)
-- If the user has typed into the gadget
if !event.eq('VALIDATE') then
!userInput = !this.equip.displayText()
-- Loop through available equipments and choice the nearest match
do !n index !this.equip.dtext
!chrs = !userInput.length()
!test = !this.equip.dtext[!n].upcase().substring(1, !chrs)
if !userInput.upcase().eq(!test) then
!this.equip.val = !n
break
endif
enddo
endif
-- Recollect the nozzles on the newly selected equipment
!this.collectNozzles()
endmethod

-----
-- Method .checkNozzles() - Check the collected nozzles and display the results --
-----
define method .checkNozzles()
if !this.nozzleList.rtext.set() then
-- Loop through the collected nozzles
do !n index !this.nozzleList.rtext
!nozz = !this.nozzleList.rtext[!n].dbref()
-- set the default result
do !m from 2 to 4
!result[!m] = |N/A|
enddo
-- Check 1: is the connection valid
if !nozz.cref.unset().or(!nozz.cref.badref()) then
!result[1] = |Check|
else
!result[1] = |OK|
!end = |t|
if !nozz.cref.href.eq(!nozz) then
!end = |h|
endif
-- Check 2: is the nozzle and the same position as the attached
if !nozz.pos.wrt( /* ).eq(!nozz.cref.attribute(!end & |pos|).wrt( /* )) then
!result[2] = |OK|
endif

-- Check 3: is the nozzle and the same direction as the attached
if !nozz.pdir[1].wrt( /* ).eq(!nozz.cref.attribute(!end & |dir|).wrt( /* )) then
!result[3] = |OK|
endif
-- Check 4: is the nozzle and the same size as the attached
if !nozz.cpar[1].eq(!nozz.cref.attribute(!end & |bore|).real()) then
!result[4] = |OK|
endif
endif
endif

```

```

-- Record the check results for this nozzle
!nozzleInfo[!n][1] = !nozz.flnn
!nozzleInfo[!n].appendArray(!result)
enddo
!this.highlight = !nozzleInfo
-- Apply the results back to the list gadget
!rtext = !this.nozzleList.rtext
!this.nozzleList.setRows(!nozzleInfo)
!this.nozzleList.rtext = !rtext
endif
endmethod

-----
-- Method .updateAtt(REAL) - Check the collected nozzles and display the results --
-----

define method .updateAtt(!flag is REAL)
-- Get the selected piece of equipment and the available attributes
!equip = !this.equip.selection().dbref()
!availAtts = !equip.attributes()
-- Update the textpane title
!this.atta.tag = !equip.flnn & ' Attributes'
!rows = !this.atta.val
-- Loop through the rows of the textpane to extract and use the information
!out = ARRAY()
do !n index !rows
-- Split the line on a "-"
!split = !rows[!n].split('-')
!attrib = !split[1].trim()
-- Evaluate the available attributes so they are the same string length as the entered
!block = object block ('!availAtts[!evalIndex].upcase().substring(1, !attrib.length())')
!avail = !availAtts.evaluate(!block)
-- If the attribute can be found, then use it
if !avail.findfirst(!attrib.upcase()).set() then
if !flag.eq(1) then
-- If the flag = 1 then update the textpane with the current attribute value
!value = !equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())])
!out.append(!attrib & | - | & !value)
elseif !flag.eq(2) then
-- If the flag = 2 then assign the value to the attribute
!val = !split[2].trim()
!equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())]).assign(!val)
!out.append(!attrib & | - | & !val)
endif
endif
enddo
-- Display the updated attribute list
!this.atta.val = !out
Endmethod

-----
-- Method .setDrawlist(REAL) - Add the select equipment to the drawlist --
-----

define method .setDrawlist()
-- Get the selected equipment and drawlist associated with the view
!equip = !this.equip.selection().dbref()
!drawlist = !!gphDrawlists.drawlist(!this.drawlist)
!drawlist.holes(TRUE)
-- Clear the draw, add the equi and set the limits
!drawlist.removeall()
!drawlist.add(!equip)
!!gphViews.limits(!this.volumeView, !equip)
Endmethod

-----
-- Method .setDrawlist(REAL) - Add the select equipment to the drawlist --
-----

define method .colourpick(!colour is ANY)
-- Based on the colour button pressed, set the colour
!col = !colour.string()

if !col.eq('A') then
!this.col = 1
!this.colourpick.visible = TRUE
elseif !col.eq('B') then
!this.col = 2

```



```

!this.colourpick.visible = TRUE
elseif !col.eq('C') then
!this.col = 3
!this.colourpick.visible = TRUE
elseif !col.eq('D') then
!this.col = 4
!this.colourpick.visible = TRUE
else
if !this.col.eq(1) then
!this.butt1.background = !col.real()
elseif !this.col.eq(2) then
!this.butt2.background = !col.real()
elseif !this.col.eq(3) then
!this.butt3.background = !col.real()
elseif !this.col.eq(4) then
!this.butt4.background = !col.real()
endif
!this.colourpick.visible = FALSE
-- Rehighlight with the new colours
!this.highlight()
endif
endmethod
-----
-- Method .highlight() - Highlight the equipment and nozzles --
-----
define method .highlight()
-- Get the selected equipment
!equip = !this.equip.selection().dbref()
-- If highlighting is required
if !this.hlNozzBut.val then
-- Highlight the equipment
!!gphDrawlists.drawlists[!this.drawlist].highlight(!equip,!this.butt1.background)
-- Update the icons
!this.hlNozzPix.AddPixmap(!PML.GetPathName('nozzle-16.png'))
!this.hlNozzBut.tag = |Unhighlight Nozzles|
-- Show then hidden frame
!this.colour.visible = TRUE
-- Loop through any displayed nozzles
if !this.nozzleList.rtext.set() then
-- Highlight the nozzles based on the check results
!nozz = !this.highlight
do !I index !nozz
!nozzle = !this.nozzleList.rtext[!I].dbref()
if !nozz[!I][2].eq(|OK|) then
!col = !this.butt2.background
elseif !nozz[!I][2].eq(|Check|) then
!col = !this.butt3.background
endif
if !nozz[!I][3].eq(|Check|) or !nozz[!I][4].eq(|Check|) or !nozz[!I][5].eq(|Check|) then
!col = !this.butt4.background
endif
-- Highlight the nozzle
!!gphDrawlists.drawlists[!this.drawlist].highlight(!nozzle,!col)
enddo
endif
else
-- Reset the icons
!this.hlNozzPix.AddPixmap(!PML.GetPathName('nozzle-16.png'))
!this.hlNozzBut.tag = |Highlight Nozzles|
-- Hide the frame
!this.colour.visible = FALSE
!this.colour.expanded = FALSE
-- Unhighlight everything
!!gphDrawlists.drawlists[!this.drawlist].unhighlight(!equip)
if !this.nozzleList.rtext.set() then
do !i index !this.nozzleList.rtext
!!gphDrawlists.drawlists[!this.drawlist].unhighlight(!this.nozzleList.rtext[!i].dbref())
enddo
endif
endif
endmethod
-----
-- Method .slide() - Increase the size of the clip box based on the slider --
-----
define method .slide()
-- Get the value of the slider
!value = !this.slide.val

```

```
-- Update the limits if required
if !this.incl.val then
    !limits = !this.limits
    !modlimit[1] = !limits[1] - !value
    !modlimit[2] = !limits[2] + !value
    !modlimit[3] = !limits[3] - !value
    !modlimit[4] = !limits[4] + !value
    !modlimit[5] = !limits[5] - !value
    !modlimit[6] = !limits[6] + !value
    !this.volumeView.limits = !modlimit
endif
-- Update the clip box and refresh the view
!this.volumeView.clipBoxXlen = !this.clipbox.box.xlength + !value
!this.volumeView.clipBoxYlen = !this.clipbox.box.ylength + !value
!this.volumeView.clipBoxZlen = !this.clipbox.box.zlength + !value
!this.volumeView.refresh()
endmethod

-----
-- Method .enableClip() - Apply the clipbox to the view --
-----

define method .enableClip()
    -- Get the drawlist and find the connected elements
    !drawlist = !gphDrawlists.drawlist(!this.drawlist)
    !connectNozz = ARRAY()
    do !n index !this.nozzleList.rtext
        !nozz = !this.nozzleList.rtext[!n]
        if !nozz.dbref().cref.unset().not().and(!nozz.dbref().cref.badref().not()) then
            !connectNozz.append(!nozz.dbref().cref)
        endif
    enddo
    -- Get the volume of the equipment. Set the clipbox to the size of the volume
    !volume = object volume(!this.equip.selection().dbref())
    !this.clipbox.box = !volume.box()
    -- If the clip is turned on
    if !this.clipBut.val then
        -- Update the icons
        !this.clipBut.tag = |Remove Connected and Disable Clipbox...|
        !this.clipPix.addPixmap(!pml.getPathName(|nozzle-16.png|))
        !this.limitslide.visible = TRUE
        -- Loop through connected and add them to the drawlist
        do !n index !connectNozz
            !drawlist.add(!connectNozz[!n])
        enddo
        -- Update the clip box and apply it
        !this.limits = !this.volumeView.limits
        !this.clipbox.active = FALSE
        !this.clipbox.set()
        !this.clipbox.active = TRUE
        !this.volumeView.clipping = TRUE
        !this.slide()
    else
        -- Update the icons
        !this.clipBut.tag = |Add Connected and Enable Clipbox...|
        !this.clipPix.addPixmap(!pml.getPathName(|nozzle-16.png|))
        !this.limitslide.visible = FALSE
        !this.limitslide.expanded = FALSE
        -- Remove the connected elements from the drawlist
        do !n index !connectNozz
            !drawlist.remove(!connectNozz[!n])
        enddo
        !this.clipbox.active = FALSE
        !this.volumeView.clipping = FALSE
    endif
endmethod

-----
-- Method .tag(GADGET) - Apply the clipbox to the view --
-----

define method .tag(!gad is GADGET)
    -- If the gadget is a single tag - only tag the selected
    !check1 = !gad.tag.upcase().eq(|TAG SELECTED NOZZLE|)
    !check2 = !gad.tag.upcase().eq(|UNTAG SELECTED NOZZLE|)
    if !check1.or(!check2) then
        !start = !this.nozzleList.val
        !finish = !this.nozzleList.val
    else
        !start = 1
        !finish = !this.nozzleList.dtext.size()
    endif
endmethod
```

```

endif
-- If tagging, loop through the nozzles and tag
if !gad.val then
  do !n from !start to !finish
    if !this.tagged[!n].eq(FALSE) then
      !nozzname = !this.nozzleList.rtext[!n].dbref().flnn
      !nozzpos = !this.nozzleList.rtext[!n].dbref().pos
      !num = 1500 + !n
      AID TEXT NUMBER $!num '$!nozzname' AT $!nozzpos
      !this.tagged[!n] = TRUE
    endif
  enddo
else
  -- else, loop through and untag
  do !n from !start to !finish
    !num = 1500 + !n
    AID CLEAR text $!num
    handle ANY
    endhandle
    !this.tagged[!n] = FALSE
  enddo
endif
-- Check if all nozzles have been tagged, if so - update the icons
!testTagged = !this.tagged
!testTagged.unique()
if !testtagged.size().eq(1).and(!testtagged[1].eq(TRUE)) then
  !this.tagNozzlesBut.val = TRUE
  !this.tagNozzlesBut.tag = |Untag All Nozzles|
  !this.tagNozzlesPix.addPixmap(!pml.getPathName(|draft_lab_06.png|))
else
  !this.tagNozzlesBut.val = FALSE
  !this.tagNozzlesBut.tag = |Tag All Nozzles|
  !this.tagNozzlesPix.addPixmap(!pml.getPathName(|draft_lab_01.png|))
endif
-- Re-check
!this.check()
endmethod

-----
-- Method .check() - Apply the clipboard to the view
-----
define method .check()
  -- If the selected is tagged, update the icons and tag
  !check = !this.tagged[!this.nozzleList.val]
  if !check then
    !this.tagNozzBut.val = TRUE
    !this.tagNozzBut.tag = |Untag Selected Nozzle|
    !this.tagNozzPix.addPixmap(!pml.getPathName(|draft_lab_06.png|))
  else
    !this.tagNozzBut.val = FALSE
    !this.tagNozzBut.tag = |Tag Selected Nozzle|
    !this.tagNozzPix.addPixmap(!pml.getPathName(|draft_lab_01.png|))
  endif
endmethod

-----
-- Method .updateRange() - Update the slider range based on user entered values
-----
define method .updateRange()
  !range[1] = !this.minslide.val
  !range[2] = !this.maxslide.val
  !range[3] = 100
  !this.slide.range = !range
  !this.slide.refresh()
endmethod

-----
-- Method .clear() - Tidy up and reset the form
-----
define method .clear()
  do !n index !this.nozzleList.rtext
    !num = 1500 + !n
    AID CLEAR text $!num
    handle ANY
    endhandle
  enddo
  !this.tagNozzBut.val = FALSE
  !this.tagNozzBut.tag = |Tag Selected Nozzle|
  !this.tagNozzPix.addPixmap(!pml.getPathName(|draft_lab_06.png|))
  !this.tagNozzlesBut.val = FALSE

```

```
!this.tagNozzlesBut.tag = |Tag All Nozzles|  
!this.tagNozzlesPix.addPixmap(!pml.getPathName(|draft_lab_01.png|))  
endmethod
```

```
-----  
-- End of Form definition and methods                                     --  
-----  
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited          --  
-----
```

Copia para EE AA

Appendix B7 - Example c2ex6.pmlfrm

```

-----
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: c2ex6.pmlfrm
--
-- Description:
-- Example form solution to Exercise 6
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex6 dialog resizable

-- Set the form title and initialisation call
!this.formTitle = |Equipment Checker|
!this.initcall = !!this.init()|
!this.quitcall = !!this.tidy()|

-- Create a MENU object for use as a context menu on the list gadget
!this.newMenu(|popupMenu|)
!popupCall = '!!CE = !this.nozzleList.selection().dbref()'
!this.popupMenu.add('CALLBACK', 'Go to Nozzle', !popupCall)

-- Define the gadgets at the top of the form
button .update linklabel |Update| call !!this.init()| width 5
para .title at xmax ymin anchor t+l+r text |Available Equipments below | width 34
line .line at xmin.update ymax anchor t+l+r horizontal width 42

!buttPos = |xmax - size ymax anchor b+r|
!anch1 = |anchor t+l+r |
!anch2 = |anchor b+l+r|

-- Define the gadgets to choose the equipments from
button .pick at xmin.update ymax+0.2 pixmap width 33 height 32
combo .equip at xmax+0.2 ymin |Select an Equipment| tagwidth 14 $!anch1 width 22
list .nozzleList at xmin.update ymax+0.5 anchor all width 40 length 5
button .refresh linklabel |Refresh Checks| at $!buttPos width 9.6

-- Define the gadgets to allow the attributes to be altered
!tag = |Equipment Attributes|
textpane .atta |$!tag| at xmin.nozzleList ymax-0.5 $!anch2 width 42 height 3.5
button .updateAtta linklabel |Update Attributes| at $!buttPos width 10.3

exit

-----
-- Constructor Method - Set the images and callbacks on the form
-----

define method .c2ex6()
-- Set the titles and context menu on the list
!title = |Nozzles/Connected?/Attached?/Aligned?/Size?|
!this.nozzleList.setHeadings(!title.split(|/|))
!this.nozzleList.setpopup(!this.popupMenu)
-- Pre-fill-in some attributes to be edited
!info[1] = 'Description - Unset'
!info[2] = 'Function - Unset'
!info[3] = 'Purpose - Unset'
!this.atta.val = !info
-- Set the gadget callbacks
!this.pick.callback = !!this.pick()|
!this.equip.callback = !!this.setEquip()|
!this.refresh.callback = !!this.checkNozzles()|
!this.updateAtta.callback = !!this.updateAtt(2)|
-- Set the icon and tooltip of the pixmap button
!this.pick.addPixmap(!pml.getPathName(|onepick.png|))
!this.pick.setToolTip(|Identify Equipment from 3D View|)
endmethod

--(Continues on next page)

```

```

-----
-- Initialisation Method - Collect all equi the current element and display nozzles --
-----

define method .init()
    !this.collectEquipment(!!ce)
endmethod

-----
-- Quit Call Method - Tidy any activate pick events --
-----

define method .tidy()
    -- Clear the any existing pick
    !!edgCtrl.remove(|Identify Equipment|)
endmethod

-----
-- Method .collectEquipment(DBREF) - Collect all equi for level and display nozzles --
-----

define method .collectEquipment(!item is DBREF)
    -- Loop up to find a SITE or ZONE
    !allowableTypes = |ZONE SITE WORL|
    do
        break if !allowableTypes.split().findFirst(!item.type).set()
        !item = !item.owner
    enddo
    -- If we have found a SITE or ZONE, continue
    if !allowableTypes.split().findFirst(!item.type).neq(3) then
        -- Collect the equipment using PML 1 style collection syntax
        VAR !coll COLL ALL EQUIP FOR $!item
        -- If no equipment are found, offer the chance to collect for the owner
        if !coll.unset() then
            !message = |No equipment found for | & !item.type & |, |
            !message = !message & |do you wish to search the | & !item.owner.type & |?|
            if !!alert.confirm(!message).eq(|YES|) then
                !this.collectEquipment(!item.owner)
                return
            endif
        endif
        -- Update the title paragraph gadget
        !this.title.val = |Available Equipments below | & !item.type & | | & !item.flnn
        -- Evaluate the fullnames of the equis and display the information
        VAR !name EVAL FLNN FOR ALL FROM !coll
        !this.equip.dtext = !name
        !this.equip.rtext = !coll
        -- Collect the nozzles
        !this.collectNozzles()
    endif
endmethod

-----
-- Method .collectNozzles() - For the selected EQUI, collect all the nozzles --
-----

define method .collectNozzles()
    -- Get the selected EQUI element
    !equipRef = !this.equip.selection().dbref()
    if !equipRef.unset() then
        -- If the element is unset, then clear the nozzle list
        !this.nozzleList.clear()
    else
        -- Perform a PML2 style collection for the nozzles
        !nozzColl = object COLLECTION()
        !nozzColl.type('NOZZ')
        !nozzColl.scope(!equipRef)
        !results = !nozzColl.results()
        !this.nozzleList.dtext = !results.evaluate(object block ('!results[!evalIndex].flnn'))
        !this.nozzleList.rtext = !results.evaluate(object block ('!results[!evalIndex].string()'))
        -- Check the nozzles and update the results
        !this.checkNozzles()
        !this.updateAtt(1)
    endif
endmethod

-----
-- Method .setEquip(GADGET, STRING) - Open callback on combo gadget to check user entry --
-----

define method .setEquip(!gad is gadget, !event is STRING)
    -- If the user has typed into the gadget
    if !event.eq('VALIDATE') then
        !userInput = !this.equip.displayText()

        --(Continues on next page)
    
```

```
-- Loop through available equipments and choice the nearest match
do !n index !this.equip.dtext
!chrs = !userInput.length()
!test = !this.equip.dtext[!n].upcase().substring(1, !chrs)
if !userInput.upcase().eq(!test) then
!this.equip.val = !n
break
endif
enddo
endif
-- Recollect the nozzles on the newly selected equipment
!this.collectNozzles()
endmethod

-----
-- Method .checkNozzles() - Check the collected nozzles and display the results --
-----
define method .checkNozzles()
if !this.nozzleList.rtext.set() then
-- Loop through the collected nozzles
do !n index !this.nozzleList.rtext
!nozz = !this.nozzleList.rtext[!n].dbref()
-- set the default result
do !m from 2 to 4
!result[!m] = |N/A|
enddo
-- Check 1: is the connection valid
if !nozz.cref.unset().or(!nozz.cref.badref()) then
!result[1] = |Check|
else
!result[1] = |OK|
!end = |t|
if !nozz.cref.href.eq(!nozz) then
!end = |h|
endif
-- Check 2: is the nozzle and the same position as the attached
if !nozz.pos.wrt( /* ).eq(!nozz.cref.attribute(!end & |pos|).wrt( /* )) then
!result[2] = |OK|
endif
-- Check 3: is the nozzle and the same direction as the attached
if !nozz.pdir[1].wrt( /* ).eq(!nozz.cref.attribute(!end & |dir|).wrt( /* )) then
!result[3] = |OK|
endif
-- Check 4: is the nozzle and the same size as the attached
if !nozz.cpar[1].eq(!nozz.cref.attribute(!end & |bore|.real()) then
!result[4] = |OK|
endif
endif
-- Record the check results for this nozzle
!nozzleInfo[!n][1] = !nozz.flmn
!nozzleInfo[!n].appendArray(!result)
enddo
-- Apply the results back to the list gadget
!rtext = !this.nozzleList.rtext
!this.nozzleList.setRows(!nozzleInfo)
!this.nozzleList.rtext = !rtext
endif
endmethod

-----
-- Method .updateAtt(REAL) - Check the collected nozzles and display the results --
-----
define method .updateAtt(!flag is REAL)
-- Get the selected piece of equipment and the available attributes
!equip = !this.equip.selection().dbref()
!availAtts = !equip.attributes()
-- Update the textpane title
!this.atta.tag = !equip.flmn & ' Attributes'
!rows = !this.atta.val
-- Loop through the rows of the textpane to extract and use the information
!out = ARRAY()
do !n index !rows
-- Split the line on a "-"
!split = !rows[!n].split('-')
!attrib = !split[1].trim()
-- Evaluate the available attributes so they are the same string length as the entered
!block = object block ('!availAtts[!evalIndex].upcase().substring(1, !attrib.length())')
!avail = !availAtts.evaluate(!block)
-- (Continues on next page)
```

```
-- If the attribute can be found, then use it
if !avail.findfirst(!attrib.upcase()).set() then
  if !flag.eq(1) then
    -- If the flag = 1 then update the textpane with the current attribute value
    !value = !equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())])
    !out.append(!attrib & | - | & !value)
  elseif !flag.eq(2) then
    -- If the flag = 2 then assign the value to the attribute
    !val = !split[2].trim()
    !equip.attribute(!availAtts[!avail.findfirst(!attrib.upcase())]).assign(!val)
    !out.append(!attrib & | - | & !val)
  endif
endif
enddo
-- Display the updated attribute list
!this.atta.val = !out
endmethod

-----
-- Method .pick() - Setup and activate the pick event
-----

define method .pick()
  -- Clear the any existing pick
  !!edgCntrl.remove(|Identify Equipment|)
  -- Declare the EDGPACKET object
  !packet = object EDGPACKET()
  !packet.elementPick(|Identify Equipment or Nozzle <Esc> to finish|)
  !packet.description = |Identify Equipment|
  !packet.action = ||c2ex6.pickProcess(!this.return[1].item)|
  -- Add the packet to the system and activate the pick
  !!edgCntrl.add(!packet)
endmethod

-----
-- Method .pickProcess(DBREF) - Process the identified equipment
-----

define method .pickProcess(!item is DBREF)
  !allowableTypes = |EQUI NOZZ WORL|
  -- Loop up to find an EQUI/NOZZ (or the world)
  do
    break if !allowableTypes.split().findFirst(!item.type).set()
    !item = !item.owner
  enddo
  -- If we have found an EQUI, proceed...
  if !item.type.eq(|EQUI|).or(!item.type.eq(|NOZZ|)) then
    -- Find the item in the list
    if !item.type.eq(|EQUI|) then
      !val = !this.equip.rtext.findFirst(!item.string())
    else
      !expression = object EXPRESSION(|REFNO OF EQUI|)
      !equi = !expression.evaluate(!item)
      !val = !this.equip.rtext.findFirst(!equi.string())
    endif
    if !val.set() then
      -- If the equipment can be found in the list, pick it
      !this.equip.val = !val
      !this.collectNozzles()
      -- If a nozzle has been picked, ensure it is selected
      if !item.type.eq(|NOZZ|) then
        !val = !this.nozzleList.rtext.findFirst(!item.string())
        !this.nozzleList.val = !val
      endif
    else
      -- If not, offer the chance to recollect
      !message = |Identified equipment not in list. Do you wish to re-collect?|
      if !!alert.confirm(!message).eq(|YES|) then
        -- Recollect for the owner of the identified equipment
        !this.collectEquipment(!item.owner)
      endif
    endif
  endif
endmethod

-----
-- End of Form definition and methods
-----

-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
-----
```


Appendix B8 - Example traExampleExplorer.pmlfrm

```

-----
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File: traExampleExplorer.pmlfrm
--
-- Description:
-- Form to demonstrate the use of a NETEXPLORER PML.NET Control inside a CONTAINER
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!traExampleExplorer dialog docking

-- Load the required .dll for the explorer .NET gadget
import 'ExplorerAddin'
handle ANY
-- Handle all errors, incase its already been loaded
endhandle
using namespace 'Aveva.Core.Presentation'

-- Set the form title
!this.formTitle = 'Example .NET Explorer'

-- Create a popup menu object
!this.newMenu(|explorerPopup|)
!this.explorerPopup.add( 'CALLBACK', 'popup', '!this.popup()' )

para .pmlTitle at x 0.2 ymax text |Standard PML gadgets|
line .pmlLine at xmin - 0.1 ymax-0.5 anchor l+t+ horizontal wid 35 hei 0.7

-- Define the functionality at the top of the form
toggle .track |Track CE| at xmin+2 ymax anchor t+l
button .select linklabel |Find CE >>| at xmax ymin anchor t+r width 8
button .reset linklabel |Reset to CE >>| at xmin ymin anchor t+r width 10

para .netTitle at xmin.pmlTitle ymax text |.NET Container|
line .netLine at xmin.pmlLine ymax-0.5 anchor l+t+r horizontal wid 35 hei 0.7

-- Define the container that will hold the NETEXPLORER
!size = |wid 33 hei 15|
container .netFrame NOBOX PMLNETCONTROL '' at xmin+1 ymax anchor all $!size

line .line at xmin.netLine ymax+0.5 anchor b+l+r horizontal width 35
para .footer at xcen - 0.5 * size ymax anchor b text |AVEVA Training| width 11

-- Store the explorer control as a form member for future reference
member .explorer is PMLEXPLORERCONTROL
member .element is DBREF
exit
-- End of form definition
--

-----
-- Constructor Method - Setup the form when it is loaded
-----

define method .traExampleExplorer()

-- Apply the callbacks to the PML gadgets
!this.select.callback = !|this.explorer.selectElement(!|ce.name)|
!this.track.callback = !|this.explorer.setFollowCe(!this.track.val)|
!this.reset.callback = !|this.explorer.resetRoot(!|ce.name)|

-- Specify the namespace before using the .NET gadget
using namespace 'Aveva.Core.Presentation'

-- Create an instance of the control
!this.explorer = object PMLEXPLORERCONTROL()
!this.netFrame.control = !this.explorer.handle()
-- Register for the OnPopup event
!this.explorer.addeventhandler('OnPopup', !this, 'rightClickExplorer')
--(Continues on next page)

```

```
-- Use methods on Explorer C# control
!this.explorer.initialise('/*','')
endmethod
-- End of Constructor Method
--
-----
-- Method .rightClickExplorer(ARRAY) - Show context menu on explorer --
-----
define method .rightClickExplorer(!data is ARRAY)
!this.netFrame.popup = !this.explorerPopup
!this.netFrame.showPopup(!data[0], !data[1])
!this.element = !data[2].dbref()
endmethod
-- End of Method .rightClickExplorer(ARRAY)
--
-----
-- Method .popup() - Query the name of the selected element --
-----
define method .popup()
q var !this.element.name
endmethod
-- End of Method .popup()
--
-----
-- End of Form definition and methods --
-----
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----
```

Copia para EEAA

Appendix B9 - Example traExampleGridControl.pmlfrm

```
--
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited
--
-- File:          traExampleGridControl.pmlfrm
--
-- Description:
--   Form to demonstrate the use of a Grid Control PML.NET Control inside a CONTAINER
--
--   Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!traExampleGridControl dialog docking

-- Load the required .dll for the explorer .NET gadget
import 'GridControl'
handle ANY
  -- Handle all errors, incase its already been loaded
endhandle
using namespace 'Aveva.Core.Presentation'

-- Set the form title and the initialisation call
!this.formTitle = |Example .NET Grid Control|
!this.initcall = |!this.init()|

-- Create a popup menu object
!this.newMenu(|gridPopup|)
!this.gridPopup.add('CALLBACK', 'Add to 3D View', '!this.addToThreeDView()')
!this.gridPopup.add('CALLBACK', 'Save to Excel...', '!this.saveToExcel()')
!this.gridPopup.add('CALLBACK', 'Import from Excel...', '!this.loadFromExcel()')

-- Specify the container that will hold the grid control .NET gadget
container .gridFrame NOBOX PMLNETCONTROL 'grid' anchor all width 60 height 20

line .line at xmin ymax+0.5 anchor b+l+r horizontal width 60
para .footer at xcen - 0.5 * size ymax anchor b text |AVEVA Training| width 11

member .grid is NETGRIDCONTROL
member .elements is ARRAY

exit
-- End of form definition
--
-----
-- Constructor Method - Setup the form when it is loaded
-----

define method .traExampleGridControl()

  -- Specify the namespace before using the .NET gadget
  using namespace 'Aveva.Core.Presentation'

  -- Create an instance of the control
  !this.grid = object NetGridControl()
  !this.gridFrame.control = !this.grid.handle()

  -- Apply the event handlers
  !this.grid.addeventhandler('OnPopup', !this, 'rightClickGrid')
  !this.grid.addeventhandler('AfterSelectChange', !this, 'afterSelectChange')
  !this.grid.addeventhandler('BeforeCellUpdate', !this, 'beforeCellUpdate')
endmethod
-- End of Constructor Method
--
-----
-- Initialisation Method - Setup the Grid Control on the form
-----

define method .init()

  -- Specify the namespace before using the .NET gadget
  using namespace 'Aveva.Core.Presentation'
  -- Create headings
  !headings = |Name Type Owner Description|
  --(Continues on next page)
```

```
-- Create model items for population of grid
var !data collect all EQUIP
var !data append collect all PIPES

-- Bind data to grid
!nds = object NETDATASOURCE('Grid Control Example', !headings.split(), !data)
!this.grid.bindToDataSource(!nds)

-- Set grid parameters
!this.grid.columnExcelFilter(TRUE)
!this.grid.setNameColumnImage()
!this.grid.outlookGroupStyle(TRUE)
!this.grid.fixedHeaders(FALSE)
!this.grid.fixedRows(FALSE)
!this.grid.columnSummaries(TRUE)
!this.grid.autoFitColumns()
--!this.grid.editableGrid(TRUE)
--!this.grid.setEditableColumn(4,TRUE)
--!this.grid.setColumnColor(4, 'white')
endmethod
-- End of Initialisation Method
--
-----
-- Method .rightClickGrid(ARRAY) - Show the context menu on the control --
-----
define method .rightClickGrid(!data is ARRAY)
!this.gridFrame.popup = !this.gridPopup
!this.gridFrame.showPopup(!data[0], !data[1])
!this.elements = !data[2]
endmethod
-- End of Initialisation Method
--
-----
-- Method .addToThreeDView() - Adds the selected elements to the drawlist --
-----
define method .addToThreeDView()
do !element values !this.elements
add $!element
enddo
endmethod
-- End of Method .addToThreeDView()
--
-----
-- Method .saveToExcel() - Save the grid to excel via the file browser --
-----
define method .saveToExcel()

-- Ensure the correct .dll is loaded
import 'PMLFileBrowser'
handle ANY
endhandle

-- Specify the namespace before using the .NET gadget
using namespace 'Aveva Core.Presentation'

-- Show the browser form and based on the file, save the grid
!browser = object PMLFILEBROWSER('SAVE')
!browser.show('C:\', 'Example.xls', 'Save to Excel', false, 'Excel Files|*.xls|,1)
!this.grid.saveGridToExcel(!browser.file())
endmethod
-- End of Method .saveToExcel()
--
-----
-- Method .loadFromExcel() - Load the grid from excel via the file browser --
-----
define method .loadFromExcel()

-- Ensure the correct .dll is loaded
import 'PMLFileBrowser'
handle ANY
endhandle

-- Specify the namespace before using the .NET gadget
using namespace 'Aveva.Core.Presentation'

--(Continues on next page)
```

```
-- Show the browser form and based on the file, fill the grid
!browser = object PMLFILEBROWSER('OPEN')
!browser.show('C:\', '', 'Load Grid from Excel', true, 'Excel Documents|*.xls', 1)
!this.grid.clearGrid()
!nds = object NETDATASOURCE('Grid Control Example', !browser.file())
!this.grid.BindToDataSource(!nds)
endmethod
-- End of Method .loadFromExcel()
--
-----
-- Method ..afterSelectChange(ARRAY) - Empty method, ready to be called by the control --
-----
define method .afterSelectChange(!data is ARRAY)

endmethod
-- End of Method .loadFromExcel()
--
-----
-- Method ..beforeCellUpdate(ARRAY) - Perform a DB update with the supplied data --
-----
define method .beforeCellUpdate(!data is ARRAY)
!this.grid.doDabaconCellUpdate(!data)
endmethod
-- End of Method .beforeCellUpdate(ARRAY)
--
-----
-- End of Form definition and methods --
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----
```

Copia para EEAA

Appendix B10 - Example c2ex9.pmlfrm

```
--
-- (c) Copyright 2009 to Current Year AVEVA Solutions Limited
--
-- File: c2ex9.pmlfrm
--
-- Description:
-- Example form solution to Exercise 9
--
-- Supplied to support AVEVA training course TM1881 - PML: Form Design
--
-----
-- Form definition
-----

setup form !!c2ex9 dialog docking

-- Load the required .dll for the explorer .NET gadget
import 'GridControl'
handle ANY
endhandle
import 'ExplorerAddin'
handle ANY
endhandle
using namespace 'Aveva.Core.Presentation'

-- Set the title and initialisation call for the form
!this.formTitle = |Equipment Viewer|
!this.iconTitle = |VIEWER|
!this.callback = |!this.init()|
!this.firstShownCall = |!this.firstShown()|
!this.killingCall = |!this.close()|
!this.formRevision = |1.0|

-- Create a popup menu object
!this.newMenu(|apopup|)
!this.apopup.add('CALLBACK', 'Navigate to Element', '!!ce = !this.selection')
!this.newMenu(|gridPopup|)
!this.gridPopup.add('CALLBACK', 'Navigate To Nozzle.', '!this.showNozzle()')

-- Set up the container gadgets - Notice the use of variables
container .explorerFrame NOBOX PMLNETCONTROL anchor t+l width 35 height 10
!pos = |at xmin.explorerFrame ymax.explorerFrame|
!size = |width 35 height 10|
container .attributeFrame NOBOX PMLNETCONTROL $!pos anchor t+l+b $!size

frame .viewFrame panel at xmax + 0.5 ymin.explorerFrame anchor ALL
view .volumeView VOLUME
width 50 height 20
border off
shading on
isometric 3
exit
exit

-- Specify the user-defined form members
member .attributeGrid is NETGRIDCONTROL
member .designExplorer is PMLEXPLORERCONTROL
member .selection is DBREF
member .drawlist is REAL
exit

-- Constructor Method
-----
define method .c2ex9()
-- Specify Namespace for .Net
using namespace 'Aveva.Core.Presentation'

-- Setup the .Net Controls
!this.designExplorer = object PMLEXPLORERCONTROL()
!this.attributeGrid = object NETGRIDCONTROL()
!this.explorerFrame.control = !this.designExplorer.handle()
!this.attributeFrame.control = !this.attributeGrid.handle()

-- (Continues on next page)
```

```
-- add PML event handlers to .Net control
!this.designExplorer.addeventhandler('OnPopup', !this, 'rightClickExplorer')
!this.designExplorer.addeventhandler('OnSelectionChanged', !this, 'updateData')
!this.designExplorer.initialise('/EQUIP', '')
!this.attributeGrid.addeventhandler('OnPopup', !this, 'rightClickGrid')

-- Create a local drawlist within the global object
!this.drawlist = !!gphDrawlists.createDrawList()
endmethod

-----
-- FirstShown Method
-----

define method .firstShown()
-- Add 3D view to view system (this shows the form)
!!gphViews.add(!this.volumeView)

-- add local drawlist to 3D view
!!gphDrawlists.attachView(!this.drawlist, !this.volumeView)
!!gphDrawlists.drawlists[!this.drawlist].holes(TRUE)
endmethod

-----
-- Initialisation Method
-----

define method .init()
-- Setup the Grid Control Gadget
!this.attributeGrid.columnExcelFilter(FALSE)      $* Turn off the filters
!this.attributeGrid.outlookGroupStyle(FALSE)      $* Turn off outlook grouping
!this.attributeGrid.fixedHeaders(FALSE)           $* Turn off fixed headers
!this.attributeGrid.fixedRows(FALSE)              $* Turn off fixed rows
!this.attributeGrid.columnSummaries(FALSE)        $* Turn off summaries
!this.attributeGrid.setLabelVisibility(TRUE)       $* Show Label
!this.attributeGrid.clearGrid()                   $* Clear any existing data
endmethod

-----
-- Close Method
-----

define method .close()
-- On unloading the form, remove drawlist from view and delete it
!!gphDrawlists.detachView(!this.volumeView)
!!gphDrawlists.deleteDrawlist(!this.drawlist)
endmethod

-----
-- Method .updateSystemData(ARRAY) - runs when the explorer is clicked
-----

define method .updateData(!data is ARRAY)

-- Establish the chosen element in the explorer
!this.selection = !data[0].dbref()

-- loop to find an EQUI.
do
break if !this.selection.type.eq(|EQUI|).or(!this.selection.type.eq(|WORL|))
!this.selection = !this.selection.owner
enddo

-- If an EQUI is found, then fill in the information
if !this.selection.type.eq(|EQUI|) then
!this.fillAttributes()
endif

endmethod

-----
-- Method .rightClickExplorer(ARRAY) - runs when the explorer is right clicked
-----

define method .rightClickExplorer(!data is ARRAY)
-- Apply the popup to the container
!this.explorerFrame.popup = !this.apopup

-- Show the popup menu
!this.explorerFrame.showPopup(!data[0], !data[1])
endmethod

-----
-- Method .rightClickGrid(ARRAY) - runs when the grid is right clicked
-----

define method .rightClickGrid(!data is ARRAY)

--(Continues on next page)
```

```
-- Apply the popup to the container
!this.attributeFrame.popup = !this.GridPopup

-- Show the popup menu
!this.attributeFrame.showPopup(!data[0], !data[1])
endmethod

-----
-- Method .showNozzle() - runs when the user selected from the popup --
-----
define method .showNozzle()
    !!ce = !this.attributeGrid.getSelectedrows()[1][1].dbref()
endmethod

-----
-- Method .fillAttributes() - fills the grid will information --
-----
define method .fillAttributes()

    -- Specify Namespace for .Net
    using namespace 'Aveva.Core.Presentation'

    -- Set the Views limits based on the chosen element
    !!gphViews.limits(!this.volumeView, !this.selection)

    -- Update the drawlist by removing the previous and adding the chosen
    !drawlist = !!gphDrawlists.drawlist(!this.drawlist)
    !drawlist.removeall()
    !drawlist.add(!this.selection)

    -- Active the volume view
    !this.volumeView.active = TRUE

    -- Fill in the Grid Control Gadget
    -- Collect all the NOZZs. PML1 syntax used as array of strings needed
    var !data collect all NOZZ for $!this.selection

    -- Attributes required for each nozzle to be filled in
    !attributes = |NAME CPAR[1] CREF|

    -- The titles of the columns for the attributes
    !titles = |Nozzle Size Connection|

    -- Required information passed to a NetDataSource object to compile the info
    !nds = object NETDATASOURCE('Grid', !attributes.split(), !titles.split(), !data)
    !this.attributeGrid.bindToDataSource(!nds)

    -- Set the name column to hold icons of DB elements and autofit the columns
    !this.attributeGrid.setNameColumnImage()
    !this.attributeGrid.autoFitColumns()

    -- Alter the information in the third column (connected)
    -- Define the icons for use in the column
    !error = !!pml.getPathName('ad_t_019.png')
    !ok = !!pml.getPathName('ad_t_018.png')

    -- Collect the current values in column three and loop through them
    !connected = !this.attributeGrid.GetColumn(3)
    do !n index !connected
        !testValue = !this.attributeGrid.getCell(!n, 3)
        !element = !this.attributeGrid.getCell(!n, 1).dbref()
        if !testValue.eq(|Nulref|) then
            !this.attributeGrid.setCellImage(!n, 3, !error) $* if found Null element
            !this.attributeGrid.setCellValue(!n, 3, |Check!|) $* apply the missing icon
            !this.attributeGrid.setCellColor(!n, 3, |white|) $* change the cell value
            !!gphDrawlists.drawlists[!this.drawlist].highlight(!element, 2)
        else
            !this.attributeGrid.setCellImage(!n, 3, !ok) $* apply the attached icon
            !!gphDrawlists.drawlists[!this.drawlist].highlight(!element, 5)
        endif
    enddo
endmethod

-----
-- End of Form definition and methods --
-----
-- (c) Copyright 2010 to Current Year AVEVA Solutions Limited --
-----
```


Appendix B11 - Example appml.pmlobj

PML addin file

```
name:          PMLTraining
title:         PML Training
showOnMenu:    FALSE
object:        appPML
```

appml.obj

```
define object appPML
endobject

define method .modifyMenus()
!this.utilitiesMenu()
endmethod

define method .modifyForm()
endmethod

define method .utilitiesMenu()
!menu = object APPMENU('SYSUTIL')
!menu.add('SEPARATOR')
!menu.add('FORM', |Calculator|, |c2ex1|)
!menu.add('MENU', |Equipment Checker|, 'EquipCheck')
!menu.add('SEPARATOR')
!!appMenuCntrl.addMenu(!menu, 'EQUI')

!menu= object APPMENU('EquipCheck')
!menu.add('FORM', |Exercise 4...|, |c2ex4|)
!menu.add('FORM', |Exercise 5...|, |c2ex5|)
!!appMenuCntrl.addMenu(!menu, 'EQUI')
endmethod
```

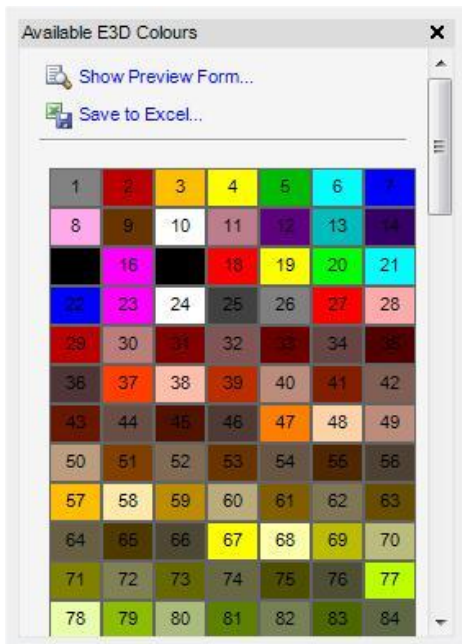
Cópia para EE AA

Copia para EE AA

Appendix C – PML training utility

This appendix explains the PML utility, supplied to support the development of PML forms and the training course. All the PML for this utility will be found in the supplied files, within the Utilities folder.

Appendix C.1 - Available AVEVA E3D Colours form



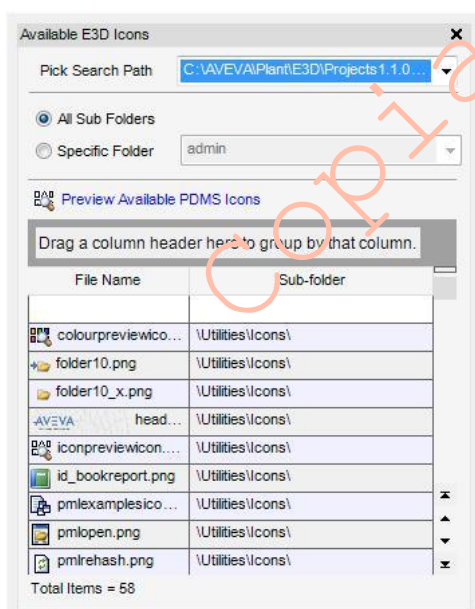
This form will display the available colours within the AVEVA E3D session. Dynamically built, it will always be correct for the current session.

If the preview form is shown, a larger version of the coloured button is available along with more information about the colour. To update the colour preview, choose a different colour from the main form.

To get a record of the colours, they can be saved to an Excel spreadsheet. This will record the colour name, name and if it's a mix colour the RGB values.

The purpose of this form is to aid with setting colours (e.g. highlighting, gadgets etc) within any customisation

Appendix C.2 - Available AVEVA E3D Icons form



This form will search the chosen PMLLIB search path for .png image files and display them in the grid control gadget. A preview of the image will be displayed and it will be possible to gain a larger preview from the context menu. PMLLIB variable can refer to more than one directory. As path separator might be used blank space (' ') or semicolon symbol(';'). Check PMLLIB and modify form definition (code line 84).

As the PMLLIB search paths could be large and contain many images, it is possible to only search specific sub folders. Standard grid control functionality is also available to help filter the search results.

The purpose of the form is to visualise all the images available in the PMLLIB search path. As the images are within the search path, these images are therefore available for use in any customisation.

Depending on the size of pml.index file, this form can take a while to collect and display all the information.

Appendix C.3 – Training Examples form

This form will provide quick access to all the training examples supplied as part of this course. Split into types, each example can be shown inside AVEVA E3D or opened in the default Windows Program.

The form builds itself based on the contents of a .xls file saved within the same folder as the form definition.

By making choices in the top two sections, different examples will be available at the bottom of the form.

The examples can be displayed within AVEVA E3D or their definition opening in the default Windows program. Choosing between the icons on the right will switch this mode:



The purpose of the form is to speed up access to the training examples and to prompt their future use as reference.

As you attend further courses, more of the options across the top will become available