

# **Altia Design 11.3**

## **User's Guide**



**December 2015**

**Altia, Inc. © 1992-2015**  
7222 Commerce Center Drive, Ste. 240  
Colorado Springs, CO 80919

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

## **Restricted Rights Legend**

Use of this manual and flexible disk(s), tape cartridge(s), or CD-ROM(s) supplied for this product is restricted to this product only. Additional copies of the program may be made for security and backup purposes only. Resale of the programs or files in their present form or with alterations is expressly prohibited.

## **Trademarks**

Altia® is a registered trademark of Altia, Inc.

Borland® is a registered trademark of Borland International, Inc.

HP-UX™ is a registered trademark of Hewlett-Packard Co.

IBM® is a registered trademark of IBM Corp.

Microsoft® and MS-DOS® are registered trademarks and Visual Basic™, Windows™, Win32s™ and Windows/NT™ are trademarks of Microsoft Corporation.

Silicon Graphics® is a registered trademark of Silicon Graphics, Inc.

SunOS® is a registered trademark and Solaris™ is a trademark of Sun Microsystems Computer Corp.

X Window System™ is a trademark of the Massachusetts Institute of Technology.

UNIX® is a registered trademark of UNIX Systems Laboratories, Inc. or its successor.

All other product names mentioned herein are the trademarks of their respective owners.

## **Notice**

The information contained in this document is subject to change without notice.

**ALTIA, INC MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.**

Altia, Inc. shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this material.

# Table of Contents

---

<b>Chapter 1: UI Overview.....</b>	<b>10</b>
<b>1.1 File Ribbon.....</b>	<b>10</b>
<b>1.1.1 Standard Controls.....</b>	<b>10</b>
<b>1.1.2 Recent Menu.....</b>	<b>11</b>
<b>1.1.3 Export Menu.....</b>	<b>11</b>
<b>1.1.4 Client.....</b>	<b>16</b>
<b>1.1.5 Help .....</b>	<b>17</b>
<b>1.2 Home Ribbon.....</b>	<b>21</b>
<b>1.2.1 Edit.....</b>	<b>21</b>
<b>1.2.2 Focus and Group.....</b>	<b>21</b>
<b>1.2.3 Visibility .....</b>	<b>22</b>
<b>1.2.4 Object Style.....</b>	<b>23</b>
<b>1.2.5 Position and Size .....</b>	<b>29</b>
<b>1.2.6 Order and Align.....</b>	<b>32</b>
<b>1.2.7 Rename Animation.....</b>	<b>34</b>
<b>1.3 Insert Ribbon .....</b>	<b>35</b>
<b>1.3.1 Model Libraries.....</b>	<b>35</b>
<b>1.3.2 Models.....</b>	<b>36</b>
<b>1.3.3 Illustrations.....</b>	<b>36</b>
<b>1.4 Instance Ribbon .....</b>	<b>39</b>
<b>1.4.1 Instance Ribbon.....</b>	<b>39</b>
<b>1.4.2 Import Changes from Base Object.....</b>	<b>39</b>
<b>1.4.3 Edit Base Object Paths .....</b>	<b>40</b>
<b>1.4.4 View Base Object.....</b>	<b>41</b>
<b>1.4.5 Un-Instance .....</b>	<b>42</b>
<b>1.5 View Ribbon .....</b>	<b>43</b>
<b>1.5.1 Zoom .....</b>	<b>43</b>
<b>1.5.2 Grid.....</b>	<b>43</b>
<b>1.5.3 Global Options .....</b>	<b>45</b>
<b>1.5.4 Resolution .....</b>	<b>47</b>
<b>1.5.5 Canvas .....</b>	<b>47</b>
<b>1.5.6 Workspace .....</b>	<b>49</b>
<b>1.6 Universe View.....</b>	<b>50</b>
<b>1.6.1 Control Bar.....</b>	<b>50</b>

1.6.2	<b>Universe.....</b>	51
<b>1.7</b>	<b>Info and Tool Panes.....</b>	52
1.7.1	<b>Info Pane.....</b>	52
1.7.2	<b>Tool Pane .....</b>	54
<b>Chapter 2: Navigator .....</b>		<b>58</b>
2.1	<b>The Object Hierarchy .....</b>	58
2.1.1	<b>Context Menu.....</b>	61
2.2	<b>Object Selection.....</b>	62
2.3	<b>Dragging Objects.....</b>	63
2.4	<b>Attributes.....</b>	64
2.5	<b>Filter Find Results.....</b>	65
<b>Chapter 3: Animation Editor.....</b>		<b>66</b>
3.1	<b>Animation Editor Layout.....</b>	67
3.1.1	<b>List area.....</b>	67
3.1.2	<b>Animation State Slider .....</b>	68
3.1.3	<b>Animation Name/State Fields .....</b>	68
3.2	<b>Animation Pane Additional Functions .....</b>	69
3.2.1	<b>Group by Focus Level .....</b>	69
3.2.2	<b>Advanced Options (Animation Attributes).....</b>	69
3.2.3	<b>Animation Pane List Context Menu.....</b>	71
3.3	<b>Defining Animation .....</b>	73
3.3.1	<b>Overview .....</b>	73
3.3.2	<b>Animation Sequences Versus Functions .....</b>	73
3.3.3	<b>Floating Point Animations.....</b>	73
3.3.4	<b>How To Define Animation.....</b>	74
3.3.5	<b>Animating Groups.....</b>	76
3.4	<b>State Interpolation.....</b>	77
3.5	<b>Replaying Animation.....</b>	78
3.6	<b>Redefining Animation .....</b>	78
3.7	<b>Moving or Transforming an Animated Object.....</b>	79
3.8	<b>Renaming Animations .....</b>	80
3.8.1	<b>Rename Animation Dialog.....</b>	80
3.8.2	<b>Animation Name List .....</b>	81
3.8.3	<b>Find and Replace .....</b>	81
3.8.4	<b>Action Buttons .....</b>	82

<b>3.8.5 Animation Names of Instances .....</b>	<b>83</b>
<b>3.9 Special Built-In Functions (Design, Runtime and Generated Code) .....</b>	<b>84</b>
<b>3.9.1 Object Input .....</b>	<b>84</b>
<b>3.9.2 Object Dimensions.....</b>	<b>84</b>
<b>3.9.3 Object Movement .....</b>	<b>86</b>
<b>3.9.4 Miscellaneous Functions.....</b>	<b>86</b>
<b>3.10 Additional Built-In Functions (Design and Runtime Only).....</b>	<b>89</b>
<b>3.10.1 Design Manipulation.....</b>	<b>89</b>
<b>3.10.2 View Manipulation.....</b>	<b>90</b>
<b>3.10.3 Object Cloning.....</b>	<b>92</b>
<b>3.10.4 Clone Manipulation.....</b>	<b>92</b>
<b>3.10.5 Miscellaneous Functions .....</b>	<b>93</b>
<b>Chapter 4: Stimulus Editor.....</b>	<b>95</b>
<b>4.1 Stimulus Editor Layout.....</b>	<b>96</b>
<b>4.2 Stimulus Pane Additional Functions.....</b>	<b>98</b>
<b>4.2.1 Stop All Timers.....</b>	<b>98</b>
<b>4.2.2 Stimulus Pane List Context Menu .....</b>	<b>98</b>
<b>4.3 Defining Input Stimulus.....</b>	<b>99</b>
<b>4.3.1 Defining the Stimulus Type .....</b>	<b>100</b>
<b>4.3.2 Defining the Stimulus Area.....</b>	<b>101</b>
<b>4.3.3 Defining Execute State and Enable Condition.....</b>	<b>104</b>
<b>4.4 Defined Stimulus List.....</b>	<b>106</b>
<b>4.5 Renaming Stimulus .....</b>	<b>107</b>
<b>4.6 Slider Example .....</b>	<b>107</b>
<b>4.7 State Interpolation.....</b>	<b>109</b>
<b>4.7.1 Rectangular Area Interpolation .....</b>	<b>109</b>
<b>4.7.2 Polar Area Interpolation.....</b>	<b>110</b>
<b>4.8 Timer Stimulus.....</b>	<b>111</b>
<b>4.8.1 Creating Timer Stimulus.....</b>	<b>112</b>
<b>4.9 Testing a Stimulus Definition.....</b>	<b>115</b>
<b>Chapter 5: Properties Editor .....</b>	<b>116</b>
<b>5.1 The Property Pane .....</b>	<b>116</b>
<b>5.1.1 The Property List.....</b>	<b>117</b>
<b>5.1.2 Standard Properties.....</b>	<b>117</b>
<b>5.1.3 Custom Properties.....</b>	<b>118</b>
<b>5.1.4 Changing Property Value.....</b>	<b>118</b>

5.1.5	<b>Changing Property Names Inline.....</b>	119
5.1.6	<b>The Menu Button.....</b>	119
5.1.7	<b>The Context Menu .....</b>	121
<b>5.2</b>	<b>Property Edit Dialog .....</b>	123
5.2.1	<b>Property Name and Description.....</b>	123
5.2.2	<b>Link Definition Tab.....</b>	124
5.2.3	<b>Property Input Tab .....</b>	130
<b>5.3</b>	<b>Learning More About Properties .....</b>	131
<b>Chapter 6: Control Editor.....</b>		132
6.1	<b>Control Code Overview.....</b>	133
6.2	<b>User Interface .....</b>	135
6.3	<b>Control Editor Toolbar.....</b>	135
6.3.1	<b>Debugging Controls.....</b>	135
6.3.2	<b>Breakpoint Controls .....</b>	137
6.3.3	<b>Error Controls .....</b>	139
6.3.4	<b>Code Assist.....</b>	140
6.4	<b>Statement Palette.....</b>	141
6.4.1	<b>Resizing the Statement Palette .....</b>	141
6.4.2	<b>Statement Palette Context Menu .....</b>	142
6.5	<b>Code Editor.....</b>	143
6.5.1	<b>Editing Code (Code Assistant OFF).....</b>	145
6.5.2	<b>Editing Code (Code Assistant ON).....</b>	148
6.6	<b>Control Code Statements.....</b>	150
6.6.1	<b>Code Structure .....</b>	150
6.6.2	<b>General Statements .....</b>	153
6.6.3	<b>Root Level Statements .....</b>	153
6.6.4	<b>First Level Statements.....</b>	156
6.6.5	<b>File I/O Advanced Statements.....</b>	165
6.6.6	<b>Views Advanced Statements.....</b>	167
6.6.7	<b>Miscellaneous Advanced Statements.....</b>	171
<b>Chapter 7: Connections .....</b>		173
7.1	<b>Connections Overview.....</b>	173
7.2	<b>Using the Connections Pane .....</b>	174
7.2.1	<b>Overview .....</b>	174
7.2.2	<b>Linking Two Objects Using Connections .....</b>	176
7.2.3	<b>Unlinking a Connection Between Two Objects .....</b>	178

7.2.4	<b>Showing the Objects Associated With a Link .....</b>	178
7.2.5	<b>Adding a New Connector to an Object.....</b>	179
7.2.6	<b>Viewing, Modifying, or Deleting a Connection Definition.....</b>	181
<b>7.3</b>	<b>External Connections.....</b>	<b>182</b>
7.3.1	<b>How External Connections Work.....</b>	182
7.3.2	<b>External Connection Animation Names .....</b>	183
7.3.3	<b>Advantages to Using External Connections.....</b>	184
7.3.4	<b>Using External Connections with Cloned Objects.....</b>	184
<b>Chapter 8: Find Pane .....</b>		<b>185</b>
<b>8.1</b>	<b>Find Overview.....</b>	<b>186</b>
8.1.1	<b>Text Find .....</b>	186
8.1.2	<b>Font Find.....</b>	187
8.1.3	<b>Object Find.....</b>	187
<b>8.2</b>	<b>Find Pane User Interface .....</b>	<b>187</b>
<b>8.3</b>	<b>Find Editor Toolbar.....</b>	<b>188</b>
8.3.1	<b>Find Controls.....</b>	188
8.3.2	<b>Result Controls.....</b>	188
<b>8.4</b>	<b>Find Results List .....</b>	<b>189</b>
<b>8.5</b>	<b>Find Criteria Dialog.....</b>	<b>191</b>
<b>8.6</b>	<b>Find Expressions.....</b>	<b>193</b>
<b>Chapter 9: Classing.....</b>		<b>198</b>
<b>9.1</b>	<b>Classing.....</b>	<b>199</b>
9.1.1	<b>Instances .....</b>	199
9.1.2	<b>Create an Instance from a Base Object.....</b>	200
9.1.3	<b>Instance Ribbon.....</b>	202
9.1.4	<b>Import Changes from Base Object.....</b>	202
9.1.5	<b>Edit Base Object Paths .....</b>	203
9.1.6	<b>View Base Object.....</b>	204
9.1.7	<b>Un-Instance .....</b>	205
9.1.8	<b>Collaboration Using Classing .....</b>	205
9.1.9	<b>Key Terms .....</b>	205
<b>Chapter 10: Models Libraries.....</b>		<b>206</b>
<b>10.1</b>	<b>Opening a Models Library .....</b>	<b>207</b>
<b>10.2</b>	<b>Models View.....</b>	<b>212</b>
10.2.1	<b>Insert As Copy .....</b>	213
10.2.2	<b>Insert As Instance.....</b>	213

<b>10.3 Using Properties .....</b>	<b>214</b>
<b>10.3.1 Opening the Properties Pane .....</b>	<b>214</b>
<b>10.3.2 Setting Property Values.....</b>	<b>215</b>
<b>10.3.3 Adding and Modifying Properties.....</b>	<b>215</b>
<b>10.4 Using Connections .....</b>	<b>216</b>
<b>10.4.1 Opening the Connections Pane.....</b>	<b>216</b>
<b>10.4.2 Linking Two Objects Using Connections .....</b>	<b>217</b>
<b>10.4.3 Unlinking a Connection Between Two Objects .....</b>	<b>218</b>
<b>10.4.4 Showing the Objects Associated With a Link .....</b>	<b>218</b>
<b>10.4.5 Adding and Modifying Connections.....</b>	<b>219</b>
<b>10.5 Advanced Object Models.....</b>	<b>220</b>
<b>10.5.1 Blur Filter.....</b>	<b>221</b>
<b>10.5.2 Clip Object Model.....</b>	<b>222</b>
<b>10.5.3 Deck Object Model .....</b>	<b>225</b>
<b>10.5.4 Image Object .....</b>	<b>227</b>
<b>10.5.5 Language Object.....</b>	<b>227</b>
<b>10.5.6 Layer Manager Object Model.....</b>	<b>232</b>
<b>10.5.7 Mask Object.....</b>	<b>244</b>
<b>10.5.8 Multi-Line Text Object Models .....</b>	<b>249</b>
<b>10.5.9 Multi Plot Object Models.....</b>	<b>262</b>
<b>10.5.10 Plot Object Models.....</b>	<b>282</b>
<b>10.5.11 Pie Object Model.....</b>	<b>286</b>
<b>10.5.12 Skin Object Model.....</b>	<b>290</b>
<b>10.5.13 Snapshot Object Model .....</b>	<b>298</b>
<b>10.5.14 Sound .....</b>	<b>302</b>
<b>10.5.15 Text Input/Output Object Models .....</b>	<b>305</b>
<b>10.5.16 3D Scene Object .....</b>	<b>326</b>
<b>Chapter 11: Validator .....</b>	<b>344</b>
<b>11.1 Validator Layout .....</b>	<b>345</b>
<b>11.1.1 Toolbar .....</b>	<b>346</b>
<b>11.1.2 Problem List.....</b>	<b>347</b>
<b>11.1.3 Problem Details .....</b>	<b>348</b>
<b>11.2 Ribbon Controls .....</b>	<b>349</b>
<b>11.3 Real Time Validation .....</b>	<b>349</b>
<b>11.3.1 Operation.....</b>	<b>349</b>
<b>11.3.2 Problem Detection.....</b>	<b>350</b>

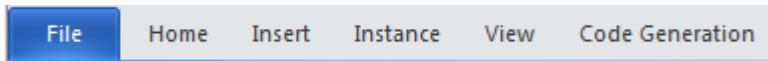
<b>Chapter 12: DeepScreen Code Generation .....</b>	<b>351</b>
<b>12.1 Code Generation Ribbon .....</b>	<b>351</b>
<b>12.1.1 Generate Source Code .....</b>	<b>351</b>
<b>12.1.2 Make Standalone .....</b>	<b>352</b>
<b>12.1.3 Run on PC .....</b>	<b>352</b>
<b>12.1.4 Pack MiniGL Data.....</b>	<b>352</b>
<b>12.1.5 Make with Editor API.....</b>	<b>353</b>
<b>12.1.6 Make Clean.....</b>	<b>353</b>
<b>12.1.7 Open Project Folder .....</b>	<b>353</b>
<b>Chapter 13: Debugger.....</b>	<b>354</b>
<b>13.1 Debugger Pane Overview.....</b>	<b>354</b>
<b>13.2 Control Buttons .....</b>	<b>356</b>
<b>13.3 Event Log.....</b>	<b>357</b>
<b>13.4 Filters.....</b>	<b>360</b>
<b>13.5 Watch List.....</b>	<b>362</b>
<b>13.6 Saving the Debugger Log.....</b>	<b>362</b>
<b>Chapter 14: Layer Manager .....</b>	<b>363</b>
<b>14.1 Layer Manager Pane Layout .....</b>	<b>365</b>
<b>14.1.1 Layer List .....</b>	<b>365</b>
<b>14.1.2 Layer Group List.....</b>	<b>368</b>
<b>14.1.3 Layer Manager Layer List Context Menu.....</b>	<b>369</b>
<b>14.1.4 Layer Manager Layer Group List Context Menu .....</b>	<b>371</b>
<b>Appendix A - Keyboard Shortcuts.....</b>	<b>372</b>
<b>Appendix B - Altia Design File Settings and Altia Editor Settings .....</b>	<b>377</b>
<b>B.1 Altia Design File Settings (.rtm files).....</b>	<b>377</b>
<b>B.2 Altia Editor Settings (altia.ini file).....</b>	<b>378</b>
<b>Appendix C - Altia Runtime.....</b>	<b>380</b>
<b>C.1 Creating A Custom Altia Runtime .....</b>	<b>380</b>
<b>C.2 Executing A Custom Altia Runtime .....</b>	<b>381</b>
<b>C.3 Customizing The Runtime Configuration File (.rtm).....</b>	<b>383</b>
<b>C.4 Terminating a Custom Altia Runtime .....</b>	<b>383</b>
<b>C.5 Special Operations Using Altia Runtime.....</b>	<b>384</b>

# Chapter 1: UI Overview

## 1.1 File Ribbon

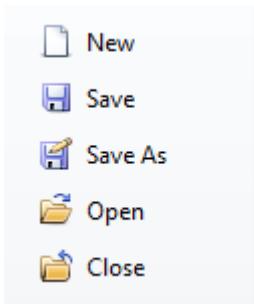
The File Menu provides access to standard file operations such as loading and saving the Altia Design project. Also provided are controls for exporting data, managing client programs, and accessing the product help.

The File Menu is accessed from the **File** button on the ribbon:



### 1.1.1 Standard Controls

Altia provides the standard file controls similar to other desktop applications:

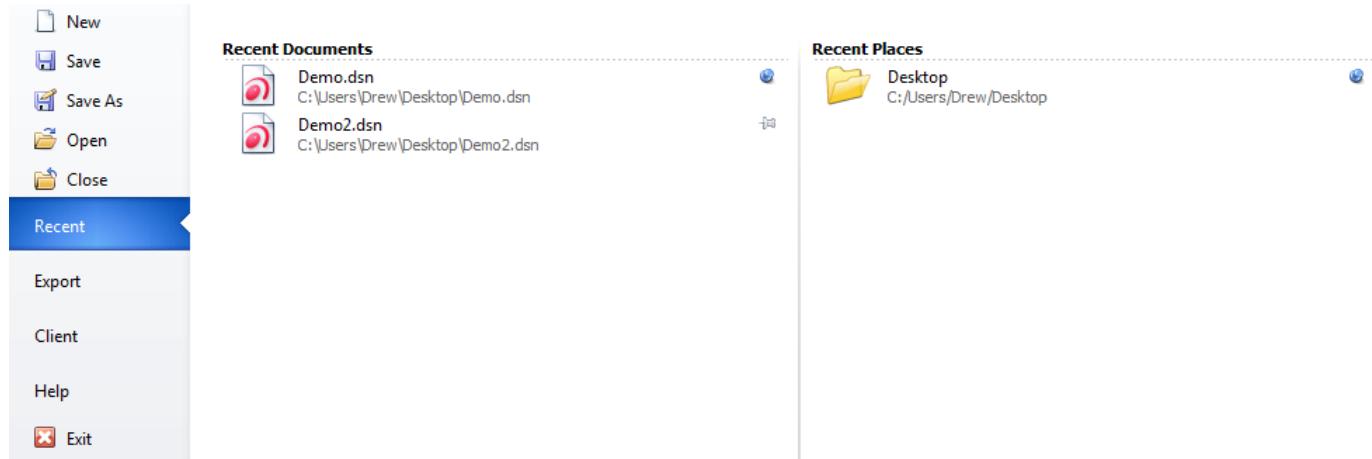


These controls are:

- **New** - Shows the **New Design** dialog.
- **Save** - Saves the current design project. Behaves same as **Save As** if the current design has never been saved.
- **Save As** - Shows the **Save File As** dialog.
- **Open** - Shows the **Open File** dialog.
- **Close** - Closes the current design project and shows the **Welcome** dialog.

## 1.1.2 Recent Menu

The Recent Menu shows a list of recently opened design project files as well as recent places where the files resided:



## Pinning Items

Both the **Recent Files** and **Recent Places** lists allow for pinning. Pinning an item in the list will move the item to the top. Pinned items will always appear in the list before unpinned items.

## Context Menus

Right clicking in either the **Recent Files** or **Recent Places** lists will show a context menu. The context menu contains controls that will:

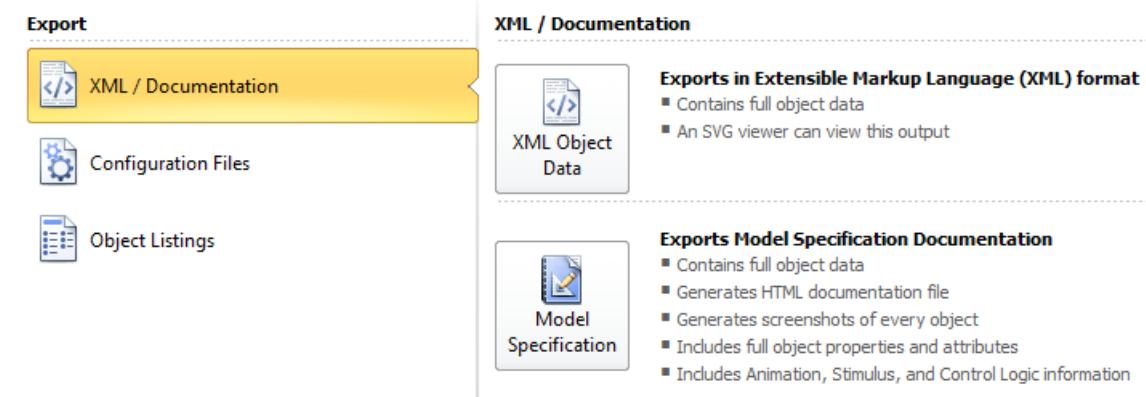
- **Open** - Opens the right-clicked item in the list.
- **Pin/Unpin** - Toggles "pinned" state for the right-clicked item in the list.
- **Remove** - Removes the right-clicked item from the list.
- **Clear Unpinned Documents** - Clears **ALL** unpinned items, removing them from the list.

## 1.1.3 Export Menu

The Export Menu provides controls for exporting XML documentation, skin/language data, and lists of object data.

## XML / Documentation Menu

Clicking the XML / Documentation Menu will show the following controls:

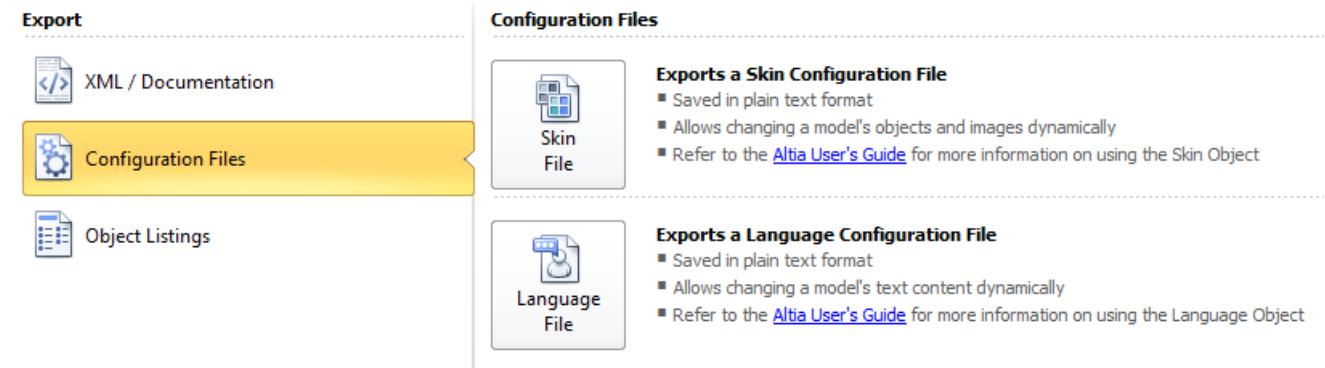


The XML / Documentation Menu contains the following controls:

- **XML Object Data** - Exports an XML version of the design project.
- **Model Specification** - Exports an HTML documentation file for the design project.

## Configuration Files Menu

Clicking the Configuration Files Menu will show the following controls:

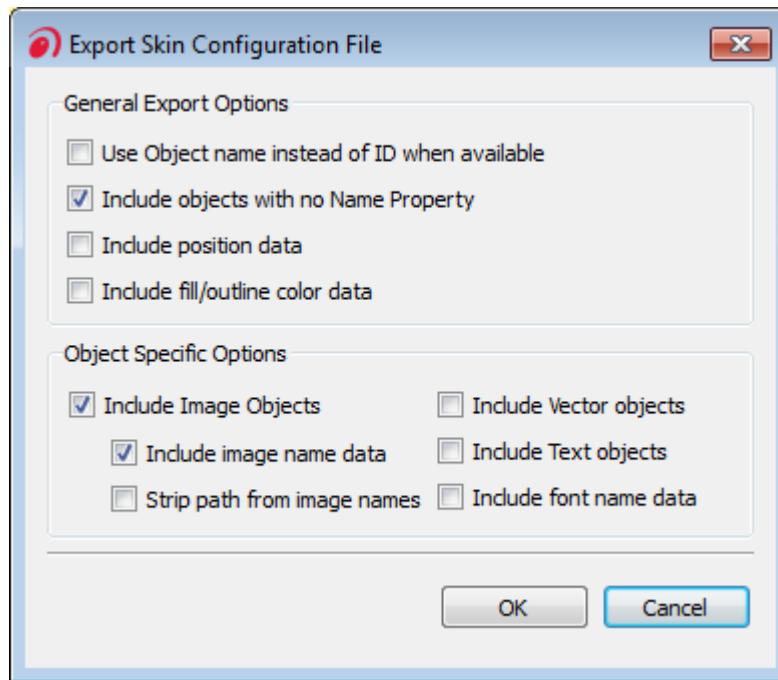


The Configuration Files Menu contains the following controls:

- **Skin File** - Shows the **Export Skin File** dialog.
- **Language File** - Shows the **Export Language File** dialog.

## Export Skin Configuration File

The Export Skin File dialog provides controls for creating skin .txt files. The .txt files can be used with the Altia Skin Object to dramatically change the appearance of the design project.



The following options are available to customize the export operation:

- **Use Object Name** - Uses an object's Name Property instead of its unique numerical ID in the exported data.
- **Include Objects with no Name Property** - Will include an object in the exported data if it doesn't have a Name Property. Unchecking this option provides a mechanism to filter objects from the exported data using the Name Property.
- **Include Position Data** - Checking this box will include an object's position data in the export. This will reposition the object when the data is loaded using the Skin Object.
- **Include Color Data** - Checking this box will include an object's color data in the export. This will change the object's color when the data is loaded using the Skin Object.

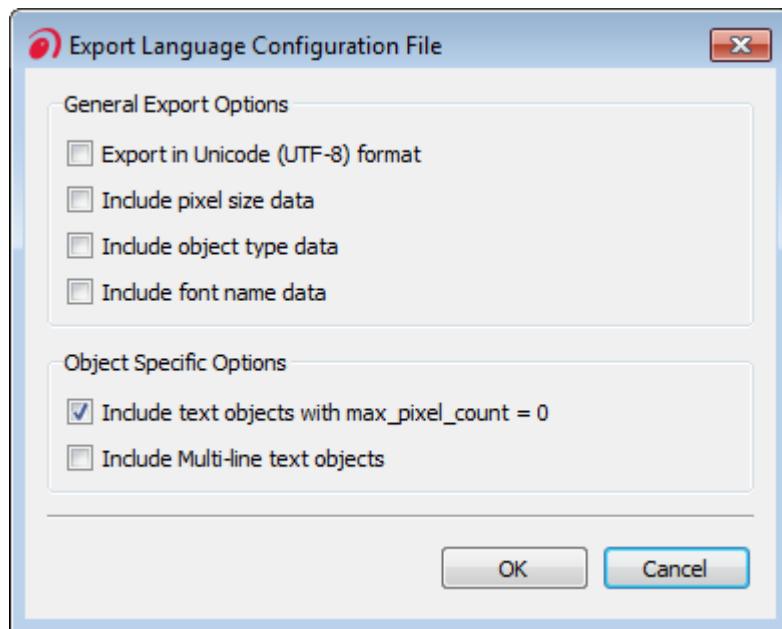
The following object-specific options are available to customize the export operation:

- **Include Vector Objects** - Will filter vector objects (lines, rectangles, etc.) from the data when unchecked.
- **Include Text Objects** - Will filter Text-IO, Multiline, and Label Objects from the data when unchecked.
- **Include Font Name Data** - Checking this box will include an object's font definition in the export. This will change the object's font when the data is loaded using the Skin Object.
- **Include Image Objects** - Will filter Image Objects from the data when unchecked.

- **Include Image Name Data** - Will include the image file name in the exported data (for Image Objects). This will load the image into the Image Object when the data is loaded using the Skin Object.
- **Strip Path from Image Names** - Checking this box will strip the path from the image file name in the exported data. This allows Image Objects to use the Skin Object's path when loading its associated image file.

## Language File Configuration

The Export Language File dialog provides controls for creating language .txt files. The .txt files can be used with the Altia Language Object to change the text present in Text-IO objects.



The following options are available to customize the export operation:

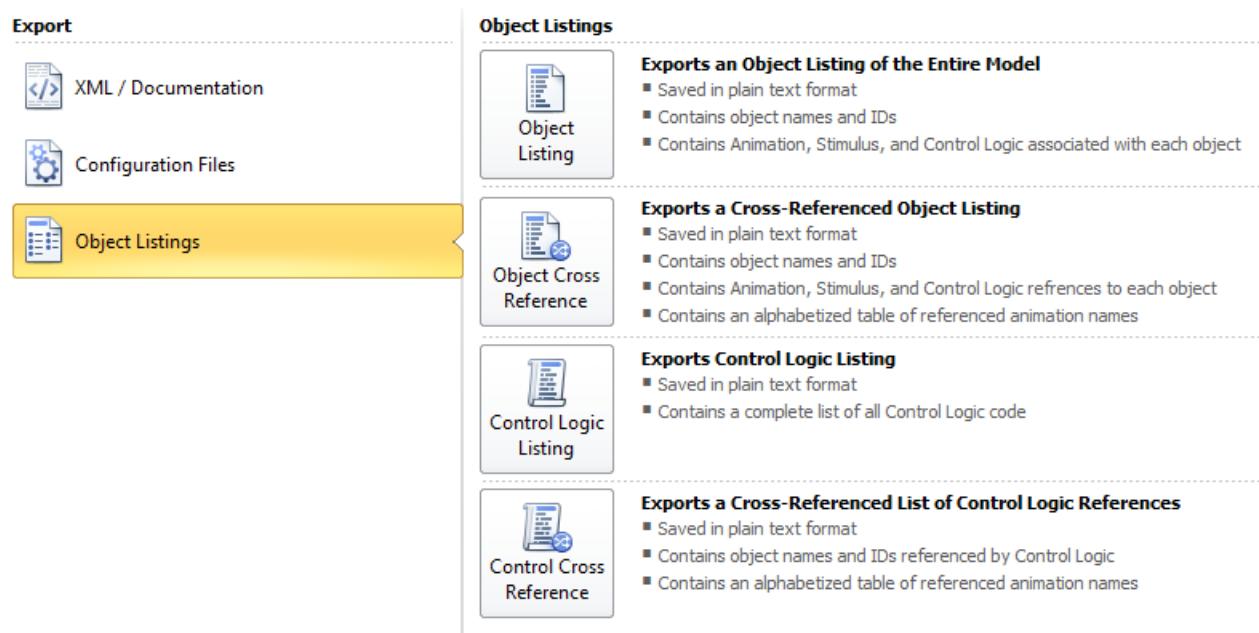
- **Export in UTF-8 Format** - Exports text data in UTF-8 instead of ASC-II.
- **Include Pixel Size Data** - Will include the pixel width of the text-io (if max\_pixel\_count is set). Not used when data is loaded by Language Object. It is provided as a metric to help create translations.
- **Include Object Type Data** - Will include object type (Text-IO or Multiline) to assist in translation. Not used when data is loaded by Language Object.
- **Include Font Name Data** - Will include the font used with each Text Object. Not used when data is loaded by Language Object. It is provided as a metric to help create translations.

The following object-specific options are available to customize the export operation:

- **Include Text Objects with max\_pixel\_count=0** - When unchecked, Text-IO objects without a max\_pixel\_count will be filtered from the exported data.
- **Include Multiline Text Objects** - When unchecked, Multiline Text Objects will be filtered from the exported data.

## Object Listings

Clicking the Object Listings Menu will show the following controls:

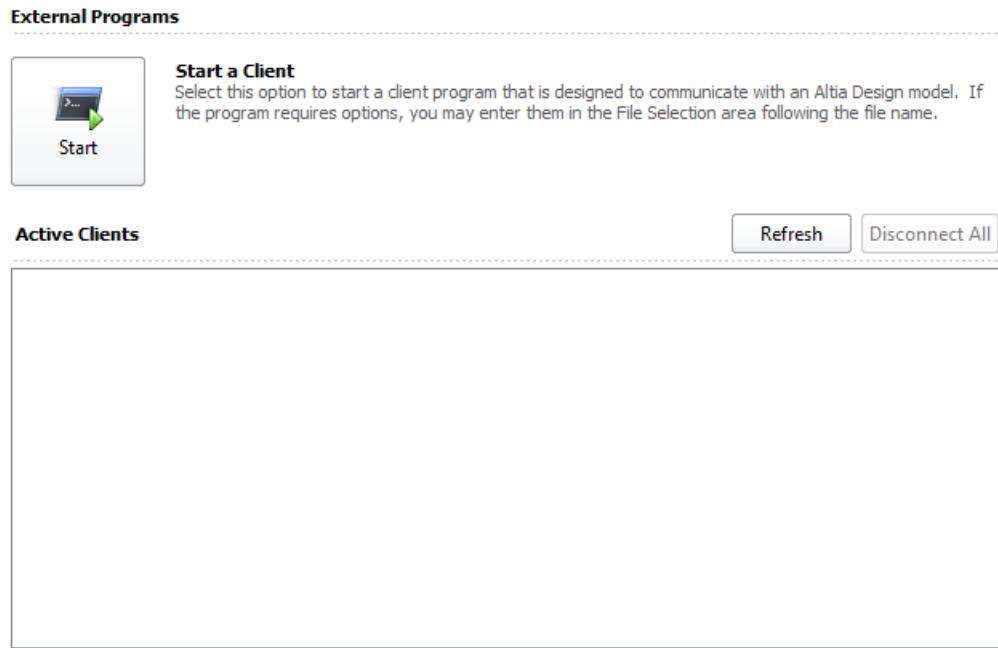


These controls are:

- **Object Listing** - Creates a list of objects used in the design project.
- **Object Cross Reference** - Creates a list of objects used in the design with a cross reference of what animations each object is referencing.
- **Control Logic Listing** - Creates a list of control code used by each object in the design project.
- **Control Cross Reference** - Creates a list of control code used in the design with a cross reference of what animations each object is referencing.

## 1.1.4 Client

The Client Menu provides controls for managing clients used with Altia Design.



### Starting a Client

Clicking this button will open a **File Open** dialog to select an application to start. Although any application can be started, the goal is to start a client application. A client application is an .exe that was built and linked with one of the Altia Design Connection Libraries (i.e. liblan.a).

After starting a client, the **Active Clients** list will be refreshed automatically. Starting a client is asynchronous and the list update may not show the newly started client as it may not have completed the connect operation.

### Active Clients List

The **Active Clients** list shows all clients that are currently connected to the Altia Design session. The list is refreshed every time the **ClientRefresh** Button.

There are two types of clients: (1) LAN, and (2) DDE. The two client types will be identified with different icons in the client list.

## Stopping an Active Client

A client can be disconnected from the Altia Design session in one of two ways:

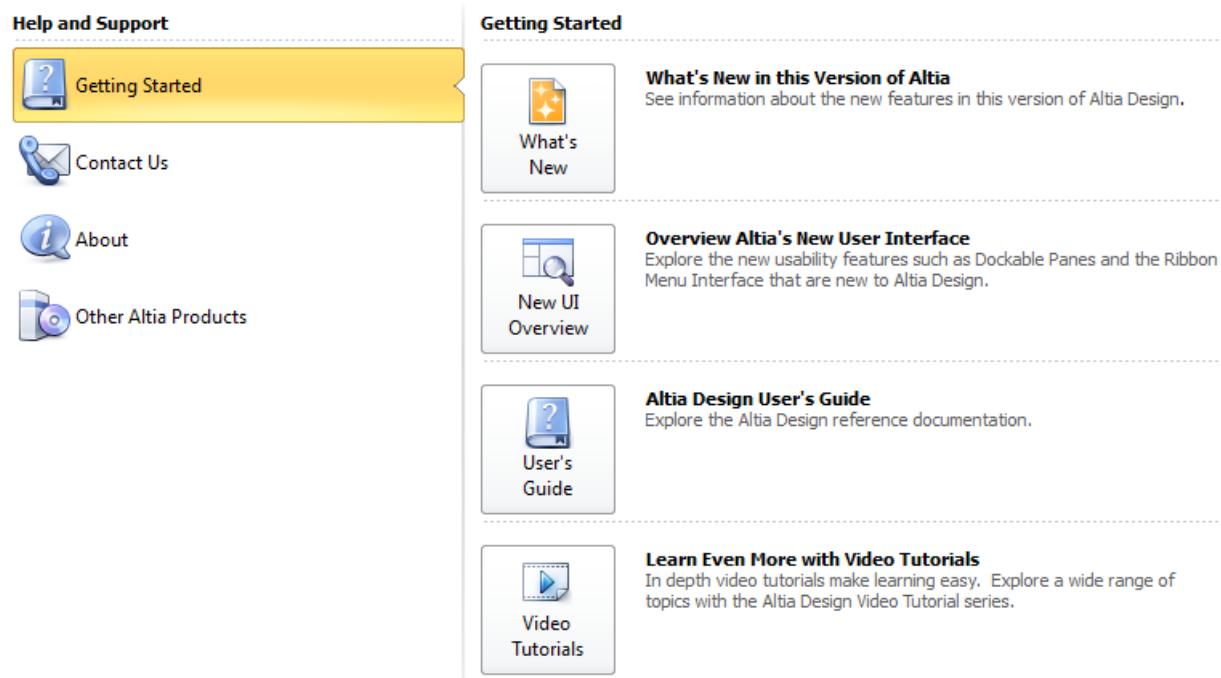
- Terminate the client .exe by closing it.
- Click the **Disconnect All** Button which will disconnect all currently connected clients.

### 1.1.5 Help

The Help Menu provides access to resources for learning about Altia Design and obtaining product support.

## Getting Started

Clicking the Getting Started Menu will show the following controls:

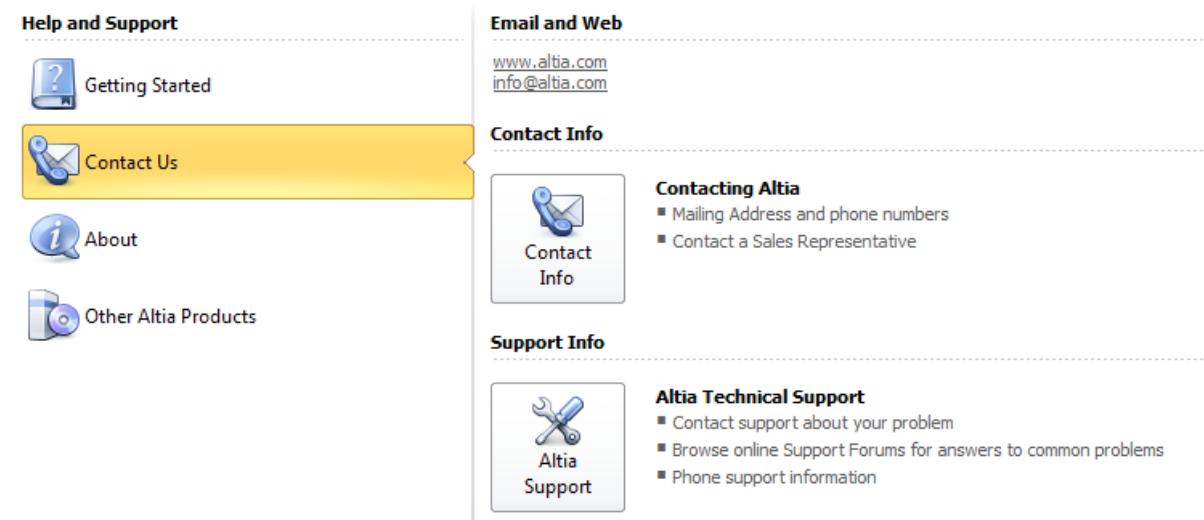


These controls are:

- **What's New** - Opens the Enhancement Summary for this release of Altia Design.
- **New UI Overview** - Opens a web browser and navigates to the Altia Design Overview Knowledge Base.
- **User's Guide** - Opens the User's Guide Documentation for this release of Altia Design.
- **Video Tutorials** - Opens a web browser and navigates to the Altia Design Video Website.

## Contact Us

Clicking the Contact Us Menu will show the following controls:



These controls are:

- **www.altia.com** - Link to the Altia Website.
- **info@altia.com** - Email link for more information about Altia.
- **Contact Info** - Opens a web browser and navigates to the Altia Contact Us Website.
- **Altia Support** - Opens a web browser and navigates to the Altia Support Website.

## About

Clicking the About Menu will show the following controls:

The screenshot shows the 'About' menu highlighted in yellow. The menu items are:

- Getting Started
- Contact Us
- About** (highlighted)
- Other Altia Products

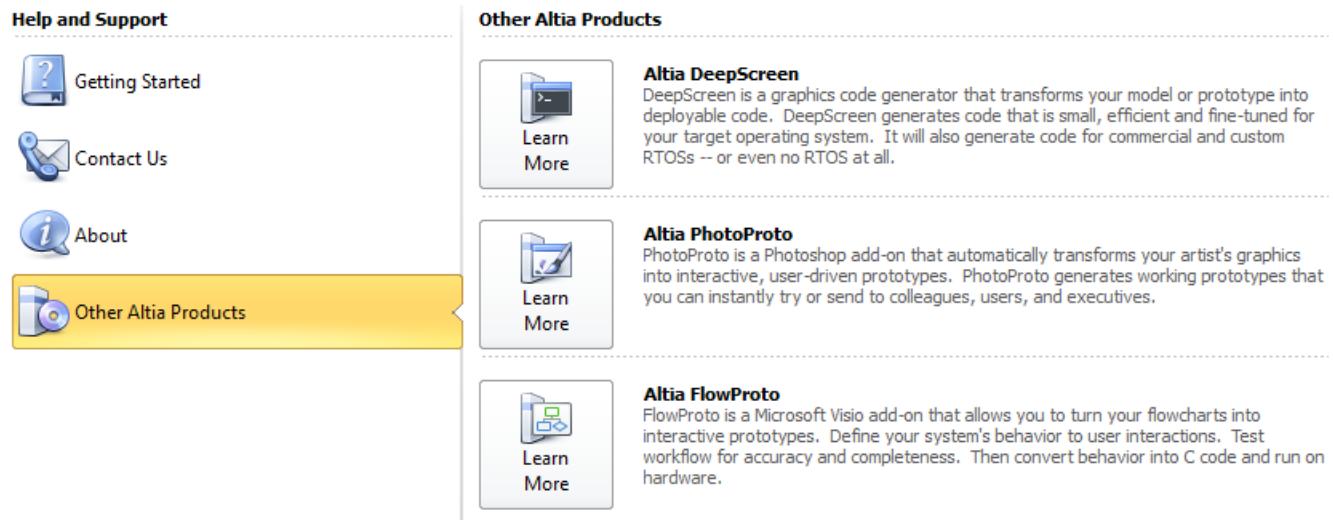
On the right side of the interface, there are several sections:

- License Information**:  
License File: C:\usr\altia11Beta6310\license/codeword.txt  
Host: SparkyII  
No Expiration
- About Altia Design**:  
Altia Design 11.0 Beta Build 6.31.0  
Copyright (c) Altia, Inc. 1991-2013  
All rights reserved worldwide  
U.S. Patents 5,883,639; 6,957,418
- Acknowledgement of 3rd Party Software Used by this Software Program**:  
The OpenSceneGraph is protected by Copyright, and is licensed under the terms of the OpenSceneGraph Public License (OSGPL). To read this license please visit: <http://www.openscenegraph.org/projects/osg/wiki/Legal>  
Nokia Corporation and/or its subsidiaries. Nokia, Qt and their respective logos are trademarks of Nokia Corporation in Finland and/or other countries worldwide. The usage of Qt in Altia Design is under the conditions of the GNU Lesser General Public License (LGPL) version 2.1. To read this license please visit: <http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>.
- Software Updates**:  
 **Check for Updates**  
Get the latest software updates available for Altia Design.

Details about your current product license and product version are presented here. A **Check for Updates** Button will open a web browser and navigates to the Altia Product Website.

## Other Altia Products

Clicking the Other Altia Products Menu will show the following controls:



These controls are:

- **Altia DeepScreen** - Opens a web browser and navigates to the Altia DeepScreen Website.
- **Altia PhotoProto** - Opens a web browser and navigates to the Altia PhotoProto Website.
- **Altia FlowProto** - Opens a web browser and navigates to the Altia FlowProto Website.

## 1.2 Home Ribbon

The Home Ribbon in Altia Design provides many of the tools that are essential to creating and editing your design, this chapter will cover their use and provide some tips and tricks to make using Altia Design a pleasurable experience.

### 1.2.1 Edit



The Edit controls operate much as you've become accustomed to from other programs like Microsoft Office and Photoshop. You can Cut and Copy selected objects (or groups of objects) and Paste them elsewhere in your design. You can even Cut and/or Copy any number of objects from different focus levels and paste them elsewhere later.

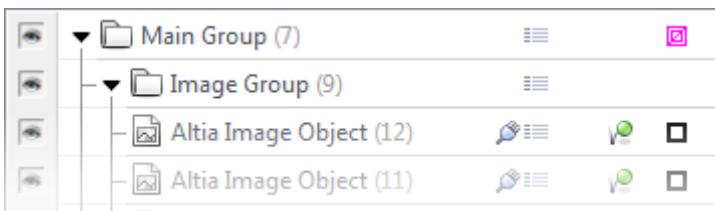
**NOTE:** When pasting multiple objects that have been copied from different focus levels the new objects will all be pasted into the current focus level (i.e. they will be flattened to the same focus level).

The Edit controls can also be used with standard shortcut key combinations; **Ctrl+C** (Copy), **Ctrl+X** (Cut), and **Ctrl+V** (Paste).

### 1.2.2 Focus and Group

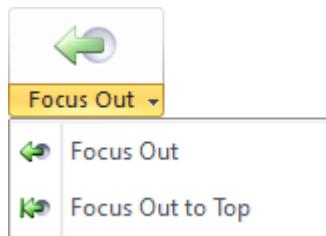


The Focus and Group controls allow you navigate through the focus levels of your design and manage the grouping of objects. Objects in an Altia Design model are organized into a tree like hierarchy; for example a typical model will have a structure much like the one below:



In this example the Main Group is considered to be at Focus Level 0, the Image Group at Focus Level 1, and so forth. Using the Focus controls you can move up and down the tree (decreasing and increasing Focus Level, respectively).

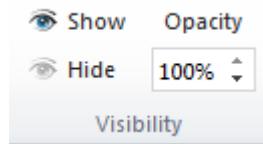
- **Focus Out (Ctrl + W):** This action is available when you're currently focused into your model (non-zero Focus Level) and will reduce your current focus by one level when clicked. The Focus Out button also has a drop down menu that can be accessed by clicking the down arrow on the Focus Out button, shown below.



In addition to the standard Focus Out by one level feature, you can also select Focus Out to Top which will bring your focus to the top level of your model (Focus Level 0).

- **Focus In (Ctrl + E):** The Focus In action is available when you have an object selected that has children, for example a Group or a Deck. When activated this option will increase the current Focus Level by 1.
- **Group (Ctrl + G):** It is often convenient to collect related objects at the same Focus Level into a group for easier management. The Group action becomes available when you have any object(s) selected, even just one and any object can be placed within a group.
- **Ungroup (Ctrl + U):** This action becomes available when you have a group selected and when clicked will break the group and move its contents up one Focus Level.

### 1.2.3 Visibility



With the Visibility controls you can modify an object's opacity or hide it completely. With the Show and Hide buttons you can set an object to be visible (with the opacity specified in the Opacity field) or to be invisible. You can also set an object's opacity to a value between 0% and 100% (fully transparent and fully opaque, respectively).

Show, Hide, and Opacity can apply to individual objects or to groups of objects. The Opacity value field can be modified by entering a decimal value between 0 and 100, using the up and down arrows next to the value, or by using your mouse wheel.

## 1.2.4 Object Style



The Style Menu gives you control over the look of primitive objects (lines, rectangles, elipses, text, etc.). You can modify an objects fill color, fill pattern, outline color, outline pattern, outline width, and even apply style properties to entire groups of objects.

The Fill and Outline controls are covered in more detail in [Fill Style Menu](#) and [Outline Style Menu](#) sections.

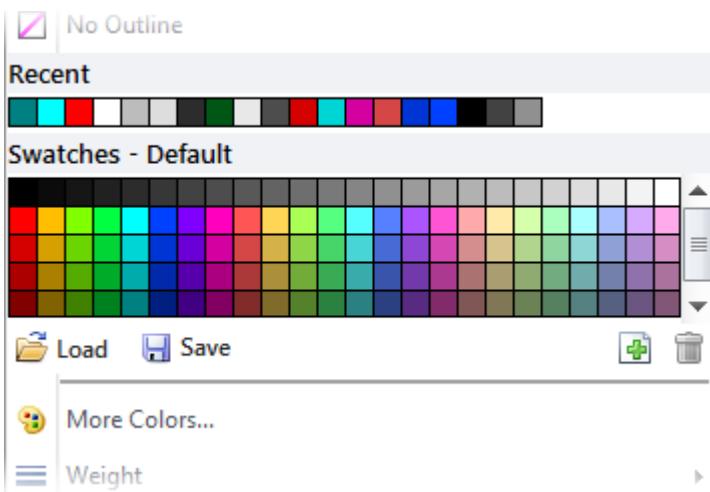
The Font Style controls should be very familiar to anyone who has used Microsoft Word or other comparable text editor. The available fonts match that of the host computer and the point size, Bold (Ctrl+B), and Italic (Ctrl+I) controls operate as one would expect.

**NOTE:** If you manually type the filename of a font file (ex. myfont.ttf) in the Font name combo box, Altia Design will search the current model's working directory to attempt to load a font with the specified filename. Directory paths relative to the current model's working directory are also allowed (ex. fonts/myfont.ttf). This allows you to create a model that uses a font that is not currently installed in Windows.

If the font does not exist in the working directory, Altia Design will then examine the installed fonts in Windows to attempt to find a match. If no match is found, Windows will substitute what it determines to be the closest match to that font.

The Clear Group Style, and the concept of Group Style in general, is unique to Altia Design. Group Style allows you to select a Group object and apply a Fill and/or Outline style which will be applied to all compatible objects (Rectangles, Lines, etc.) in the selected Group. The Clear Group Style button becomes active when a Group object with a style is selected and is used to reset the Group's style.

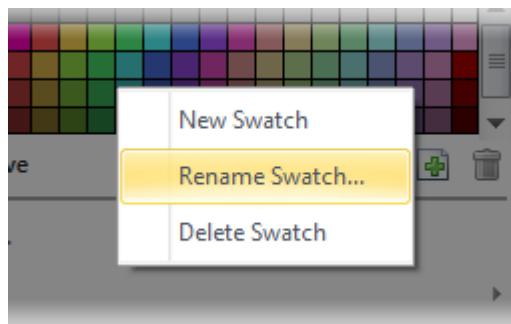
## Style Menu Color Swatches



The Fill and Style buttons both open the Style Menu which allows you to specify colors and manage color swatches as well as specific Fill or Outline settings. The Color Swatches interface is common to both menus and has a number of features that ease color management.

- **Recent:** The Recent swatch area contains a history (newest on the left) of the colors swatches you've selected (or created) during the editing of your model. Clicking on a swatch in the Recent list will apply the color of the swatch to the property being modified (Fill Color, Outline Color, Universe Color, etc.) and move it to the front of the history. The Recent list will remember the 24 most recent color swatches used, after which it discards the oldest.
- **Swatches:** The main swatch area is a scrollable view containing all of the available color swatches. By default it is loaded with a range of common colors. It is also possible to load an Altia Palette file (\*.pal) for convenience and improved collaboration with other designers.

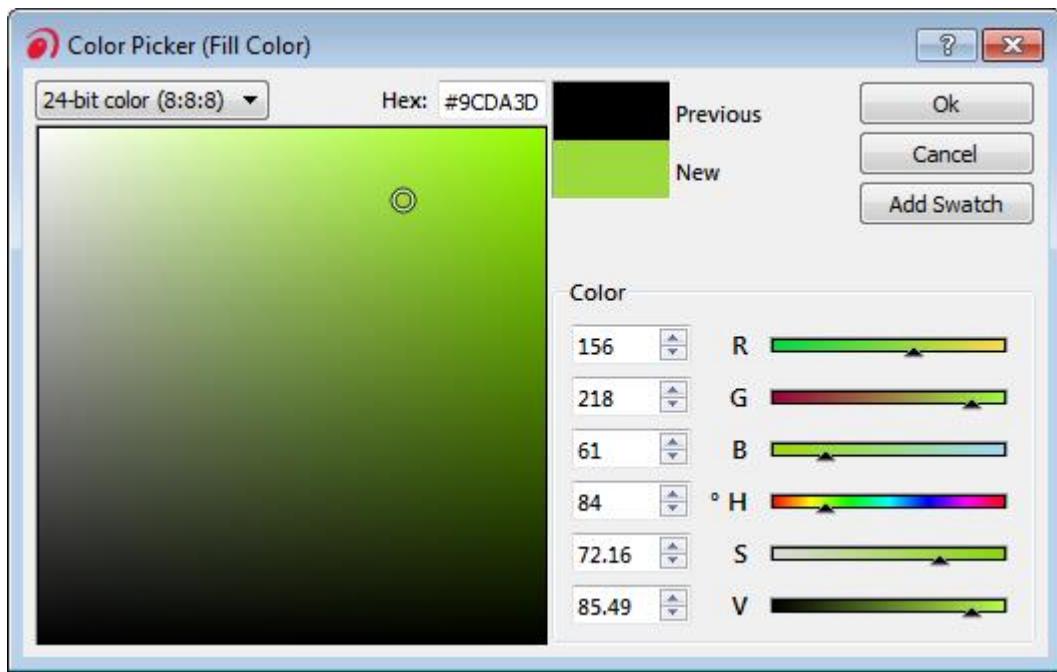
Unlike the Recent color swatch area, the main swatch area is configurable by the user. Swatches can be clicked and dragged into new positions for easy organization and every color swatch has a context menu shown below (accesable by right clicking on a color swatch).



From the color swatch context menu you can add a new color swatch, rename the swatch that you clicked on or even delete the color swatch. The New Swatch option will launch the Color Picker (See [Style Menu Color Picker](#)) that you can use to select a new color while the Rename Swatch option will raise a simple text entry dialog for you to enter the name.

- **Load:** The load action will open the Open File dialog from which you can navigate to and select a Palette file (\*.pal) that's been generated by Altia Design. When a Palette file is loaded the current set of color swatches will be removed and replaced with those from the selected file.
- **Save:** During the course of creating and modifying a design it's common to accumulate a large number of color swatches specific to the design. The save action will open a Save File dialog from which you can save your swatches into a Palette file. The names of the swatches long with their order in the swatch area will be maintained when saving and loading a Palette file.
- **Add Swatch (  ):** Much like the New Swatch action from a Color Swatch's context menu , the Add Swatch action will open a Color Picker dialog for the user to select a color for a new swatch. See [Style Menu Color Picker](#) for details about the Color Picker.
- **Remove Swatch(  ):** This icon is not a button but rather a target for a drag and drop operation. The Color Swatches support drag and drop and when a Color Swatch is dragged from the swatch area to the trash can icon and dropped it is deleted. The trash can icon is highlighted during a Color Swatch drag operation to indicate that it can be deleted via the Remove Swatch drop target.
- **More Colors:** This menu action will open a Color Picker dialog for the user to select a color for the property being modified. See [Style Menu Color Picker](#) for details about the Color Picker.

## Style Menu Color Picker



The Color Picker dialog is a full featured color tool that is used many places in Altia Design. The Color Picker supports RGB color, HSV color, and variable levels of bit depth to match common bit depths amongst LCDs.

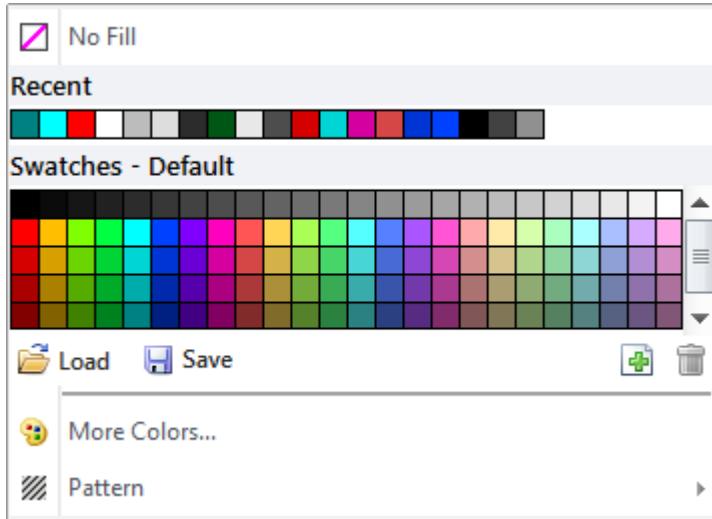
With the large Palette area you can click and drag to select a color (fixed hue, variable saturation and value) and then fine tune your color with the RGB and HSV value fields and sliders. The value sliders update dynamically to show how modifying their value will change the current color for more intuitive color editing.

The RGB value fields each support 256 values (0-255) while the Hue support 360 possible values (0-359). The Saturation and Value fields use a floating point value representation that ranges from 0.0 to 100.0.

The color depth used when rendering the Palette area (and controlling the available RGB and HSV values) can be modified with the depth pull down that's above the Palette. Possible color depth options are; 24 bit RGB888, 24 bit RGB666, 16 bit RGB565, 16 bit RGB555, and 16 bit RGB444.

Clicking on the Previous color swatch will reset the Color Picker to the previous color that was applied to the property being modified. You can also click Cancel to revert back to the previous color and close the Color Picker dialog. Clicking on Ok will apply the new color to the property being modified and close the dialog. Finally, clicking on the Add Swatch button will add a new Color Swatch with the Color Picker's current color to the swatch area. The Add Swatch will not close the dialog, making it easy to create many Color Swatches in a short amount of time.

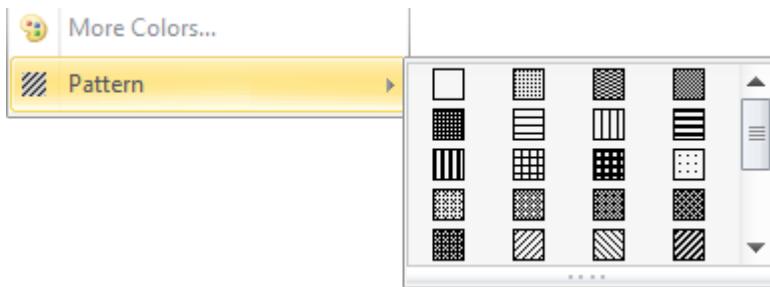
## Fill Style Menu



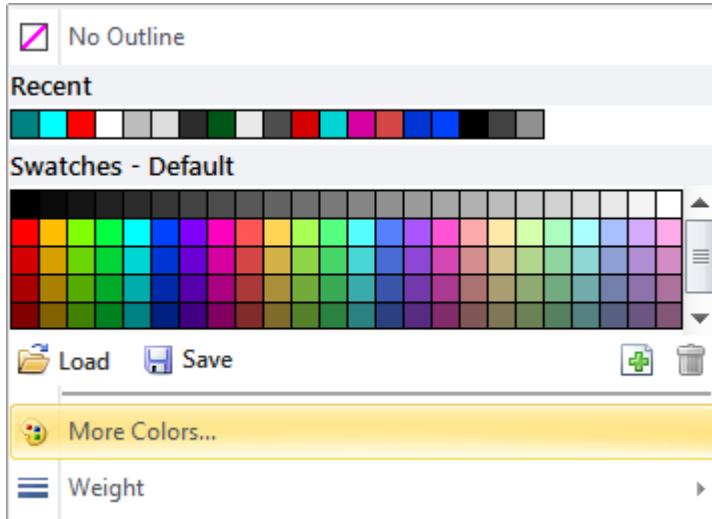
The Fill Style menu controls the fill appearance of primitive objects like Rectangles, Ellipses, etc. In addition to setting the fill color, the Fill style menu has options specific to fill behavior, specifically the No Fill and Pattern options.

The No Fill option removes any fill color and/or pattern and fills that space instead with transparent pixels.

The Pattern menu, shown below, provides a number of fill patterns in a gallery style interface. The bottom of the Pattern menu is a resize handle that you can use to expand the size of the menu. Clicking on a pattern square will apply the pattern to the selected object and close the Fill Style Menu.



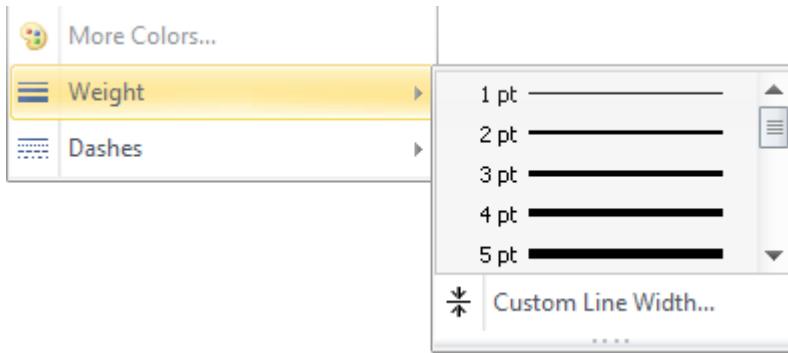
## Outline Style Menu



The Outline Style menu controls the outline appearance of primitive objects like Rectangles, Ellipses, etc. In addition to setting the outline color, the Outline style menu has options specific to outline behavior, specifically the No Outline, Weight, and Pattern options.

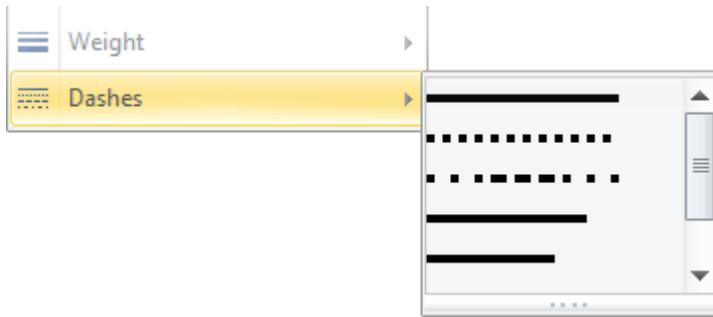
The No Outline option removes any outline color and/or pattern and fills that space instead with transparent pixels.

The Weight Menu, shown below, provides a number of outline weights to choose from with examples of their appearance. The bottom of the Weight Menu is a resize handle that you can use to expand the size of the menu. Clicking on a weight will apply it to the selected object and close the Outline Style Menu.



If you need a line weight not provided in the Weight Menu you can select the Custom Line Width... option which will open a simple dialog enabling you to specify a line weight from 1 to 64 pixels in size.

You can also apply an Outline Pattern using the Dashes Menu shown below. The bottom of the Dashes Menu is a resize handle that you can use to expand the size of the menu. Clicking on an Outline Pattern will apply it to the selected object and close the Outline Style Menu.

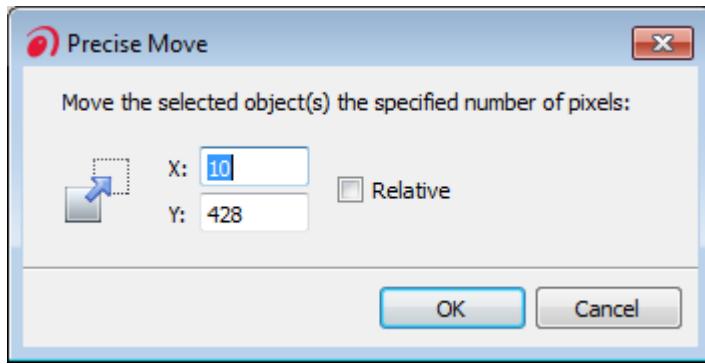


## 1.2.5 Position and Size



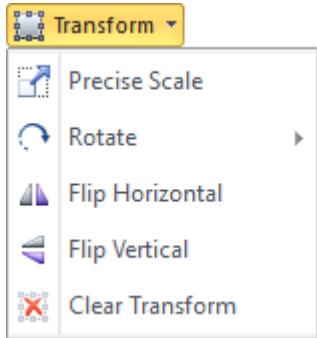
The Position and Size Ribbon group provides a number of interfaces used to move, scale, rotate, and otherwise manipulate the physical properties of an object. This Ribbon group is inactive until an object is selected. Built in controls include:

- **X:** This field displays the current X position of the selected object, which can be modified by entering a new value in this field.
- **Y:** This field displays the current Y position of the selected object, which can be modified by entering a new value in this field.
- **Delta (▲):** This control enables and disables the entry of relative position values into the X and Y value fields. When the Delta option is enabled (▲) the X/Y value fields display 0,0 and any value entered will offset the selected object by the amount specified.
- **Width (□):** This field displays the current width of the selected object. Entering a new value in this field will scale the selected object by the specified amount horizontally. Available units include Pixels, Inches, Centimeters, and Millimeters.
- **Height (□):** This field displays the current height of the selected object. Entering a new value in this field will scale the selected object by the specified amount vertically. Available units include Pixels, Inches, Centimeters, and Millimeters.
- **Aspect Ratio Lock (🔒):** When the Aspect Ratio Lock is enabled, any vertical or horizontal scaling will maintain the aspect ratio of the selected object.
- **Precise Move (Ctrl + M):**  
The Precise Move dialog is raised either by clicking the Precise Move button in the Position and Size Ribbon group or by selecting an object in the Universe Window with the Select Tool and then exercising the Ctrl+M keyboard shortcut.



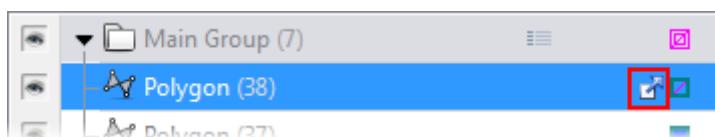
The Precise Move dialog operates identically to the Position and Size Ribbon group controls.

## Transform Options



The Transform Menu has many options for manipulating objects, they include:

- **Precise Scale (Ctrl + L):** The Precise Scale option opens the Precise Scale dialog detailed below.
- **Rotate:** This option opens the [Rotate Menu](#).
- **Flip Horizontal:** When activated this option will flip the selected object about its horizontal axis.
- **Flip Vertical:** When activated this option will flip the selected object about its vertical axis.
- **Clear Transform:** Over time an object can accumulate a lot of transformation; rotations, scaling, distortion, etc. This option will remove all transforms applied to an object. The Navigator (See [Chapter 2 - Navigator](#) for details on the Navigator) will inform you that an object has a transform via a small decoration. The image below shows the "has transform" graphic, highlighted in red:

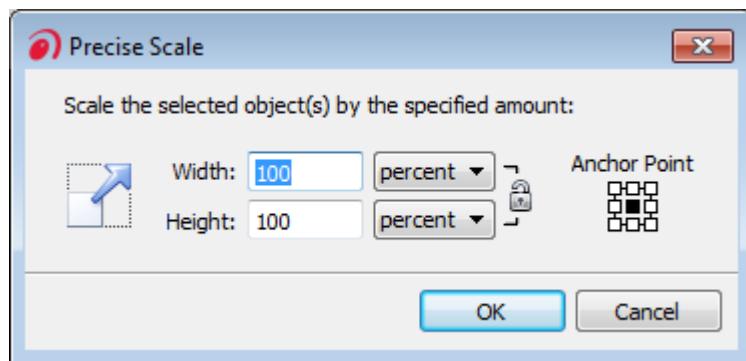


The Precise Scale dialog, shown below, is raised either by clicking the Precise Scale option in the Position and Size Ribbon group Transform Menu, or by selecting an object in the Universe Window with the Select

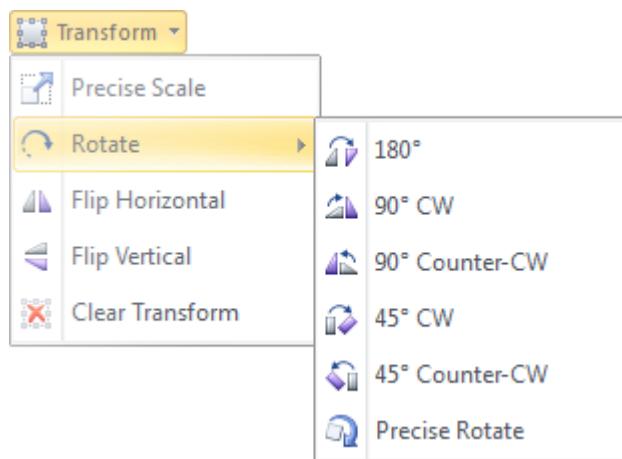
Tool and then exercising the **Ctrl+L** keyboard shortcut. The Precise Scale dialog is quite similar to the scale controls in the Position and Size Ribbon group (via the Width and Height controls) and the scale controls provided by the Scale Tool in the Tool pane).

Width and Height scaling is controlled via the Width and Height value fields with available units including Pixels, Inches, Centimeters, and Millimeters. The Precise Scale dialog also includes the Aspect Ratio Lock in order to maintain a particular ratio between width and height of the object post scaling.

Like the Scale Tool from the Tools Pane, the Precise Scale dialog includes an Anchor Point to control how the objects position changes during the scaling operation. The 9 squares that make up the Anchor Handle are selectable and specify which corner or side of the scaled object will remain in the same position before and after the scale.



## Rotate Menu

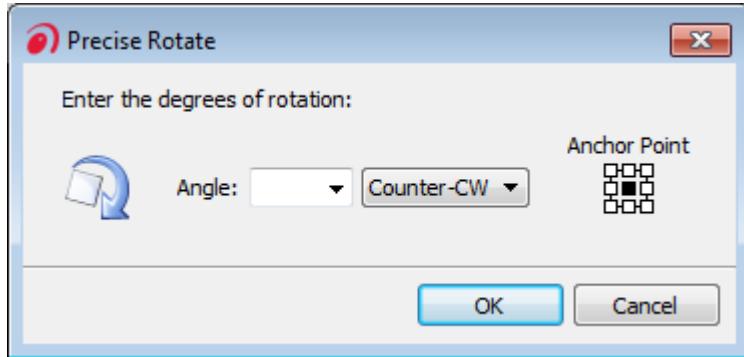


The Rotate Menu contains a number of commonly applied rotation operations along with the option to launch the Precise Rotate dialog (detailed further below).

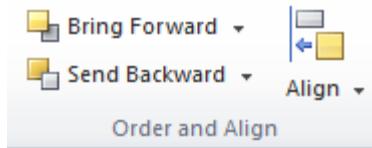
The  $180^\circ$ ,  $90^\circ$  CW,  $90^\circ$  CCW,  $45^\circ$  CW, and  $45^\circ$  CCW options all assume that the center of the selected object will be the point about which the object is rotated. For instances where this assumption is not ideal, the Precise Rotate dialog gives you more control.

The Precise Rotate dialog can be launched either from the Position and Size Ribbon group's Transform Rotate Menu or by selecting an object in the Universe Window with the Select Tool and then exercising

the **Ctrl+R** keyboard shortcut. You can enter the rotation amount in the Angle value field (negative values are accepted) and you can choose the direction of rotation (either Clockwise or Counter(Anti)-Clockwise). The Anchor Point control allows you to select the point about which the selected object will be rotated.

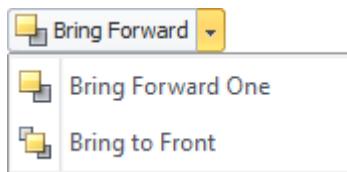


## 1.2.6 Order and Align



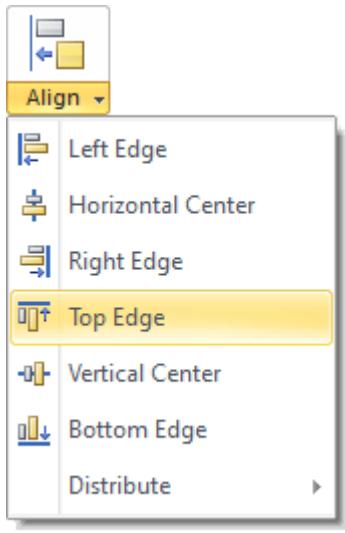
The Order and Align Ribbon group helps you manage the vertical stacking order of the objects in your design and to create precisely organized groups of objects. The vertical stacking order (or Z-Order) of objects at the same focus are easily controlled with the Bring Forward and Send Backward buttons. The Forward and Backward functions are also available via a keyboard shortcut; Bring Forward (Ctrl + K), Send Backward (Ctrl + J). Every time you push the Forward or Backward button the selected object will come up or drop down a single level in the vertical stacking order.

The Bring Forward and Send Backward buttons also have a drop down menu accessible by clicking the down arrow to the right of the button's name, shown below:



The Bring Forward One and Send Backward One options behave the same as simply clicking the respective button. The Bring to Front and Send to Back options, however, provide a convenient method to move an object to the front or back of the stacking order with a single click.

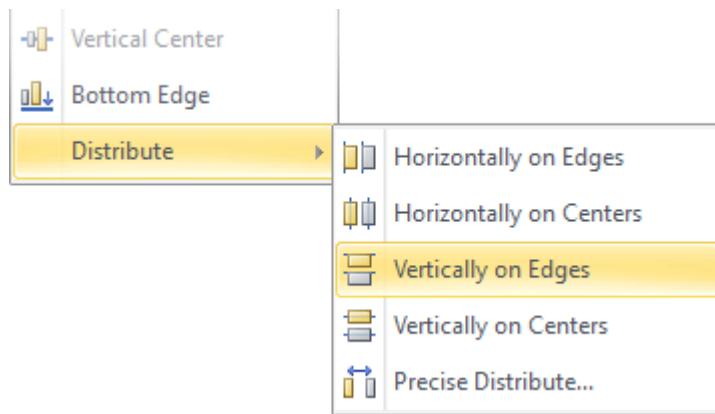
## Alignment Options



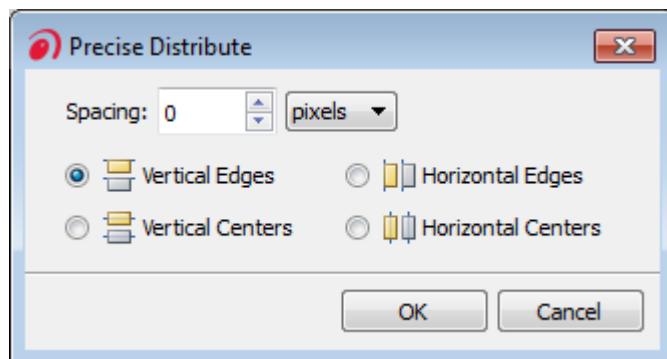
The Alignment controls make it easy to organize groups of objects.

- **Left Edge:** Objects will line up on their left most edge without changing their vertical position.
- **Horizontal Center:** Objects will line up on the same vertical axis at the midpoint of their width.
- **Right Edge:** Objects will line up on their right most edge without changing their vertical position.
- **Top Edge:** Objects will line up on their top most edge without changing their horizontal position.
- **Vertical Center:** Objects will line up on the same horizontal axis at the midpoint of their height.
- **Bottom Edge:** Objects will line up on their bottom most edge without changing their horizontal position.

## Distribute Options

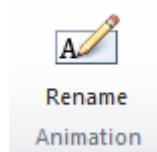


The Distribute Menu provides the ability to space out groups of objects based on either their horizontal or vertical edges and centers.



For more control over the Distribute options, the Precise Distribute dialog lets you specify an exact pixel value to be used when distributing the objects you have selected. The options behave as they do if selected from the Distribute Menu, only the spacing will be affected.

### 1.2.7 Rename Animation



The Rename button provided by the Home Ribbon Animation controls will launch the Rename Animation Dialog when clicked. The Rename Animation Dialog provides a powerful set of tools for managing animation names in your design and is covered in more detail in [Chapter 3: Animation Editor, Section 3.8 – Rename animations](#).

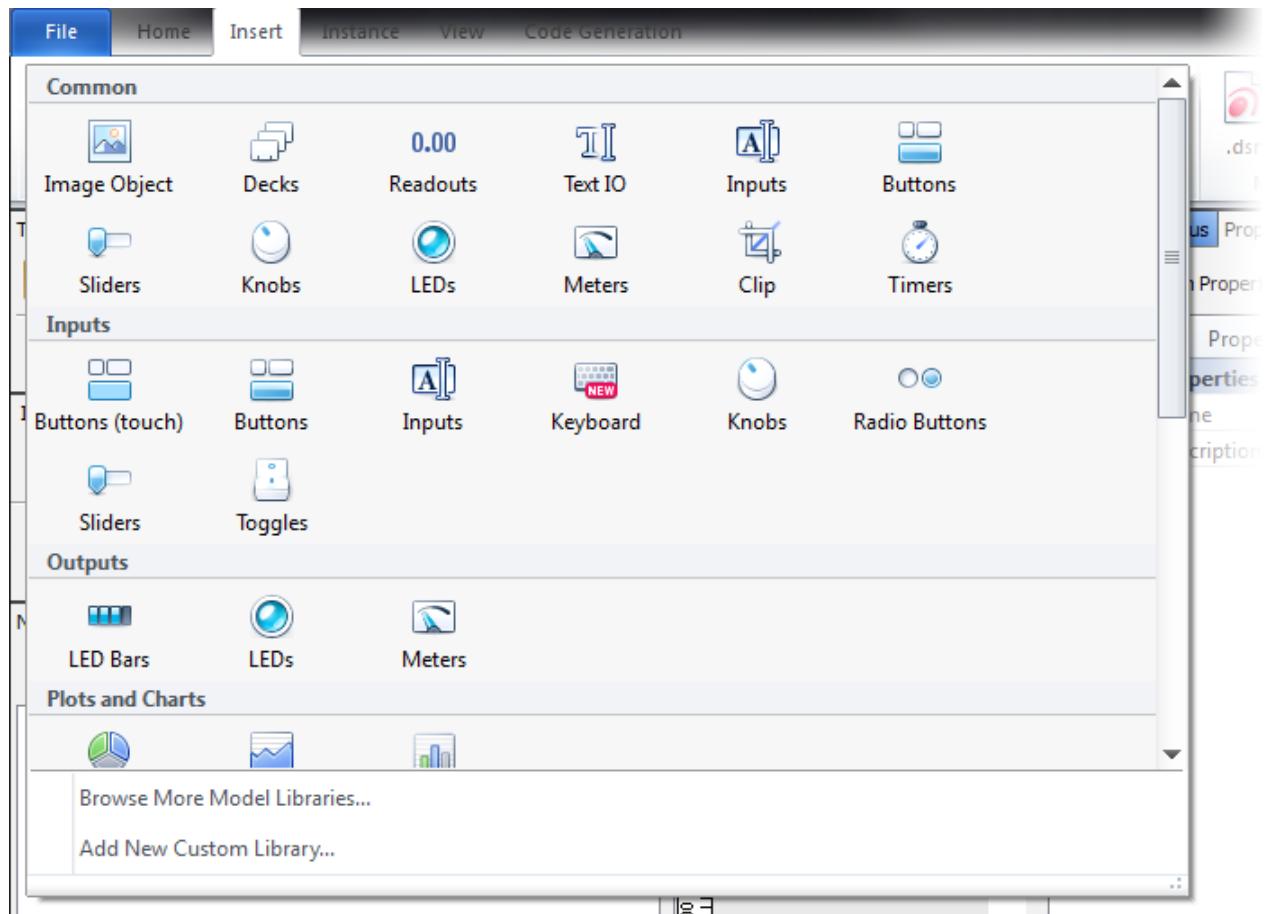
## 1.3 Insert Ribbon

The Insert ribbon in allows you to import Model Library objects, complete models, or images into your Altia model.

### 1.3.1 Model Libraries



The Model Libraries gallery allows you to easily import objects from any of the standard Altia model libraries as well as any user-created model libraries. Select a model library to import from from the Model Libraries gallery dropdown.

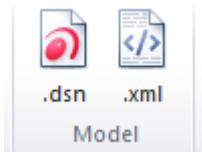


Choose **Browse More Model Libraries...** to navigate to and open any .dsn file as a library. If you find yourself frequently using a user-created .dsn as a library, choose **Add New Custom Library...** to add it to

the Model Libraries gallery dropdown. If you wish to remove a custom library added this way, right click on the library within the dropdown and choose **Remove Model from Custom Model List**.

For more information on importing and using objects in the standard Altia Model Libraries, see [Chapter 10: Model Libraries](#).

## 1.3.2 Models



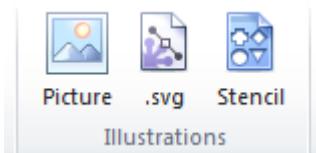
### Insert .dsn

Select this option to merge an Altia Design file with the design currently open (without removing the current design). Objects from an imported design will be placed in the work area on top of any existing objects and will appear at the coordinates they occupied in the original design. All animation names in the imported model are unchanged after importing (unlike automatically prepended numeric prefixes when importing an object from a model library).

### Insert .xml

Select this option to import objects described in an Altia Design formatted XML file. Objects in the imported XML file are added to any objects that currently exist in the editor. In most cases, the XML file is generated from Altia Design (See [Chapter 1: XML / Documentation](#)) and completely describes the behaviors and layout of the objects so that they can be re-created.

## 1.3.3 Illustrations



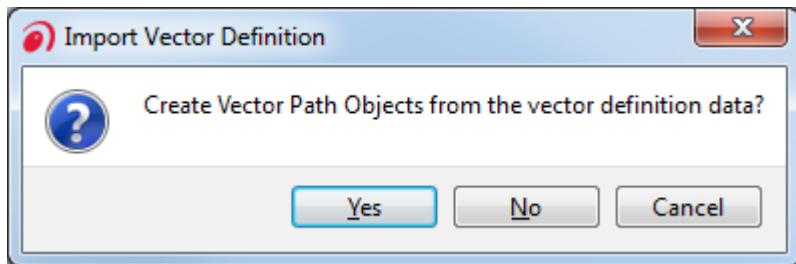
### Insert Picture

Select this option to import a color BMP, JPG, or PNG file. Imported pictures' raster data is saved directly in the design file. If your project requires the ability to dynamically change images or you would prefer to keep the model file size smaller you can use an Image Object from the Image Object model library. See [Chapter 10: Model Libraries, Section 10.5.4: Image Object](#) for more information.

## Insert .svg

Select this option to import a Scalable Vector Graphics (SVG) file. Objects in the imported SVG file are added to any objects that currently exist in the editor. The supported SVG elements are: group, rectangle, ellipse, circle, line, text, image, polygon, polyline, path, and clip-path. The supported attributes are: fill, stroke, stroke-width, alpha, font-family, font-size, font-weight, and font-style.

After selecting an SVG file to import, you will be presented with the option to import the objects in the SVG as Vector Path Objects in Altia.



Choosing **Yes** will convert all imported objects to Vector Path Objects. The Vector Path object is an Altia graphic object capable of rendering advanced vector shapes using a set of commands and their associated parameters. It allows the specification of paint definitions such as color, line characteristics, and gradients to outline and fill the path.

Some important considerations about converting to Vector Path objects include:

- If text is present, it is converted to paths during an export from Adobe Illustrator®. If you don't convert text to paths in Illustrator, it gets pulled in as Altia text, and the appearance will likely differ from the original. Appearance can be most closely matched by converting text to paths. This is easily done using the **Type > Create Outlines (Shift+Ctrl+O)** command in Illustrator.
- Vector Path objects cannot be created in Altia Design from scratch; they must be created in Illustrator and imported as an SVG file. In addition, drawings not originally created in Illustrator may be unable to render correctly even if later exported from Illustrator.
- The Vector Path object provides enhanced SVG “path” element import including Bezier curve support, SVG line gradient and radial gradient element imports. In addition, translation, scaling and rotation are supported and can be animated. However, vector and image fills are not currently supported in this release. The code generation target for Vector Path objects is only OpenVG targets.
- Paint characteristics cannot be changed once it is imported, but animation in the form of scaling and rotation is supported.
- Code generation is limited to OpenVG targets.
- The PowerVR Windows emulation is known to have performance issues.

**NOTE:** SVG construction and interpretation can vary widely. Getting two different programs to write and reproduce identical SVG content can be difficult. An SVG drawing created in one application will often render differently in any other application. For the latest on how best to work with SVG and the Vector Path object, please visit the Altia Design section of the Altia forums at: [www.altia.com/forums](http://www.altia.com/forums).

Choosing **No** will import all objects as standard Altia vector objects. Some formatting may be lost.

## Insert Stencil

Select this option to import a 1-bit (monochrome) BMP file as a stencil. A stencil in Altia Design is a special image type that allows you to change the image's colors by setting the *Outline* color in Altia. Pixels that are white in the imported monochrome image become transparent, and pixels that are black become the currently selected *Outline* color.

When a full color BMP file is imported as a monochrome bitmap, the low intensity colors (those closest to black) are rendered in the currently selected foreground color and high intensity colors (those closest to white) become transparent.

## 1.4 Instance Ribbon

### 1.4.1 Instance Ribbon

The **Instance** ribbon provides the special operations that are allowed on Instances. These include:

- **Update Instance**
- **View Base Object**
- **Change Base Object**
- **Un-Instance**
- **Edit Base Object Paths**

### 1.4.2 Import Changes from Base Object

After a Base Object has been edited in its own design file and is ready for importing into a design file with Instances, open the design file containing Instances into Altia Design. Select at least one of the Instances of the Base Object or select an object containing an Instance or Instances. Click on the **Instance** ribbon, and choose **Update Instance**. This will update the Base Object Local Copy and all Instances to the most recent version of the Base Object saved to its design file.

If an internal change is required for an Instance, Altia recommends changing the Base Object and then import the changes so Instances update to the new Base Object. If the change should only apply to the exact Instance and not any other Instances and not the Base Object, an un-instance operation could be the solution. This operation “unlinks” the Instance from its Base Object and the Instance is just like any regular object. An un-instance operation is performed from the **Un-Instance** option in the Altia Design **Instance** ribbon.

If an Instance is transformed via a scale, rotate and/or distort operation, that transform on the Instance overrides any transform changes done to the Base Object. If the Instance has only been moved, then any transform (scale, rotate, and/or distort) changes done to the Base Class will be reflected in the Instance the next time Base Class changes are imported into the design file containing Instances

Any connections that are linked to/from an Instance from/to another object remain connected when a new version of its Base Object is imported. If connections are removed or added for the Base Object, the changes are reflected in all Instances when the new Base Object is imported. Even if an Instance’s connection has a link, the connection is removed if that connection has been removed for the Base Object being imported. Similarly, if a new version of a Base Object has a connection added, the Instances will have this new connection when the new version of the Base Object is imported.

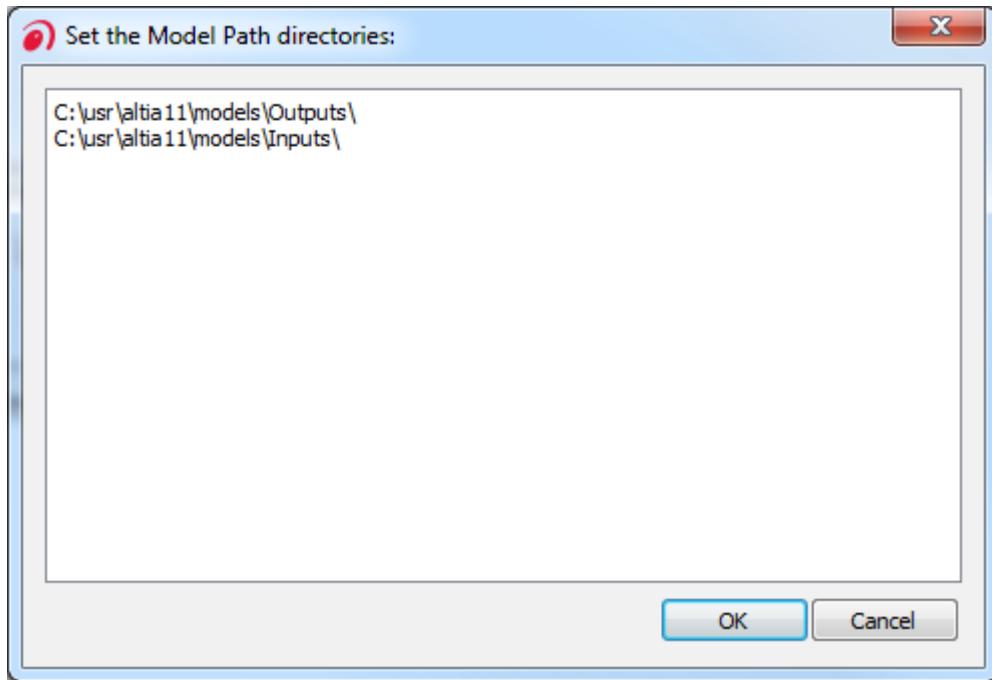
Any properties that have been set by the user on an Instance remain in effect even after a new version of the Instance's Base Object is imported. In other words, properties set by the user override Base Object property settings and properties never set by the user take their settings from the Base Object even after an import of a new version of the Base Object. If a new version of a Base Object has a property removed, that property is removed from Instances when the new version of the Base Object is imported even if the property was set by the user. Similarly, if a new version of a Base Object has a property added, the Instances will have this new property when the new version of the Base Object is imported.

Since focusing into an Instance is not enabled, be sure that any properties or connections for the Base Object are at its top level.

### 1.4.3 Edit Base Object Paths

This command opens a dialog displaying a list of directories in an edit field, each one a separate line. This type of list is referred to as a "path" in desktop computing environments. For this usage, it is called the "Model Path." The user can then edit the Model Path or just examine it. The Model Path is a list of directories where Base Objects can be found for a given Altia Design software installation. It is saved in a separate location from the design file and applies to all design files opened from that installation of Altia Design.

When an Instance is created, it refers to a Base Object Local Copy in the design file. This Base Object Local Copy holds the file name from which the original Base Object came as well as the object ID number of the Base Object in the file. Since design files can be on different machines and the location of these Base Object design files can vary on these different machines, fully qualified path names for finding these files are not used. This is indicated by the object identification shown in Altia Design when an Instance is the selected object. It reads something like "Inst of id# 190 in buttons.dsn, id# 11" which says it is an Instance of the object ID 190 in buttons.dsn (and the Instance itself has object ID 11 in the current design file). Notice there is no path given for the design file containing the Base Object, just the name of the design file. The design file name is used in conjunction with the Model Path to locate a Base Object's design file such as on a Base Object import operation. Whenever an Instance is dragged from a Models Library window into the Editor's work area, the directory of the Models Library window is added to the Model Path if that directory is not already in the Model Path.



The Model Path is the list of directories to look for design files containing Base Objects with each directory separated by a semi-colon (;). Order of the directories is important. Altia Design will search the directory list in order (left to right) looking for the first occurrence of a design file with the correct file name that contains the correct object ID. This allows the user to “overload” a particular Base Object design file with a different version by putting that directory containing the different version earlier in the Model Path. For example, if the user created an Instance of a Base Object from the standard Altia software installation Meters library and would like to change the Base Object without changing the original Altia Meters.dsn file, the user could make a copy of Meters.dsn, change the Base Object meter, and save the modified Meters.dsn file using the same name, but in a new directory. The user could then add that new directory to the Model Path and place it before the Altia software installation models directory. When an **Update Instance** is performed in the future, it would find the new Base Object first and update Instances with that new Base Object. Notice that the user should work from a copy of the original Meters.dsn and modify the exact meter with the required object ID number to create the new Base Object because the object ID number must remain the same. Otherwise, the new Base Class Object is not found during an import of changes.

#### 1.4.4 View Base Object

This command will open a Models Library window for the design file containing the Base Object and selects that object in the Models window. If the Base Object cannot be found because the object was deleted, or the design file containing the Base Object is not in the Model Path, an error dialog is shown.

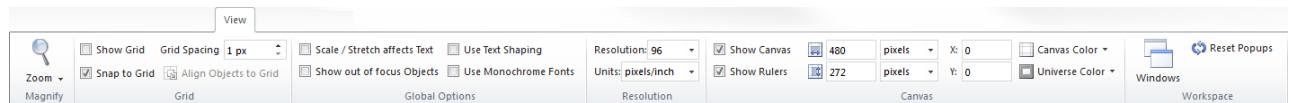
This feature is helpful if multiple users are editing and/or using the same Base Object and you are not sure if you wish to import changes from the Base Object. Altia recommends viewing the changed Base Object prior to updating to it.

#### **1.4.5    Un-Instance**

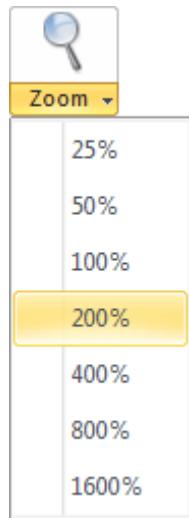
Selecting this option will take the currently selected Instance and turn it back into a regular object no longer tied to a Base Object.

## 1.5 View Ribbon

The View Ribbon interface and Universe Window lets you customize how you view and interact with models in Altia Design.



### 1.5.1 Zoom



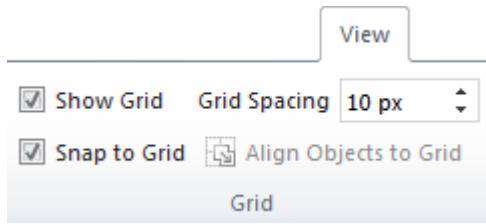
The Zoom menu provides a convenient method to change the zoom factor used to display the design model being edited in the Universe Windows (See [Section 1.6.2 - Universe](#)).

The top portion of the Zoom button is used to return the Universe view to a default Zoom factor of 100% while the bottom portion will allow you to select a value between 25% and 1600% from a drop down menu.

**NOTE:** When generating code for your design (See [Chapter 12: DeepScreen Code Generation](#)) the Zoom factor applied to the Universe will also be applied to the generated code.

Click the magnifying glass icon in the Zoom button to return to 100% zoom before generating code.

### 1.5.2 Grid

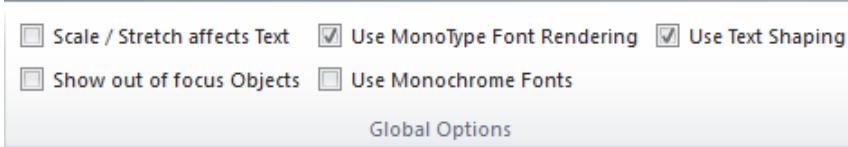


The Grid controls allow you to control the display of a grid in the Universe Window in addition to controlling alignment of objects. The available features are:

- **Show Grid:** When checked, this option will display a grid of dots in the Universe Window with the spacing specified in the **Grid Spacing** field.
- **Snap to Grid:** When enabled this option will force all objects to be positioned on the grid. This applies to all objects created while **Snap to Grid** is enabled as well to objects that are being moved.
- **Grid Spacing:** This editable field controls the spacing between the grid of dots drawn in the Universe Window when **Show Grid** is enabled. Objects previously snapped to the grid will not move when this value is edited and can be re-snapped by moving them with **Select Tool**, the arrow keys, the **Precise Move** dialog (Ctrl+M), or the X/Y value fields in the **Home Ribbon**. Objects can also be (re)snapped to the grid using the **Align Object to Grid** feature detailed below.
- **Align Objects to Grid (Ctrl + Q):** This action becomes available when one or more objects are selected in the Universe Window. When clicked the selected object(s) will snap their top-left corners to the nearest grid axes.

**NOTE:** When an object is snapped to the grid it does so based on the position of its top-left corner, e.g. the object moves such that its top-left corner moves to the nearest grid axes

### 1.5.3 Global Options



**NOTE:** The Global Options are saved with a design (.dsn) in the associated .rtm file. Each design can have different settings.

The Global Options give you the ability to customize the appearance of text and how the object hierarchy is displayed in the Universe Window. The available features are:

- **Scale/Stretch Affects Text:** When unchecked, this option prevents text from being scaled if the text object (or a group containing the text object) is scaled with the **Scale Tool**, the **Precise Scale** dialog, or the **Select Tool** via the object's resize handles. However, if the text object or its containing group is transformed using the **Distort Tool** then that transform will apply regardless of the **Scale / Stretch affects Text** option.
- **Show out of focus Objects:** This option controls what is visible in the Universe Window when focused into a Design's object hierarchy. When this option is unchecked an object's parent will not be visible nor selectable via the **Select Tool** in the Universe Window when focused in on said object.
- **Use Monochrome Fonts:** This option controls the depth of the character images created by the font engine (either MonoType or FreeType). When this option is checked, the characters will be one bit per pixel with no edge blending. When this option is unchecked, the font characters will be eight bits per pixel (grayscale) with edge blending.

**NOTE:** The Monochrome Fonts feature will be used when generating code to set the depth of the generated font characters. Previously, this option was set in the Code Generation Dialog when generating code. Starting with Altia Design 11.1, this option is determined by the Monochrome Fonts checkbox on the View Ribbon.

- **Use Text Shaping:** This option enables Text Shaping on all text (static labels, TextIO objects, and Multi-Line Text objects). Shaping is used to convert "logical" text into "displayable" text. Logical text is always ordered left to right with raw character codes. Displayable text will be arranged in a language specific manner (example: right to left for Arabic). In addition, glyph substitution will occur in order to create the proper appearance of adjacent characters in the text (example: Lam-Alef combination in Arabic). Mixed language text may have portions ordered right-to-left, and other portions ordered left-to-right.

If you are not using a language in your design that requires Text Shaping (such as Arabic, etc.) you do not normally need to enable Text Shaping.

**NOTE:** The Text Shaping feature is not supported on Altia DeepScreen Targets version 5.x or lower. You will not be able to generate code using those targets when the Text Shaping option is checked on the View Ribbon.

- **Use Monotype® Font Rendering:** This option enables use of the iType® Font Engine which support Monotype® fonts. When unchecked, the FreeType Font Engine will be used.

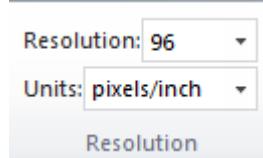
**NOTE:** The Use Monotype Font Rendering setting will be used when generating code to set the font rendering technology.

Starting with Altia Design 11.1, this option is determined by the Monotype® checkbox on the View Ribbon. In prior versions of Altia Design this option was set in the Code Generation Dialog when generating code.

**NOTE:** A license is required to use the MonoType® feature. This feature will be disabled without the proper license.

For more information or to request a MonoType license, go to <http://www.altia.com/monotype> or email your request to [info@altia.com](mailto:info@altia.com).

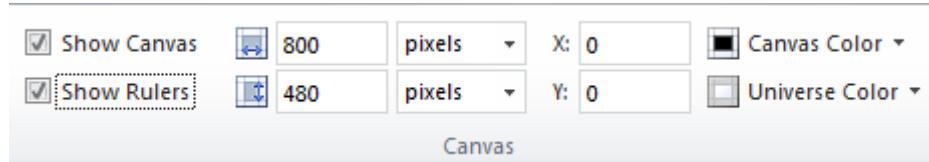
## 1.5.4 Resolution



The **Resolution** options allow you to control the pixel density which primarily affects the rendering of text. The default is configured for a display with a pixel density of 96 pixels/inch, the standard Microsoft Windows DPI setting.

You can manually enter new values into the **Resolution** field and you can also click the drop down arrow to select from other standard values (72 DPI and 133 DPI). Values you enter also become available in the drop down list for quick recall. You can also specify the **Units**, with options for pixels per inch and pixels per centimeter being available.

## 1.5.5 Canvas



Altia Design 11 introduces the concept of a **Canvas** and **Universe** to help you separate assets like Control Code from the actual content that will be rendered by your target. Objects placed within the **Canvas** area will be rendered and displayed while objects not within the **Canvas** will not be visible. The **Canvas** controls allow you to customize the **Canvas** and **Universe** area for your target and per your preferences.

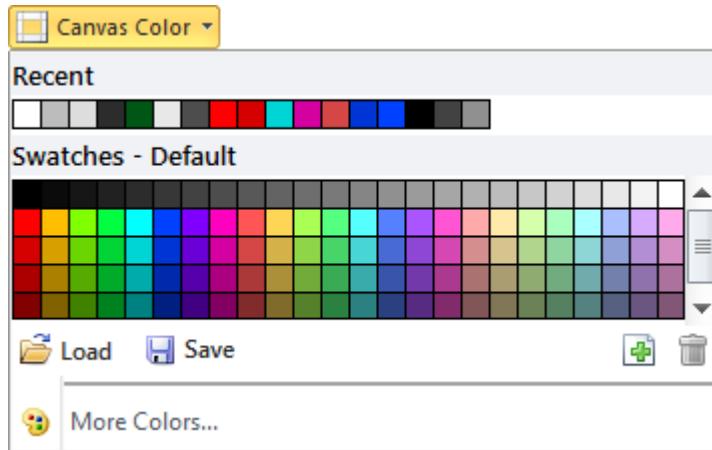
The available options include:

- **Show Canvas:** Enables and disables the **Canvas** area display
- **Show Rulers:** Enables and disables the display of the **Rulers** around the **Universe Window**
- **Width( ):** Sets the width of the **Canvas** area.
- **Height( ):** Set the height of the **Canvas** area.

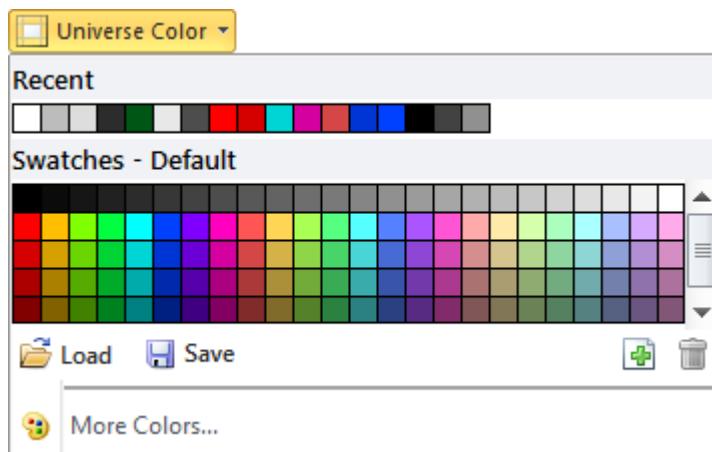
**NOTE:** Setting the width and height of the **Canvas** equal to that of your target's display is a useful way to visualize how your design will look on target. For units you can select Pixels, Inches, Centimeters, or Millimeters. For units that are not Pixels the effective pixel size will be calculated based on the current **Resolution** value.

- **X (origin):** Positions the **Canvas** within the **Universe Window**. The X origin is the left most edge of the **Canvas**.

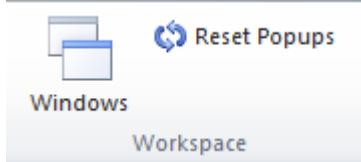
- **Y (origin):** Positions the Canvas within the Universe Window. The Y origin is the bottom most edge of the Canvas.
- **Canvas Color:** Clicking this button will display the Color Palette interface so you can choose a color for the Canvas area:



- **Universe Color:** Clicking this button will display the Color Palette interface so you can choose a color for the Universe area:



## 1.5.6 Workspace

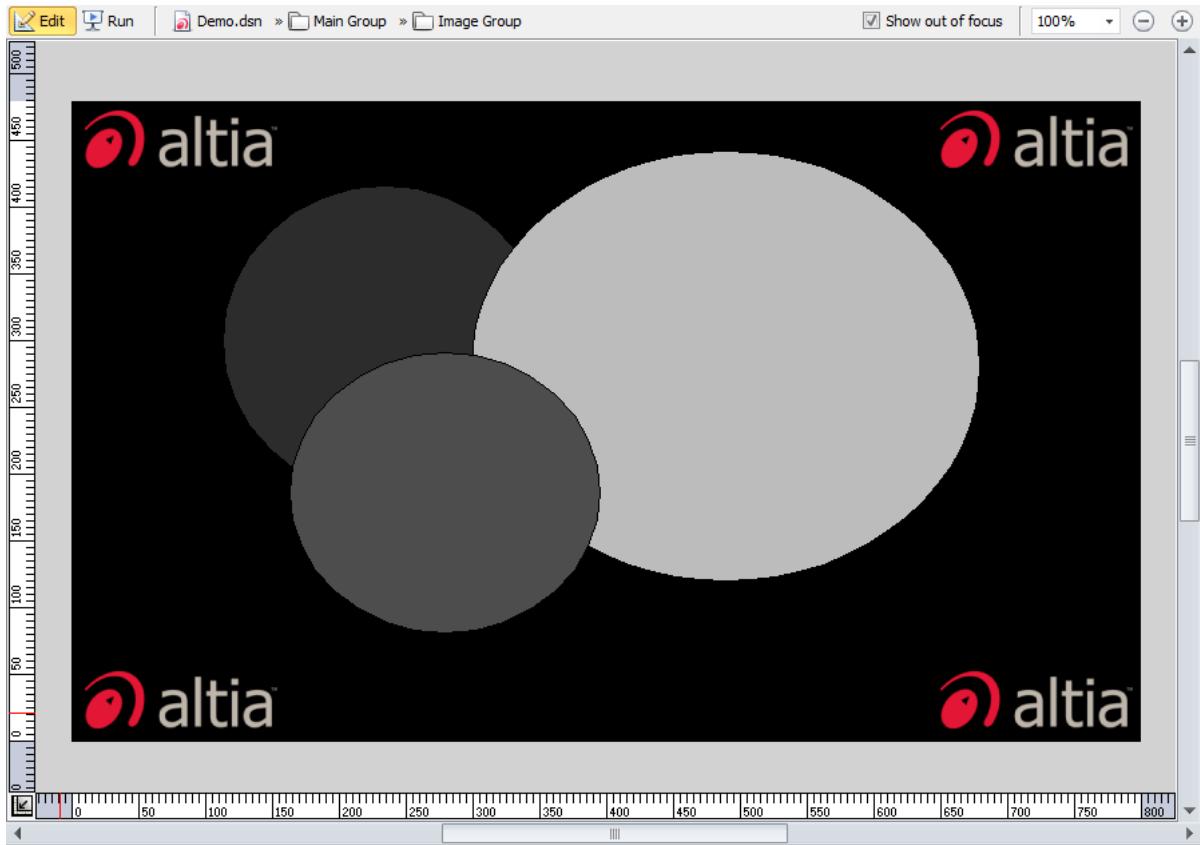


The **Workspace** controls let you control the visible work panes and the state of warning and informational dialog popups. When you click on the **Windows** button a list of all work panes will be displayed.

- The **Windows** drop down shows which work panes are currently visible via the check marks: . Clicking on a checked work pane will hide it until it is checked again.
- The **Reset Popups** button lets you re-enable any popup message dialogs that have been disabled by selecting the "*Don't show this message again*" checkbox.

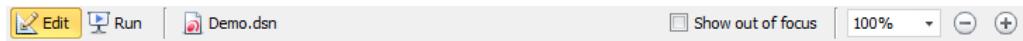
## 1.6 Universe View

The Universe Window, shown below, is the primary means of viewing and editing Altia Design models.

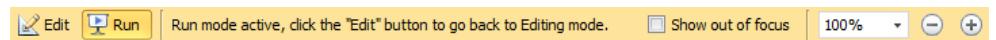


### 1.6.1 Control Bar

The Control Bar at the top of the Universe Window provides a number of controls for managing state and navigation.



- **Edit:** This button puts the design into **Edit Mode**, allowing you to add, remove, delete, move, transform, and generally manipulate the objects visible in the **Universe Window**.
- **Run:** This button puts the design into **Run Mode** which lets you run and interact with your design as an end user would on target. While in Run Mode it is not possible edit the design, the Control Bar will change its appearance while in Run Mode to indicate editing isn't possible:



- **Breadcrumbs:** As you navigate in your design and focus in on objects and groups the **Breadcrumbs** interface will display your current "path" and allow you to go back to a previous

location by simply clicking on a listed object. For example, the **Control Bar** below shows that we're currently focused into a pair of nested groups:

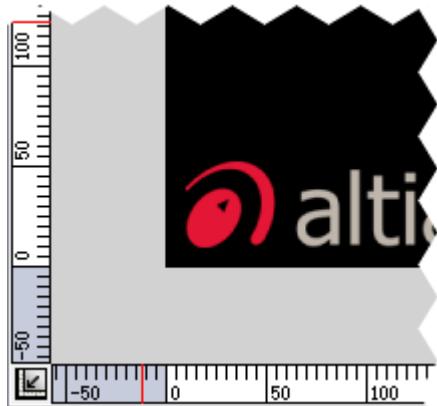


If you wanted to go back to the top level of the design in this example, click *Demo.dsn*. Similarly if you want to go up just a single level, click *Main Group*. This feature makes navigating complicated designs much easier.

- **Show out of focus:** This option is identical to the Show out of focus Objects in [Section 1.5.3 - Global Options](#).
- **Zoom:** The **Control Bar** Zoom controls ( ) give you the ability to change the **Universe Window's** Zoom factor with either a drop down list of options or the +/- buttons. Available Zoom factors are 25%, 50%, 100%, 200%, 400%, 800%, and 1600%. See [Section 1.5.1 - Zoom](#) for more details about Zoom factor and Altia Design models.

## 1.6.2 Universe

The **Universe Window** displays the Universe, the Canvas and, of course, the model you're designing. The **Universe Window** has a number of features beyond the **Control Bar** which are detailed below.



Along the sides of the **Universe Window** are displayed the graduated Rulers (assuming the **Show Rulers** option is enabled, see [Section 1.5.5 - Canvas](#)). The white sections of the Rulers indicate pixels that fall within the Canvas area that will be displayed on target while the gray sections indicate non-Canvas area that will not be displayed on target. The Rulers also indicate the current mouse position with a red line that corresponds to the mouse pixel coordinates within the Universe.

Finally, at the bottom right of the Universe Window is the centering button ( ). When clicked, this button will place the Canvas area at the center of the Universe Window.

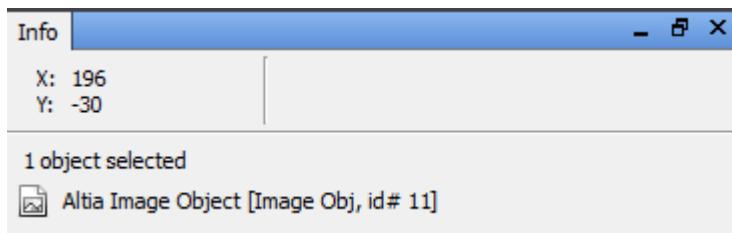
# 1.7 Info and Tool Panes

The Information and Tool Panes provide easy access to the Altia Design Editor drawing tools and drawing tool status.

If the Info pane or the Tool pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Info** or **Tool** menu options are checked.

## 1.7.1 Info Pane

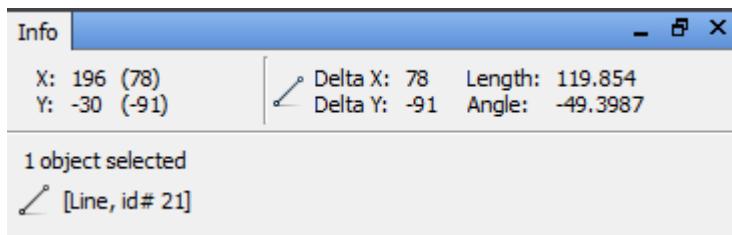
The **Information** pane provides status on the current draw operation. The draw operation is specified by the active tool in the **Tool** pane. The following sections detail the different draw operations and how each is presented in the **Information** pane.



In every draw operation, the current selection is displayed in the bottom of the **Information** pane. The status will show the number of object selected (if any). For single object selections, additional information about the selected object will be displayed including the unique Object ID.

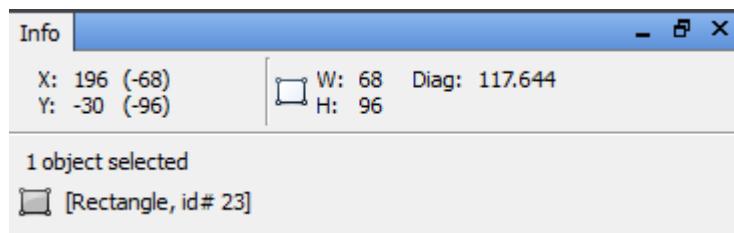
The current cursor position (in canvas pixel coordinates) is always displayed in the top left of the **Information** pane.

### Line Status



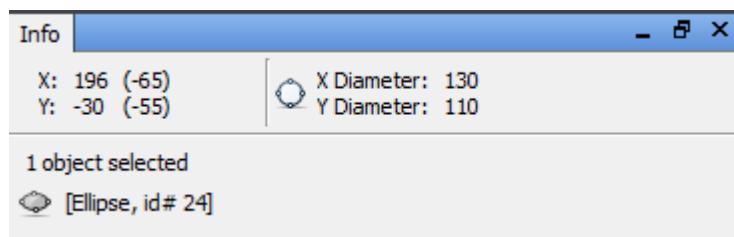
When drawing lines, the **Information** will show data on the current line segment. This includes the start coordinate on the left along with the deltas from the start coordinate (in parenthesis). The right side will show data on the line vertical size, horizontal size, total length, and angle from zero (horizontal at 3 o'clock position).

## Rectangle Status



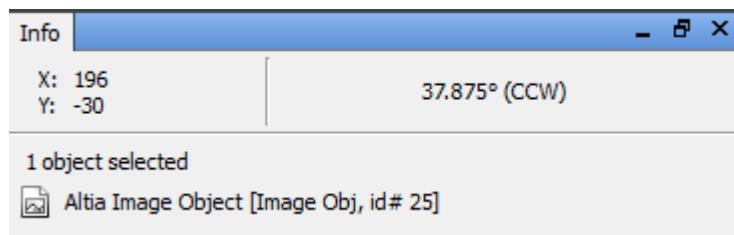
When drawing rectangles, the **Information** will show data on the object. This includes the start coordinate on the left along with the deltas from the start coordinate (in parenthesis). The right side will show the width, height, and diagonal size of the rectangle (in pixels).

## Ellipse Status



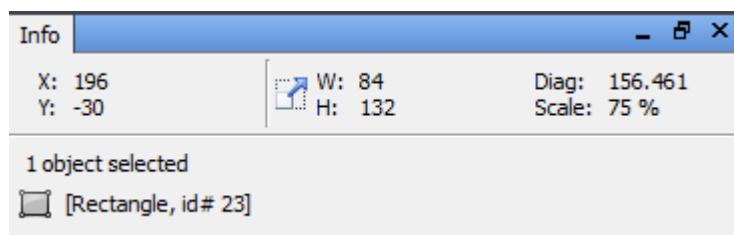
When drawing ellipses, the **Information** will show data on the object. This includes the start coordinate on the left along with the deltas from the start coordinate (in parenthesis). The right side will show the horizontal and vertical diameter of the ellipse (in pixels).

## Rotate Status



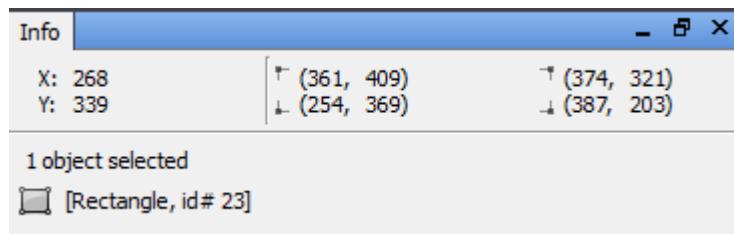
When rotating objects, the **Information** will show data on the operation. The right side will show the rotation angle delta (in degrees) from the objects original position.

## Scale Status



When scaling objects, the **Information** will show data on the operation. The right side will show the new size of the object (width, height, and diagonal) along with the scaling factor in percent.

## Distort Status



When distorting an object, the **Information** will show data on the operation. The right side will show the four corners of the distorted quadrangle (in canvas pixel coordinates).

## 1.7.2 Tool Pane

The **Tool** pane provides controls to select different draw tools. The following sections detail the different draw tools and how each is presented in the **Tool** pane.

**NOTE:** When activating a tool, it will remain activated until another tool is selected. This behavior can be changed by modifying the altia.ini file located in the "defaults" folder where Altia Design is installed. Find the line "Altia\*keepTool" and change the value from "On" to "Off".

**NOTE:** Each tool can be activated using a keyboard shortcut (specified in the title of each tool section below).

### Selection Tool (V or M)

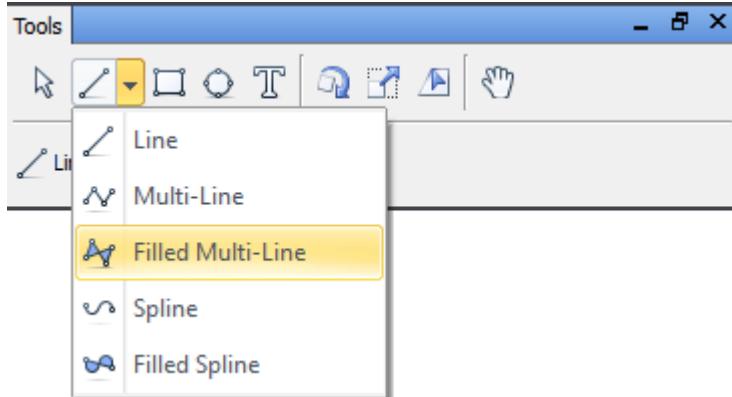


The selection tool allows the User to select objects on the canvas using left-click and drag operations with the mouse. Two buttons are provided in the bottom of the **Tool** pane when using this mode:

- **Select All** - Selects all objects at the current focus level.
- **Deselect** - Deselects all objects (i.e. no selection).

Pressing and holding CTRL while selecting will delta the new objects from the current selection. If the new objects are not in the selection, they'll be added. If the new objects are already in the selection, they'll be removed.

### Line Tool (\)



The line tool allows the User to draw one of five different line types. The line type can be selected using the drop down menu. Selecting a new line type will change the icon for the line tool button. Clicking the button will use the currently active line type (as shown on the icon). Pressing and holding SHIFT while drawing will constrain the line segment to 45-degree angles.

### Rectangle Tool (Q)



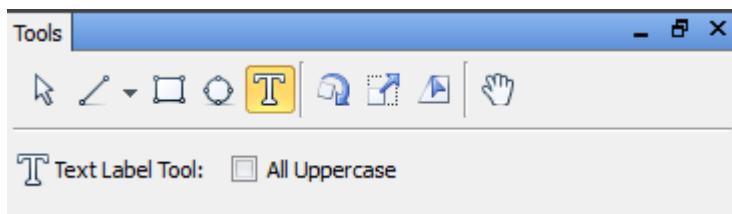
The rectangle tool allows the User to draw a rectangle. Pressing and holding SHIFT while drawing the rectangle will constrain the width and height to a square (width == height).

### Ellipse Tool (L)



The rectangle tool allows the User to draw an ellipse. Pressing and holding SHIFT while drawing the ellipse will constrain the width and height to a circle (width == height).

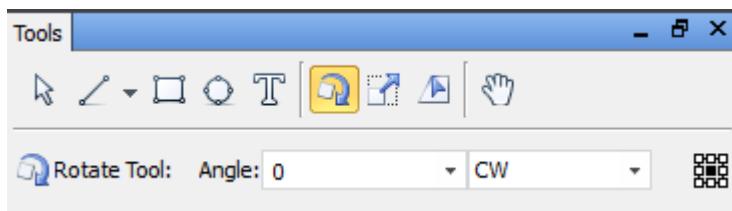
## Text Tool (S)



The text tool allows the User to create static text objects (also called Label Objects). The Label Objects cannot be changed at runtime to show a different text string. A checkbox is shown in the bottom of the **Tool** pane when using this mode:

- **All Uppercase** - Forces all letters entered to be uppercase.

## Rotate Tool (R)



The rotate tool allows the User to rotate objects. The following controls are provided in the bottom of the **Tool** pane when using this mode:

- **Angle** - Allows the entry of an exact rotation angle to precisely rotate the object. The angle can be fractional.
- **Direction** - Specifies a rotation direction (clockwise or counter-clockwise).

Pressing and holding CTRL before performing a mouse rotation on the canvas will set the rotation point. The default rotation point is the center of the selected object.

## Scale Tool (B)



The scale tool allows the User to scale objects. The following controls are provided in the bottom of the **Tool** pane when using this mode:

- **Width** - Specifies new width dimension. This value can be fractional.
- **Height** - Specifies new height dimension. This value can be fractional.

- **Units** - Specifies dimensional units of percent, pixels, inches, or centimeters.

### **Distort Tool (D)**



The distort tool allows the User to apply perspective transformations on objects. Each corner of the bounding box for an object can be left-clicked-and-dragged to a new location on the canvas. Right click to complete the distort operation.

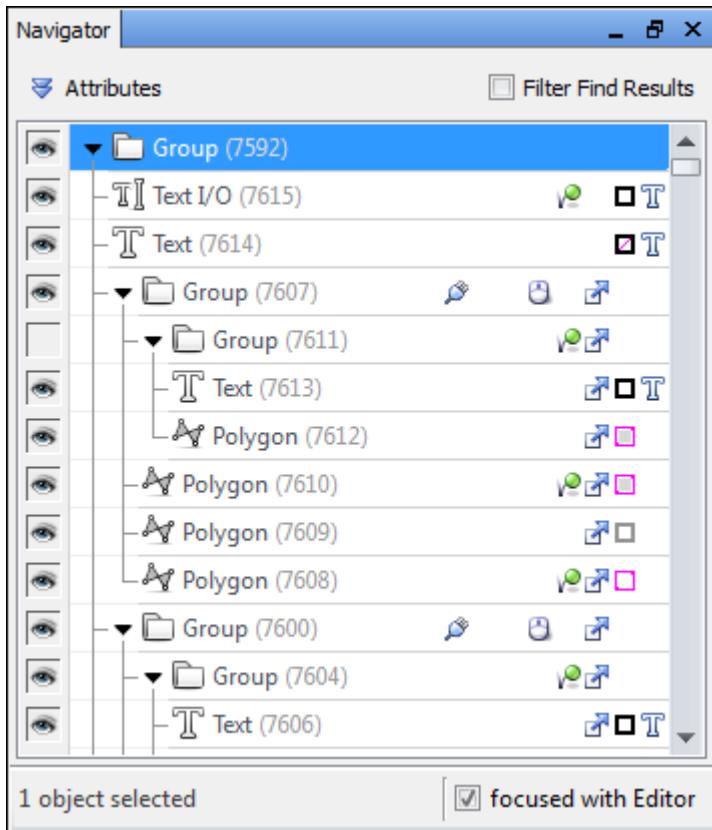
### **Pan Tool (SPACE)**



The pan tool allows the User to move the canvas in the view window. Left-click-and-drag to slide the canvas around in the view.

# Chapter 2: Navigator

The Navigator pane displays your design's graphic objects in a hierarchical tree. For each object, it displays useful information for quick assessment at a glance. It's also useful in changing attributes of selected objects in bulk.



## 2.1 The Object Hierarchy

Each object in your design can be found in the object hierarchy tree. The objects are sorted according to their position in the Z-order where they are drawn from bottom to top within a focus level. Objects that are lower in the Z-order may be obscured by objects at higher level when rendered in the Graphic Editor.

## Show / Hide

An eye icon, , indicates whether the object is currently showing or hidden. Click on this icon to toggle it.

## Hierarchy Tree

Objects such as Groups or Decks that contain other objects can be expanded to show their child objects or collapsed to hide them. Click on the expand or collapse arrow to display or hide the child objects.

Objects that are otherwise hidden from view, such as cards in a Deck that are not currently showing, are also listed in the tree. Selecting one (while **focused with Editor** is checked) will make it the showing card.

Connecting lines serve to identify an object's relationship with another object as a parent, child, or sibling. These lines can be toggled on or off using the "**Show hierarchy tree**" checkbox item in the context menu.

Instance objects will not show their child objects and cannot be expanded.

## Object Icon

An object type icon is displayed for each object so that its type can be easily identified. It can be toggled on or off using the "**Show object icon**" checkbox item in the context menu.

## Name / Type / Object Id

If an Object Name property is defined for an object, this name will be displayed. Otherwise, the object's type is displayed. Double click on the name to change it. This is a good way to give your objects more descriptive names. The unique object id is displayed inside parentheses.

Instance objects will show their base object id and the design file they came from.

## Informational Icons

Useful information about each object is depicted as icons.

-  - This icon indicates that the object is an instance.
-  - This icon indicates that the object has defined connections. Press the icon to display the Connection Editor pane.
-  - This icon indicates that the object has defined properties. Press the icon to display the Property Editor pane.
-  - This icon indicates that the object has defined control code. Press the icon to display the Control Code Editor pane.
-  - This icon indicates that the object has defined stimulus. Press the icon to display the Stimulus Editor pane.

- - This icon indicates that the object has defined animations. Press the icon to display the Animation Editor pane.
  - - This icon indicates that the object has been scaled, rotated, or distorted.
  - - This icon indicates that the object has fill and outline definitions. The colors of the icon will match what is defined for the fill and outline colors.
- This icon indicates that the object has neither fill nor outline but colors are still defined on the object. This state is achieved by selecting the "**No Fill**" menu item of the **Fill** menu button and the "**No Outline**" menu item of the **Outline** menu button in the **Home** ribbon.

**NOTE:** The icons shown are extreme examples when both fill and outline are defined. You may expect to see a mix of the two examples when either fill or outline is defined.

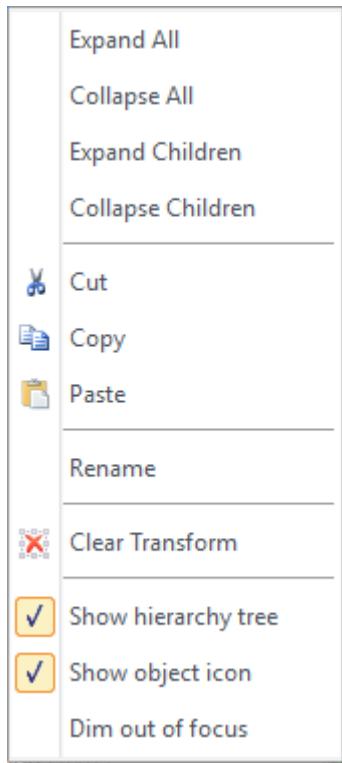
- When the icon is not present, the object does not have fill, outline, or colors defined. To achieve this state, press the **Clear Group Style** button in the **Home** ribbon.

**NOTE:** There is also the possibility that in this case both the fill and outline colors are defined as white (the list item's unselected background color). To expose the icon, select the object so that the list item's background is blue.

- - This icon indicates that the object has a font definition.

## 2.1.1 Context Menu

Action items in the context menu are applied to all items selected in the object hierarchy tree.



- **Expand All** - Expand to show all objects in the hierarchy.
- **Collapse All** - Collapse to show only top level objects.
- **Expand Children** - Expand to show all child objects of the parent object under the menu. This menu item is not available if the object type is not a container.
- **Collapse Children** - Collapse to hide all child objects of the parent object under the menu. This menu item is not available if the object type is not a container.
- **Cut** - Remove the selected objects. These objects may be pasted back at a later time.
- **Copy** - Make a copy of the selected objects. These objects may be pasted back at a later time.
- **Paste** - Paste the objects in the copy buffer at the highest Z-order of the current focus level.
- **Rename** - Give the object a more descriptive name.
- **Clear Transform** - Clear any scaling, rotation, or distortion placed on the object.
- **Show hierarchy tree** - Toggles the connecting hierarchical lines between parent, child, and sibling objects.
- **Show object icon** - Toggles the object type icon.
- **Dim out of focus** - When checked, gray out the objects not in the current focus level.

## 2.2 Object Selection

The bottom right corner of the Navigator has a checkbox with the text "**focus with Editor**". This checkbox controls how the Navigator behaves with the Graphic Editor, specifically how objects are selected. There are two modes available.

### Focused with Editor (Checkbox checked)

In this mode, the Navigator is "linked" or "focused" with the Editor. The objects selected in the Navigator match the objects selected in the Graphic Editor. The two interfaces (Editor and Navigator) are working together and are synchronized. Changing the selection in the editor will change the selection in the Navigator and vice-versa. In addition, the Navigator will auto-scroll as the selection is changed in the editor. This way the selected objects are always in view in the Navigator as you work with your design in the Graphic Editor.

When linked, selecting objects in the Navigator must follow the same rules as when selecting objects in the Editor. For example, selecting objects from different parents is not allowed (since this type of selection would not be possible in the Editor as only the children of the current focus object in the Editor can be selected). If an invalid selection is made in the Navigator, the mode will automatically change to "unfocused" (see the following section for unfocused mode).

When in "focused" mode, the selected objects in the Navigator will be highlighted in a blue.

### Unfocused with Editor (Checkbox unchecked)

In this mode, the Navigator is "unlinked" or "unfocused" with the Graphic Editor. The objects selected in the Navigator can be different than the objects selected in the Editor. The two interfaces (Editor and Navigator) are working independent of each other.

This mode is useful when trying to select objects from different parents. These types of selection sets are not possible in the Editor but they are possible in the Navigator. The selected objects can then be manipulated in bulk (such as changing the font or color) using the Attributes panel.

When in this "unfocused" mode, the selected objects in the Navigator will be highlighted in a yellow.

### Special Selection Features

- Selecting a Card in a Deck Object will select all objects in the Card.
- Selecting a hidden Card or objects in a hidden Card will automatically make that Card the current card of the Deck.
- Double-clicking an object in the Navigator (not on the name) will select it and center the object in the Graphic Editor. This works in either focus mode.

## 2.3 Dragging Objects

You can drag objects from their current location to a new location in the Navigator. This is a very powerful feature that allows you to quickly manipulate your design. For example, you can move objects between cards of a deck, make new cards in a deck, or reorder cards in a deck.

To perform a drag-and-drop operation, execute the following steps:

1. Select the objects to drag in the Navigator. You can use SHIFT and CTRL to make complex selections. Use “unfocused” mode to select objects from different parents.
2. Mouse over your selection in the Navigator.
3. Press-and-hold the left mouse button.
4. Drag the mouse while holding the left-mouse button down. The mouse icon will change to one of the following:
  - a. Gripper Hand Cursor - shows that you are dragging some objects and the drop-location is valid.
  - b. Red Invalid Circle Cursor - shows that you are dragging some objects and the drop location is NOT valid.
5. A thick black line will appear in the Navigator showing the drop location for the objects. You can drop above or below an object. You can also drop into a container such as a group, deck, or card.
6. You cannot drop into any object that is in the set of objects being dragged (i.e. you cannot drag an object into itself or a child of itself).
7. Dragging above the top object in the Navigator or below the bottom object in the Navigator will cause the Navigator to auto-scroll. You can also scroll the Navigator while dragging using the mouse wheel.
8. Release the left-mouse-button to drop the objects into the new location. If the mouse cursor is currently red/invalid then the drop will be aborted.

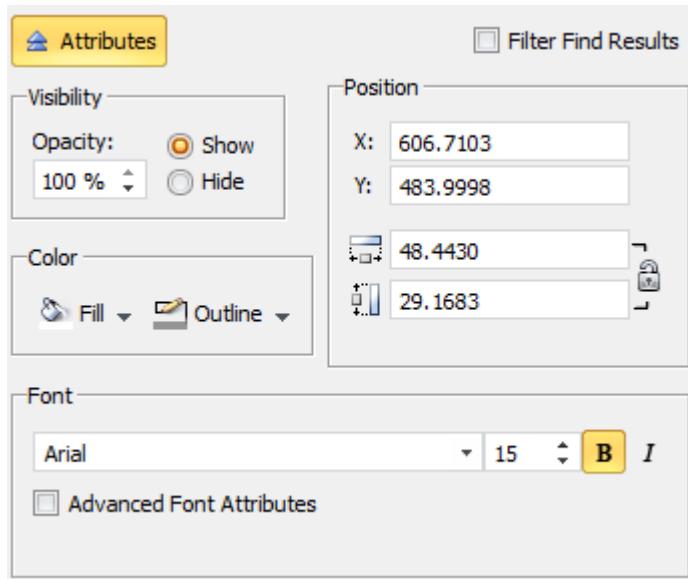
### Special Drag-and-Drop operations:

- Dropping objects into a group will move the objects to the “bottom” of the group.
- Dropping objects into a deck will move the objects into a new card with a card number that’s one larger than the last card in the deck.
- Dropping objects into a card will move the objects the objects to the “bottom” of the card.
- Dropping objects between cards will move the objects into a new card that has a number which fits between the two cards. If there is no available number between the two cards, then you’ll be asked if the Navigator can auto-renumber the cards for you. This is a very quick way to manipulate card numbers in a deck.

## 2.4 Attributes

An advantage of being able to select objects from different focus level is that the attributes of these objects can be changed all at once. This is done by using the Attributes panel. To display the panel, press the **Attributes** toggle.

The information displayed in the attributes pane reflects that of the lowest Z-order object within the set of selected objects. However with some exceptions, a change in an attribute value will be applied to all selected objects.



### Visibility

The selected objects' **Opacity** and whether it is shown (**Show**) or hidden (**Hide**) can be changed here.

### Position

Changes to values here do not behave as with the other attributes. A change in the **X** or **Y** position does not move all selected objects to the same location. Rather, it is used to calculate a positional delta of the lowest Z-order object within the set of selected objects. The delta is then applied to all selected objects which preserves their relative position to each other.

The width and height can only be changed for a single object selection.

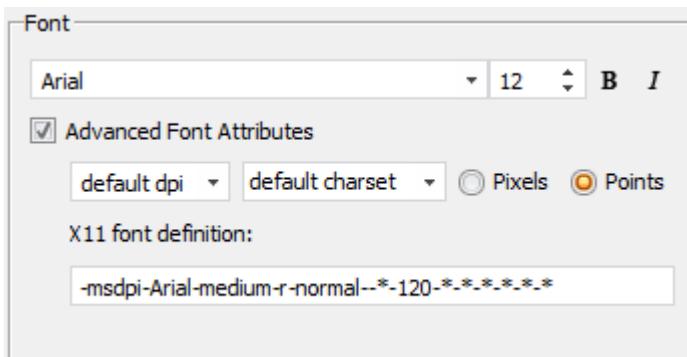
### Color

The selected objects' **Fill** and **Outline** colors and patterns can be changed here.

## Font

The selected objects' font name and size as well as whether it's bold or italicized can be changed here. More advanced font settings can be changed by checking the **Advanced Font Attributes** checkbox.

### Advanced Font Attributes



Advanced font definition such as DPI, charset, or whether to use pixel or point values are assigned for the selected objects here. X11 font definition can also be manually specified.

## 2.5 Filter Find Results

The Navigator can be put into a special mode where only the objects found in a global find operation are displayed. In this mode, all Navigation actions may be performed on the listed objects with the exception of Drag-and-Drop. This is useful when you want to temporarily work with a subset of objects.

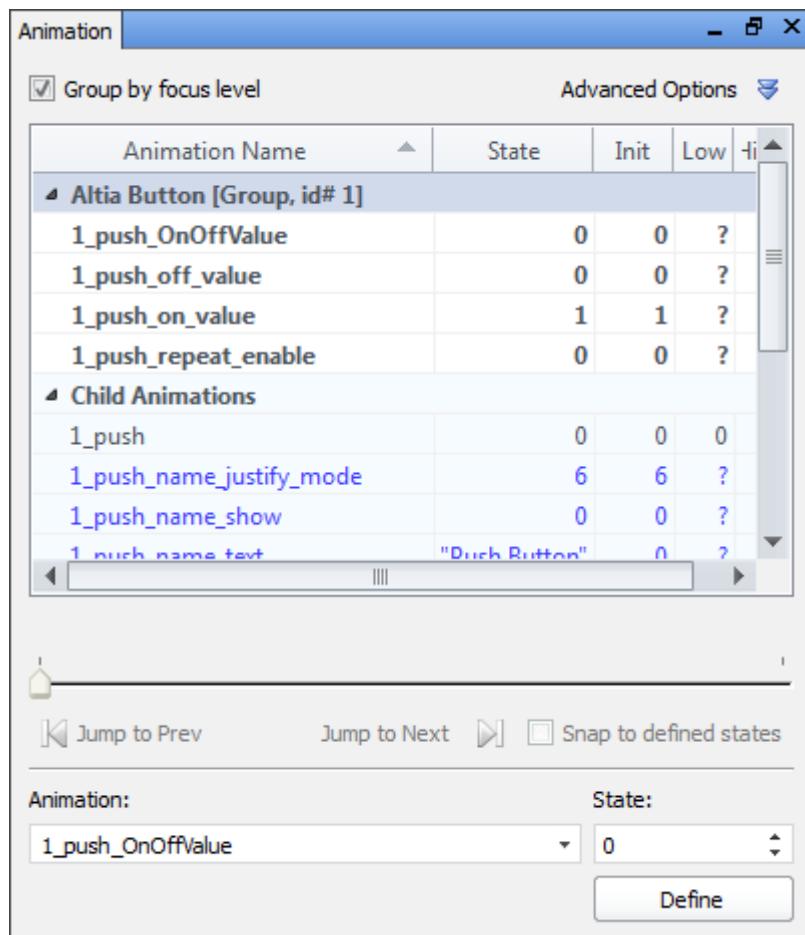
For example, if you have a design with multiple button objects that you have named "Button 1" through "Button 10". If you want to see these buttons in the Navigator all at once to check their font but it is not possible because the design is too large, then follow these simple steps:

1. Press Ctrl-F to bring up the global search dialog.
2. In the **Find Text** tab, type "Button" for the search text.
3. Uncheck **All Object Data** and check **Object Name**.
4. Press **Find Now** and wait for the search to complete.
5. In the Navigator, check **Filter Find Results**.
6. You should now see only the buttons you want to work with.

# Chapter 3: Animation Editor

Altia Design's Animation pane is used in conjunction with the Universe view. After creating objects in the Universe view, use the Animation pane to define animations for the objects.

If the Animation pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Animation** menu option is checked.



## 3.1 Animation Editor Layout

The Animation pane is divided into three sections:

- **List area**
- **Animation state Slider**
- **Name and State fields**

### 3.1.1 List area

This section displays the currently defined animation names and states for the selected graphical objects (as well as all animation states and names for all children of the selected graphical objects if the selection is a group or container object).

The screenshot shows the 'Animation' pane with a table listing animation sequences. The table has columns: Animation Name, State, Init, Low, and High. The first section, 'Altia Button [Group, id# 1]', contains four rows: '1\_push\_OnOffValue' (State 0, Init 0, Low ?, High ?), '1\_push\_off\_value' (State 0, Init 0, Low ?, High ?), '1\_push\_on\_value' (State 1, Init 1, Low ?, High ?), and '1\_push\_repeat\_enable' (State 0, Init 0, Low ?, High ?). The second section, 'Child Animations', contains four rows: '1\_push' (State 0, Init 0, Low 0, High 1), '1\_push\_name\_justify\_mode' (State 6, Init 6, Low ?, High ?), '1\_push\_name\_show' (State 0, Init 0, Low ?, High ?), and '1\_push\_name\_text' (State "Push Butt...", Init 0, Low ?, High ?). A checkbox 'Group by focus level' is checked at the top left, and 'Advanced Options' are available at the top right.

Animation Name	State	Init	Low	High
Altia Button [Group, id# 1]				
1_push_OnOffValue	0	0	?	?
1_push_off_value	0	0	?	?
1_push_on_value	1	1	?	?
1_push_repeat_enable	0	0	?	?
Child Animations				
1_push	0	0	0	1
1_push_name_justify_mode	6	6	?	?
1_push_name_show	0	0	?	?
1_push_name_text	"Push Butt...	0	?	?

Each row of this scrollable list contains information about one animation sequence. Because several objects may have sequences defined with the same name, names may appear in the list several times. In addition, there may be many more sequences listed than objects selected since objects may have multiple sequences defined for them.

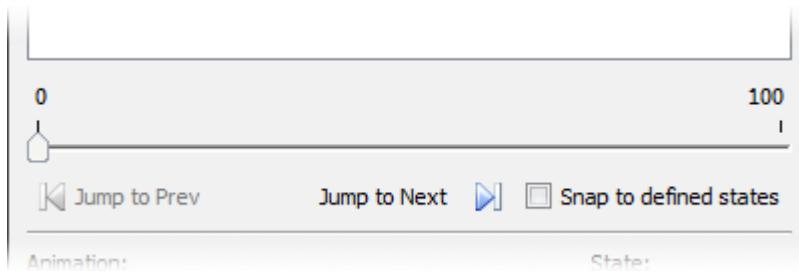
The **Low** and **High** values for a row indicate the range of state values currently defined for the sequence in the **Animation Name** column, while **State** indicates the sequence's current state. The **Init** column lists the state that will become the sequence's current state when the design is first opened.

**NOTE:** If a selected object is a group and an object embedded inside the group has defined animation states, the embedded object's animation will appear in the "Child Animations" area. Child Animation rows have a light blue background and

can be expanded/collapsed by clicking on the Child Animations expand/collapse arrow. Child Animations cannot be redefined without focusing into the group and selecting on the actual object that has the defined animation.

### 3.1.2 Animation State Slider

Use the Animation State Slider to easily change the animation state within the defined animation range. The slider moves within all of the interpolated states between each explicitly defined state.



Clicking the **Jump to Next** or **Jump to Previous** buttons will set the state of the selected animation to the next highest/lowest defined state.

When **Snap to defined states** is checked, the slider will always "snap" to defined states and will not move within the interpolated states.

### 3.1.3 Animation Name/State Fields

**Animation** name and **State** fields display the selected animation name and its current state.



An animation state can be changed either interactively (through stimulus, control, or linked code) or manually (by typing in a value or toggling the current value with the increment and decrement arrows next to the **State** field).

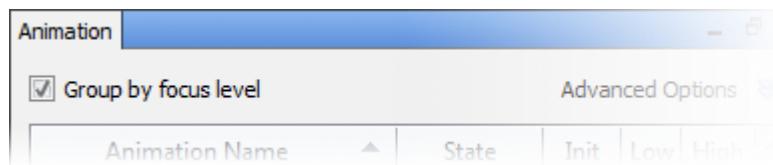
The Animation pane's **Define** button is used to save a selected object's current graphical state as an animation state. Define will save the animation name and state displayed in the Name and State fields.

## 3.2 Animation Pane Additional Functions

Several useful functions are available within the Animation pane.

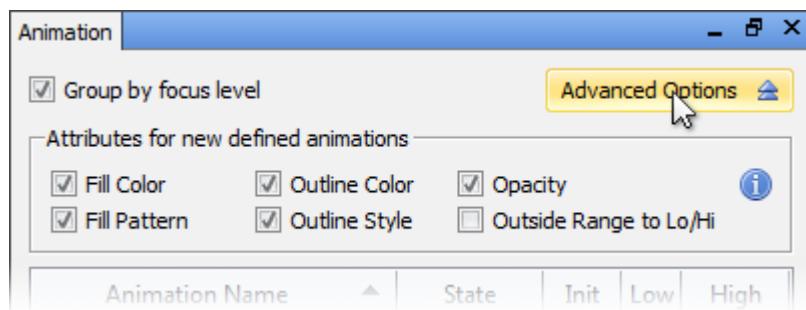
### 3.2.1 Group by Focus Level

When checked, **Group by focus level** groups any child animations and moves them to the bottom of the Animation pane's list. When unchecked, all animations (including any child animations) are listed alphabetically within the Animation pane list. Remember that many animation functions (Define, Redefine, etc.) can only be performed to animations at the current focus level. When animations are listed alphabetically, child animation rows (if any) are still colored blue for easy identification.



### 3.2.2 Advanced Options (Animation Attributes)

Click the **Advanced Options** toggle button to display the Animation Attributes.



Checking or unchecking the checkbox attributes options will only affect animation states to be defined. In other words, the current attribute settings are stored with an animation state when the state is defined with the **Define** button.

### Fill Color, Fill Pattern, Outline Color, Outline Pattern, Outline Style, and Opacity

These options allow you to choose which attributes you want to store with an animation sequence definition. When an animation state is replayed, each of these attributes will be either set as it was when the state was defined or left unchanged, depending on the selections in the Animation Attributes window.

By default, all attributes are stored with each animation definition (indicated by the checked checkboxes). In general, choose to store attributes that are inherent to the type of animation being defined.

For example, if the animation is designed to change an object's fill color, check **Fill Color** and that attribute will be saved with the definition. If the fill color is not involved in the animation, uncheck it. By unchecking attributes not directly involved in the animation, those attributes can easily be changed for the object at a later time without having to redefine all states of the animation sequence.

There are different situations where one choice is more appropriate than the other. The following are two such examples.

### Attributes Stored With Animation Example

An animation sequence is defined for a warning light. Animation function **warning** turns the light red when in state 1, green when in state 2. In this case, the **Fill Color** attribute should be saved as part of each state of animation sequence **warning**. As a result, the warning light will always change to red in state 1 and green in state 2. These colors will remain part of the animation even if the color of the light is changed with the Graphics Editor: executing function **warning** with state 1 (2) will change the light back to red (green).

### Attributes Not Stored With Animation Example

An animation sequence is defined for a bouncing ball. Animation function **bounce** traces the path of the ball with 20 states. The ball just happened to be red when the 20 animation states were defined. If the Foreground Color attribute toggle button was selected, the ball would be red for each state. Even if the color is changed with the Graphics Editor, executing function **bounce** with states 1-20 will immediately change the color back to red again.

Making the ball blue becomes a somewhat tedious task - each of the explicitly defined states must be redefined with the ball being blue. However, if the Fill Color attribute button was not chosen, simply changing the ball's color would be sufficient. None of the state definitions would change the color back, so the ball would be blue in all states.

### Outside range to Lo/Hi Attribute

If **Outside range to Lo/Hi** is checked, then when an animation receives a value that is outside of its defined range, the animation will move/transform to the closest defined state. That is, if an animation named **gauge** is defined from 0 to 5, it will jump to the position defined for its highest value (5) if gauge is set to state 10. Note that the animation's state will actually be 10, even though the animated object only knows how to move/transform to its highest defined position (5).

By default this option is not enabled which means that if the value for a state change is outside of an object's own animation sequence range, the object ignores the state change.

### 3.2.3 Animation Pane List Context Menu

Right click on one of the rows in the Animation pane list to display the Animation pane's right-click context menu.

Some of the options on the Animation pane right-click context menu include:

- **Delete Selected Animation**
- **Delete Current State**
- **Rename Animation...**
- **Set all Initial States to Current States**
- **Set Selected Initial State to Current**
- **Execute All Initial States**
- **Refresh Animation**

#### Delete Selected Animation

To remove an animation name and all of its states, highlight it in the List Area (by clicking on it with the left mouse button) and choose Delete Selected Animation.

**NOTE:** Deleting an animation in the Animation pane list CANNOT be undone via the Undo command.

#### Delete Current State

Choose this option to remove just the current animation state (not the animation name itself). Highlight the animation name or enter it in the Name field, enter the appropriate animation *state* (the one you wish to delete) in the State field, and select Delete Current State.

**NOTE:** Deleting an animation's current state in the Animation pane list CANNOT be undone via the Undo command. Also, remember that you cannot delete a selected animation or current state if the animation is a "child" animation in the Animation pane list. You must focus into the grouped object and select the specific child that has the defined animation.

## Rename Animation...

This option opens the Rename Animations window to allow you to rename one or more animations. See [Section 3.8 - Renaming Animations](#) for more info

## Set All Initial States to Current States

By choosing this option, the initial states of all the listed animation sequences, selected or not, are reset to the sequences' current states.

## Set Selected Initial State to Current

Choose this option to change the initial state of the selected animation to its currently set state. For example, if **box** is first defined with an initial state of 0, but is now at state 5; choosing this option will cause the initial state of **box** to become 5.

## Execute All Initial States

Choose this option to immediately reset all of the selected object's animations to initial states.

**NOTE:** **Multiple animation sequences with the same name may be edited to have different initial states. This scenario can be confusing and should be avoided if possible. If this does occur, the initial state associated with the object behind all (lowest in the object z-order) others will have precedence and will become the initial state for the animation function (i.e., all of those sequences).**

## Resetting Initial States During Runtime

Application programs, control code, or stimulus definitions can execute a special built-in function to set all animations in a design to their initial states. The function's name is **altiaExecuteInitialStates** and it must be executed with a value of 1. See [Section 3.9 – Special Built-in Functions](#) later in this chapter for more details.

## Refresh Animation

Choose this option to set the selected animation to its current state. This is most often used when writing or debugging Control logic code to quickly or repeatedly trigger a WHEN statement.

## 3.3 Defining Animation

### 3.3.1 Overview

The process of defining animation is similar to recording frames in a video camera. You define at least two states for an object, then Altia Design interpolates any states that would fall in between those two points. For example, you could draw a circle and assign to it a state of **0**. You might then move the circle to the right and assign to it a state of **10**. Altia Design will then interpolate states **1-9**.

Graphics Editor *actions* that are saved as animation are **Move**, **Scale**, **Rotate**, **Stretch**, **Distort**, **Show**, and **Hide**. In addition, the *object attributes* of fill color, outline color, fill pattern, outline style, and opacity may be saved as animation. Any animation sequence of two or more states may include any combination of actions and attribute changes.

### 3.3.2 Animation Sequences Versus Functions

Animation sequences are sometimes called animation “functions”. While the terms are often used interchangeably, the difference becomes apparent when different sequences for different objects are given the same name. Formally, an animation “sequence” is the behavior of *one* object in response to changes in the sequence’s state. An animation “function” is the behavior of *one or more* sequences with the same name. Therefore, it is best to think of replaying the animation function, which causes all animation sequences with that name to update appropriately.

It follows that the **Animation** name and **State** fields of the Animation pane really refer to the animation function, since changing the state affects all sequences with names that match the **Animation** name field.

#### Function State Changes

When the state of an animation function is changed (for example, by changing the state value field in the Animation Editor, via input or timer stimulus, or by an application program), the state change is “broadcast” to all objects that have animation sequences with the same name. As each object receives the state change message, it decides how it should change its visual appearance.

After all objects have a chance to determine the visual changes they require, the actual display is updated in an instantaneous fashion. If, however, the visual changes span a large screen area (approximately greater than 1024x768 pixels), the update may appear to flash.

### 3.3.3 Floating Point Animations

You can use a floating point value as an animation state value. This means that you can define a specific state with a floating point value (e.g., 1.5). For an existing animation that has multiple states already

defined, you can set the animation's state to a floating point value and it will automatically interpolate to that value.

For example, you can have a meter needle animation with defined states of 0 and 1.0. If you set the animation's current state to 0.5, the needle will move half-way between 0 and 1.0.

### 3.3.4 How To Define Animation

Defining animation with Altia Design is a simple process.

1. With the Graphics Editor in **Edit** mode, select the object to which you want to add an animation sequence.
2. Enter an animation name in the **Animation** name field of the Animation pane.
3. Enter a state value in the **State** field either by typing in a number or toggling the increment/decrement arrows.
4. Press the **Define** button.
5. Transform the graphical object using the Graphics Editor's tools or commands.
6. Assign a state value to the transformed object by entering a new number in the **State** field or use the increment/decrement arrows next to the **State** field to change the number.

If the state value chosen is not consecutive to the value chosen in step 4, the intervening animation states will be automatically interpolated and defined by Altia when **Define** is pressed (in step 8). See [Section 3.4, State Interpolation](#) for more details about state interpolation.

7. Press the **Define** button.

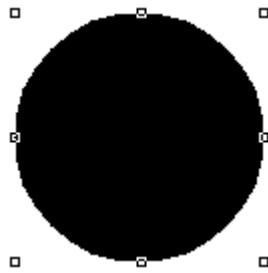
Continue defining animation by repeating step 5 through step 7, if desired.

**NOTE:** **Move, Scale, Rotate, Stretch, Distort, Hide/Show, Color, Pattern, Line Style, and Opacity changes are saved as animation.**

### Example of Defining Animation

In [Figure 7-4](#), a black circle is given the animation name sphere and a state of 0. When its state is changed to 1, the circle becomes striped.

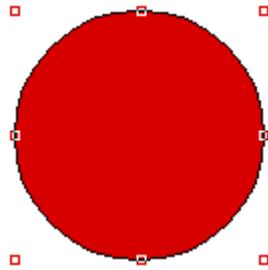
- A circle with a fill color of black is drawn in the Graphics Editor and selected.



- The circle's animation is defined with the name **sphere** and a state value of 0.



- The circle's fill color is changed to red.



- The state value of **sphere** is incremented to 1 and defined.



**Figure 7-4: Example of Animation Definition**

**REMINDER:** To define an animation name and state value for an object, the object must be selected and you must press Define.

To test an animation definition, use the increment/decrement arrows to toggle the state value field, or select the animation in the Animation pane's list and use the Animation State Slider to play back the animation.

### 3.3.5 Animating Groups

The power of animations is greatly enhanced by the ability to animate groups. By layering animations on different levels within a group, almost any type of behavior can be defined. Taking advantage of Altia's hierarchical object structure can lead to some very complex related animations, such as a blinking light being placed on the moving handle of a slider.

#### Exclusive Animations

Objects in Altia may have more than one animation defined on them. If more than one animation is defined for the object on the same Focus Level (or group level) of the object, then these animations become mutually exclusive - setting a state for one animation causes its behavior to override the behavior of any other animations previously defined.

For example, if one animation named `color` changed the color of an object, and another animation named `x_position` placed the object along a horizontal line, then changing the value of the `color` animation would move the object to the location and set the color defined for the given `color` state. Even if you had previously set the value of `x_position` to a place where it was moving along a line, the position would change to the position associated with the `color` animation. If two animations need to work together at the same time, careful use of grouping can create such a behavior.

#### Multiple, Hierarchical Animations

By creating animations on different Focus Levels of a grouped object, you can create related and inherited animations. For example, an object that is a blinking light might be grouped with a parent object that moves in the horizontal direction, resulting in an object that both blinks and moves back and forth.

**NOTE:** We recommend that you always define animations on a grouped object even if you don't foresee the need to create multiple hierarchical animations. This allows you the maximum flexibility if you decide to make changes to the animated object at a later time.

For example, if you draw a circle and group it, then define a bouncing animation on the group, you could later focus in to the group and replace the circle with a square. After focusing out, the bouncing animation would apply to the square - if you had defined the bouncing animation to the original circle WITHOUT grouping it, you would have to re-create the bouncing animation if you later wanted to change the circle to a square (animations cannot be moved from one object to another).

## 3.4 State Interpolation

Defining each individual state value of an animation sequence is not always necessary. Altia Design will interpolate any state values between two defined states that are non-sequential. For example, if you define an animation state value of 0, then define another state value of 20, Altia Design will linearly interpolate animation state values 1-19.

Interpolation can be used when moving, scaling, stretching, rotating, and changing the opacity of objects. ***Interpolation cannot be applied for changes in color, pattern, outline, or show/hide.***

Rotation interpolation requires a rotation point. The rotation point used for all rotation interpolation of an object will be the last rotation point the object was rotated about before **Define** was pressed.

### Example of State Interpolation

In [Figure 7-5](#), a square is given an animation name of **box** and a state value of 0. The square is then moved diagonally up and to the right and given a state value of 5. Altia Design interpolates the position of box when the state values are **1-4**.

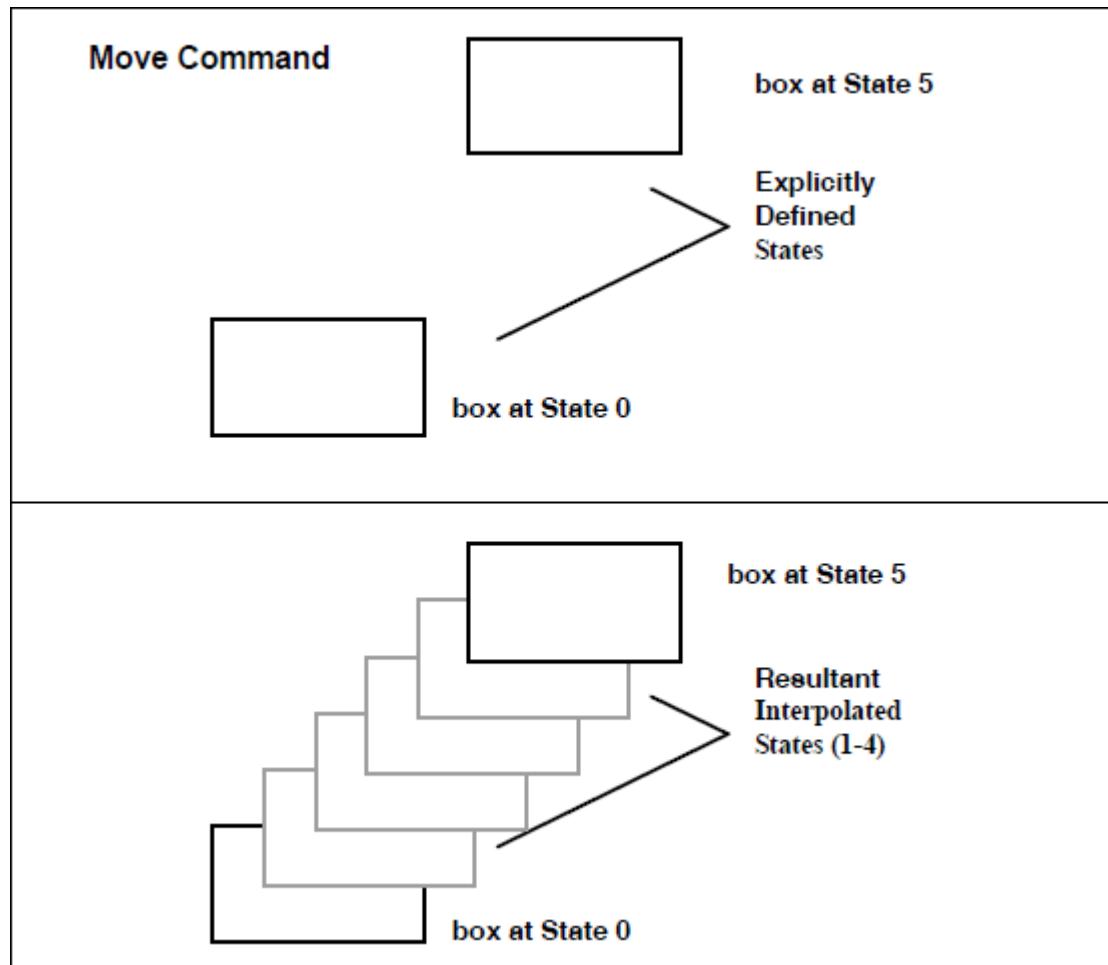


Figure 7-5: Example of Explicitly Defined and Interpolated States

## 3.5 Replaying Animation

Once animation is defined, it can be replayed using several methods.

- Cycle through the state numbers with the Animation pane by highlighting an animation row from the list or typing into the **Animation** name field the animation you would like to change. Click on the **State** field's increment/decrement arrows, or drag the Animation State Slider's handle to change the animation state
- Execute the animation with state values generated from stimulus input events such as mouse events (mouse button down, up, motion, etc.) Please see [Chapter 4: Stimulus Editor](#) for more information on using this method.
- Define control code that triggers animation state changes. Please see [Chapter 6: Control Editor](#) for more information on using this method.
- Generate function calls from external code to change the animation states. Please see the [Altia API Reference Manual](#) for more information on using this method.

**NOTE:** It is a good idea to always test newly defined animation by toggling the increment/decrement arrows or using the Animation State Slider.

## 3.6 Redefining Animation

Animation sequences can be selected for editing/deleting by simply highlighting an animation row in the Animation pane list. This populates the **Animation** name and **State** fields with the selected animation sequence/function name and its current state.

### Revisiting the Menu Options

To delete the selected animation sequence(s), select the **Delete Selected Animation** item from the Animation pane list's right-click context menu, or by pressing the **Delete** key on your keyboard. To delete the current explicitly defined state, select **Delete Current State** from the Animation pane list's right-click context menu. To rename the animation sequence, double-click on the animation name within the Animation pane list, or select the **Rename Animation...** item from the Animation pane list's right-click context menu.

### How to Redefine Animation

Change a previously defined or interpolated animation state by doing the following:

1. With the Graphics Editor in **Edit** mode, select the object to be modified.
2. In the Animation pane's list, highlight the name of the appropriate animation by clicking on it with the left mouse button. It will then appear in the **Animation** name field.

If the animation you selected was a “child” animation, you must focus one or more levels into the object and select the specific child that has the defined animation. The “child” notation will disappear when you have focused in to a level such that the correct child object is selected.

3. Change the **State** value to the state to be redefined.
4. Change the graphical object - move, scale, rotate, stretch, distort, hide, or show it or change its color, pattern, brush, or opacity.
5. Press **Redefine**.

Use the increment/decrement arrows or the Animation State Slider to test your animation.

**NOTE:** When redefining a state, always place the object into that animation state before changing it for the new definition. If you change the object and then set the Animation Name and State fields in the Animation pane, the change will be applied to the object as a whole and not to the individual state that you are redefining.

## 3.7 Moving or Transforming an Animated Object

Animation behavior is defined relative to the object’s location so that when the object is moved, rotated, distorted, or scaled, the animation will replay relative to the new location. For example, an animation that moves an object 100 pixels to the left will continue to move the object 100 pixels to the left after the object is moved, scaled, rotated, distorted, or stretched.

Likewise, when an animated object is added to a group or a group is created that just contains the object, the object’s animations execute relative to the group’s position and any scale, rotate, distort or stretch that is applied to the group. The group can also be animated to dynamically move, scale, rotate, distort, and/or stretch, etc. When these group animations occur, they will have a relative effect on the object in the group and its own animations.

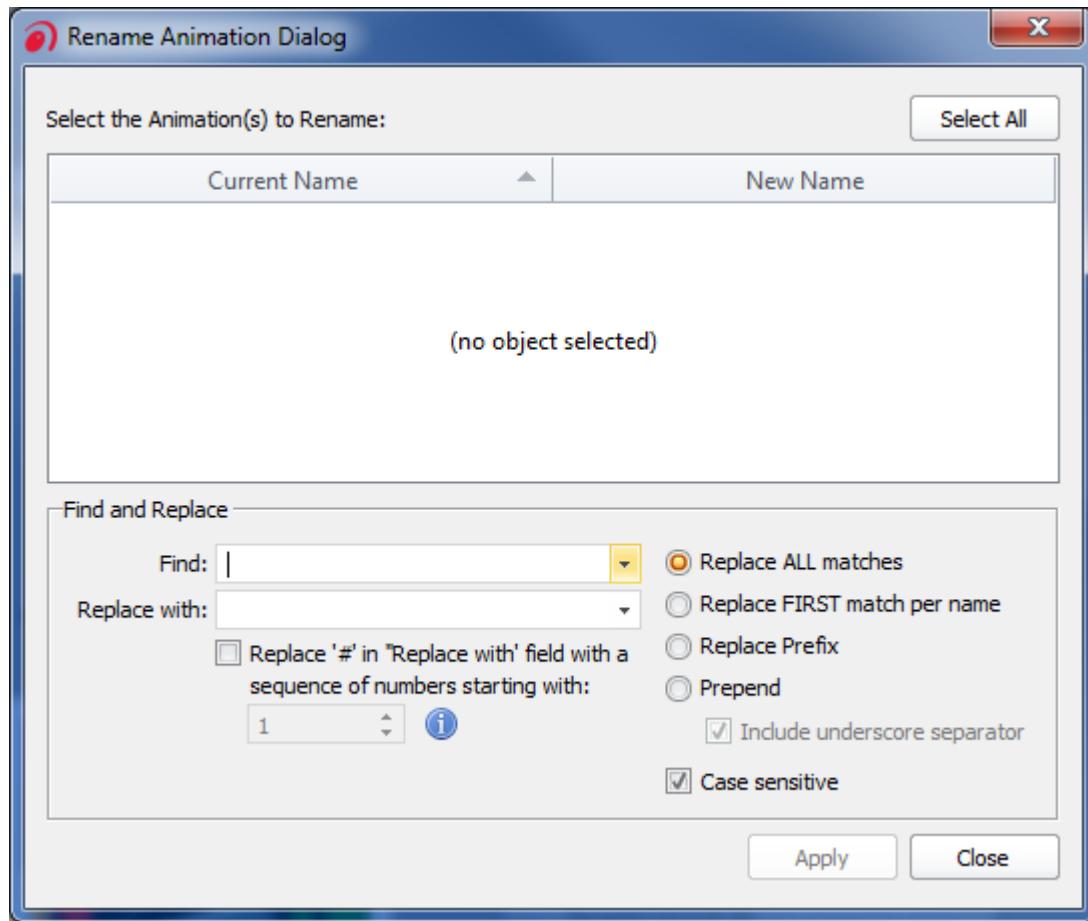
By creating levels of groups and animating at the various group levels, very complex animation sequences can be created. For example, a rotating wheel on a moving car is quite easy to create. A fully functional robot arm on a moving platform takes only a few extra steps.

In order to have a rotate, scale, distort or stretch apply to and change an entire animation sequence, group the object before rotating, scaling, or stretching it.

## 3.8 Renaming Animations

Animation names may be globally modified using the Rename Animation Dialog. Changes will apply to both user defined and intrinsic animations listed in the Animation Editor. Names used to define connections, stimuli, and control code are also affected.

### 3.8.1 Rename Animation Dialog



The dialog may be accessed several different ways:

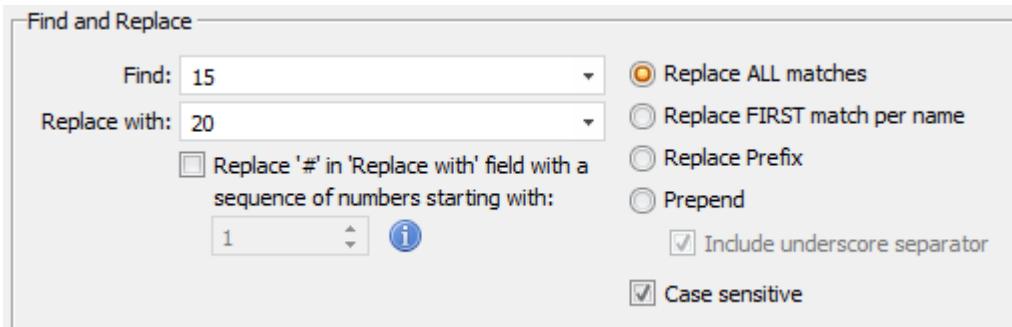
- Press the **Rename** button in the **Home** ribbon
- Press the **Ctrl-Alt-H** keyboard sequence
- Choose "**Rename Animations...**" in the Animation Editor's context menu
- Choose "**Rename Animations...**" in the Stimulus Editor's context menu
- Choose "**Rename Animations...**" in the graphic object's context menu

### 3.8.2 Animation Name List

Current Name	New Name
15_angle_out	
15_card	
15_knob	knob
15_knob_AlreadyInitialized	knob_AlreadyInitialized
15_knob_AlreadyOpen	knob_AlreadyOpen
15_knob_InitTemp	
15_knob_LastAngle	
15_knob_MaxAngle	
15_knob_MaxFar	
15_knob_MaxVal	

Animation names belonging to one or more selected graphic objects are shown under the "Current Name" column of the list. These names may be used in control code, stimuli, connections, or as animations defined on the selected objects and their children. The "New Name" column holds the names after the "Find and Replace" criteria have been applied. However, only selected animation names in the list are affected by the criteria.

### 3.8.3 Find and Replace



Names may be modified by several different methods as indicated by these radio buttons:

- **Replace ALL matches** - Any part of a name that matches the **Find** condition will be replaced by the contents of the **Replace with** field.
- **Replace FIRST match per name** - Only the first match of the **Find** condition found in the name will be replaced by the contents of the **Replace with** field.
- **Replace Prefix** - If a prefix exists, replace it with the contents of the **Replace with** field. Any characters before the first underscore character, '\_', in a name are considered part of the prefix.

- **Prepend** - Add the contents of the **Prepend** field in front of the existing name. Check the **Include underscore separator** checkbox to automatically insert an underscore to make the additional text an animation name prefix.

## "Find" Field

What you enter in this field will be replaced by the "**Replace with**" field. Valid characters for this field include '0'-'9', 'a'-'Z', and '\_'. Additionally, these characters are allowed with special searching meaning.

- '**A**' - Replace subsequence characters only from the start of the name. For example, "^co" will only replace the starting "co" in "color\_column", but not the second.
- '**.**' - Replace any single character in the name. For example, "...\_button" will replace "123\_button" and "abc\_button".
- '**\$**' - Replace preceding characters at the end of the name. For example, "ng\$" will only replace the ending "ng" in "single\_setting", but not in "single".

Uncheck the **Case sensitive** checkbox to ignore case during the character search.

## "Replace with" Field

This field is the replacement text for matches from the "**Find**" condition. Valid characters for this field include '0'-'9', 'a'-'Z', and '\_'. Additionally, the character '#' is allowed with a special function when **Replace '#' in "Replace with" field...** is checked. Hover your mouse over the information icon to see a full description of this function.

You may leave this field empty to remove the characters found under the "Find" condition.

## 3.8.4 Action Buttons

### Apply

The Rename Animation Dialog gives you a preview of the changes you are about to make. Changes are not applied to your design until you press the Apply button. After this button is pressed, the animation name list is refreshed with the modified names and the "Find and Replace" criteria is applied on the new names.

**WARNING:** You cannot undo the name change after the Apply button is pressed!

### Close

This button closes the Rename Animation Dialog. You do not need to close the dialog to select different graphic objects.

### 3.8.5 Animation Names of Instances

When you select one or more objects that are instances or parents of instances, the Rename Animation Dialog switches into a special mode where only the prefix of animation names may be changed. The **Replace ALL matches** and **Replace FIRST match per name** are disabled. Furthermore, the "Replace with" field must not be empty to keep the prefix from being deleted.

Animation names of instance objects are automatically selected in the animation names list and may not be deselected.

To bypass these restrictions for instance objects, focus into a child object that is not an instance and does not have instance children.

## 3.9 Special Built-In Functions (Design, Runtime and Generated Code)

Altia has a number of built-in functions (animations) with capabilities beyond those of user-defined animation/stimulus functions. These built-in animations can be set anywhere an animation can be manipulated in Altia Design (the Animation pane, defined stimulus, in control logic, etc.)

**NOTE:** The built-in functions in this section are fully supported in the Altia Design editor, the Altia Runtime environment, and in DeepScreen generated code. Refer to your DeepScreen target's documentation for additional details.

These functions are listed in the following sections, grouped by category.

### 3.9.1 Object Input

Functions controlling object stimulus input:

#### **altiaEnableOnlyInput:**

When sent an object ID number, this function disables all stimulus input areas of all other objects (i.e., only the given object will receive stimulus input). Subsequent calls with object ID numbers can re-enable the stimulus input areas of those objects as well.

#### **altiaDisableOnlyInput:**

When sent an object ID number, this function re-enables all stimulus input areas for objects if **altiaEnableOnlyInput** was previously called with that ID number. If the object ID number is 0, all stimulus input areas of all objects are re-enabled.

### 3.9.2 Object Dimensions

Functions to query an object's dimensions:

#### **altiaGetObjWH:**

Send the object ID of the object to be queried to this animation - this should be called first!

### **altiaObjW:**

Query the object whose ID number was specified first by **altiaGetObjWH** for the width of that object in pixels. If the object has been transformed (scaled, rotated, and/or stretched), the reported width includes the transformation.

### **altiaObjH:**

Query the object whose ID number was specified first by **altiaGetObjWH** for the height of that object in pixels. If the object has been transformed (scaled, rotated, and/or stretched), the reported height includes the transformation.

For example, querying the width and height can be done in control code like the following (where the number 100 is the id# of the object in this example):

```
SET altiaGetObjWH 100
```

After sending the id# to altiaGetObjWH, query the value of the built-in animation **altiaObjW** or **altiaObjH** for the width or height, in pixels, of the object. The following control statement tests if the object's width and height are less than or equal to 50:

```
IF altiaObjW <= 50 AND altiaObjH <= 50
```

In application code, use the Altia API function **altiaSendEvent()** or **AtSendEvent()** to send a similar event as in:

```
altiaSendEvent("altiaGetObjWH", (AltiaEventType)100);
```

Then use the Altia API function **altiaPollEvent()** or **AtPollEvent()** to get the current value for each of these built-in animations as in:

```
AltiaEventType width, height;
altiaPollEvent("altiaObjW", &width);
altiaPollEvent("altiaObjH", &height);
```

### 3.9.3 Object Movement

Functions controlling object movement:

#### **altiaSetObj:**

Send the object ID of the object to be modified to this animation - this should be called first!

#### **altiaMoveObjX:**

Sets the horizontal pixel offset to the assigned/ passed value.

#### **altiaMoveObjY:**

Sets the vertical pixel offset to the assigned/ passed value.

**NOTE:** Once BOTH offsets are set, the object identified by the previous call to **altiaSetObj** moves relative to its current position.

#### **altiaGetObjXY:**

Pass an object ID number to this animation and the x and y coordinates of that object will be returned to the built-in animations **altiaObjX** and **altiaObjY**, respectively.

### 3.9.4 Miscellaneous Functions

Additional functions that do not fit into any of the above categories:

#### **The “None” Animation:**

The animation name **None** is a reserved name that only supports a state value of 0. The Animation Editor will not allow you to change the state of **None** to a value other than 0.

#### **\_global\_:**

This very useful prefix can be added to an animation name to prevent renaming of the animation when copying an object. For example, if an object has animations **1\_a**, **1\_b**, and **\_global\_c**, a quick copy and paste of the object will yield a new object with animations named **2\_a**, **2\_b**, and **\_global\_c**. With this feature, it is easy to create many objects that all use the same animation name (perhaps as an initial

value, as a display color, etc.). If it is later desired to rename a global animation, the **Rename** field of the Rename Animation window can do it.

### **altiaCacheOutput:**

Application code using the C/C++ API has the ability to choose when Altia updates visual changes to objects. The **altiaCacheOutput()** or **AtCacheOutput()** API functions enable event caching and the cache can be flushed at designated points in the application code. Objects in Altia that have visual changes will update simultaneously when the cache is flushed. This reduces the total number of screen updates. This can significantly improve performance and the perceived responsiveness of a design.

There is also a built-in Altia animation function, **altiaCacheOutput**. Execute this animation from control or even stimulus to suspend or resume updates of object changes. For example, to suspend the updates from control code, use a **SET** statement like the following:

```
SET altiaCacheOutput 1
```

When **altiaCacheOutput** is set back to 0, all objects that need updating will get updated. If caching is enabled and never disabled, visual changes for objects will never happen unless the Altia window has another reason to refresh (for example, it is exposed after being minimized or covered by another application window). In typical situations, it is very important to only enable caching when it is truly necessary and disable it otherwise.

### **altiaColorFgObj:**

Send a string to this animation containing an RGB value (such as “46 20 00”) or the name of a color (such as “red”) to change the foreground (Outline) color of the object set by the **altiaSetObj** built-in animation.

### **altiaColorBgObj:**

Send a string to this animation containing an RGB value (such as “046 200 0”) or the name of a color (such as “red”) to change the background (Fill) color of the object set by the **altiaSetObj** built-in animation.

### **altiaErrorOutput:**

If control code or a client application using the Altia API detects some kind of error condition, it can force the display of an error dialog by writing a string to the built-in Altia animation function **altiaErrorOutput**. The string will appear in an Altia Error dialog. The user must press **OK** in the dialog before they can continue.

As an example, a **SET** statement in control can activate the dialog with something like:

```
SET altiaErrorOutput "Range Error: Max value too large."
```

### **altiaInitDesign:**

Altia automatically routes this event when a design is loaded. The value routed is the ID number of the design that was loaded, starting with 0.

### **altiaSetBrush:**

Send a numerical value to this animation representing the brush pixel width to change the line width for an object set by the **altiaSetObj** built-in animation.

### **altiaSetFont:**

Send a X11 style font ID string to this animation to change the font for an object set by the **altiaSetObj** built-in animation. For example, to use **altiaSetFont** in control code, use **SET** statements like the following:

```
SET altiaSetObj 15
SET altiaSetFont "*-courier new-medium-r-normal--13-*-*-*-*
65-*"
```

See the Advanced Font Attributes in the in the Navigator pane for an example of an X11 font string or refer to the font string "Point Size Font Palette Specifications" documentation in the **altia.ini** file located in [altia Design Install Folder] \defaults\.

### **altiaSetImage:**

Send a image file string to this animation containing the path and file name of an image to change the image for an Image Object set by the **altiaSetObj** built-in animation. This animation will only function on an Image Object.

### **altiaSetObj:**

Identify the object ID number of the object to be manipulated by the next built-in animation function. For example, to use control logic to set the color of object 116, use the **altiaSetObj** statement like the following:

```
SET altiaSetObj 116
SET altiaColorFgObj "255 255 0"
```

### **altiaSetOpacity:**

Send a value from 0 to 100 to this animation to change the opacity of the object set by the **altiaSetObj** built-in animation. Sending 0 will make the object completely invisible and sending 100 will make the object completely solid.

## 3.10 Additional Built-In Functions (Design and Runtime Only)

The built-in functions in this section are similar to the built-in functions described above in usage. These built-in animations can be set anywhere an animation can be manipulated in Altia Design (the Animation pane, defined stimulus, in control logic, etc.)

**WARNING:** The built-in functions in this section are available in the Altia Design editor and the Altia Runtime environment ONLY.

They are NOT supported in DeepScreen generated code.

These functions are listed in the following sections, grouped by category.

### 3.10.1 Design Manipulation

Functions for opening, closing, and manipulating design (.dsn) files:

#### **altiaSetDesignId:**

Send the design ID number of the design to be manipulated to this animation - should be called to identify the design prior to opening it, manipulating/cloning objects of the design, or opening views into the design.

#### **altiaOpenDesignFile:**

Sets the name of a design file to open: call the function with a file name surrounded by double quotes (e.g., "my\_file.dsn"). Note that to make the design visible, a view must also be opened.

#### **altiaCloseDesignId:**

Closes the design with the assigned/passed design ID number.

#### **altiaExecuteInitialStates:**

When this animation is sent a non-zero value, all open designs will have all their animations set to their initial states.

#### **altiaQuit (ONLY APPLICABLE IN THE RUNTIME ENVIRONMENT):**

When this animation is sent any value, the Altia runtime will exit. The name of this function can be changed via an entry in the Altia configuration file.

## 3.10.2 View Manipulation

Functions to manipulate views. Views are windows which can display design files.

### **altiaSetView:**

Send the view ID of the view to be manipulated to this animation - should be called first! To manipulate the Main View, use an ID of 0.

### **altiaSetViewX:**

Sets the x coordinate of the design which should appear in the lower left corner of the view.

### **altiaSetViewY:**

Sets the y coordinate of the design which should appear in the lower left corner of the view.

**NOTE:** Once BOTH coordinates are set, the view identified by the previous call to altiaSetView is updated to display the indicated portion of the design.

### **altiaSetViewWidth:**

Sets the width of the view to the assigned/passed value.

### **altiaSetViewHeight:**

Sets the height of the view to the assigned/passed value.

**NOTE:** Once BOTH dimensions are set, the view identified by the previous call to altiaSetView is updated to the new size.

### **altiaSetViewMag:**

Sets the view's magnification to the assigned/passed value divided by 100.

### **altiaSetViewName:**

Sets the name of the view; call the function with a name surrounded by double quotes (e.g., "My View"). The view's banner is then updated with the new name.

**altiaOpenView:**

Opens a new view with the assigned/passed view ID number.

**altiaOpenViewId:**

Treats the window with the assigned/passed windowing system ID number as a new Altia view. By passing a window ID to this animation, you can have Altia draw into a different window on the system. This is extremely powerful and has been used to draw fully interactive Altia designs into child windows of MFC or Motif applications.

**altiaCloseView:**

Closes the view with the assigned/passed view ID number.

**altiaCloseViewPending** (*ONLY APPLICABLE IN THE RUNTIME ENVIRONMENT*):

If this event/function has been selected via the function **altiaSelectEvent()** in Altia's API, then a user action to close a run-time view will *not* close the view. Instead, this event will be routed, with the value being the ID number of the view the user tried to close. This allows client control over the view exiting process, which could close the view via **altiaCloseView** (above), exit the run-time Altia session via **altiaQuit**, or ignore the request altogether.

**altiaGetViewSize:**

When sent a view ID number, two events/functions are routed: **altiaViewWidth** and **altiaViewHeight** with values of the view's width and height, respectively.

**altiaWindowId:**

When sent a view ID number, routes the same event/function (**altiaWindowId**) with the windowing system ID number of the view's window.

**altiaViewResize:**

Altia automatically routes this event/function when a view is resized by the user. The value routed is the ID number of the view that was resized.

### 3.10.3 Object Cloning

Functions controlling object cloning:

#### **altiaSetObj:**

Send this animation the object ID number of the about-to-be-created clone - should be called before creating the clone!

#### **altiaCloneObj:**

Send this animation the object ID number of the object to be cloned. Altia then clones the object and gives it the object ID number from the preceding call to **altiaSetObj**.

All animation/stimulus/control defined for the original object is retained by the clone. However, two types of modifications to these definitions are made automatically to ease manipulation of the clone:

1. All user-defined animation/stimulus functions are renamed by prepending “#:” to the original name (where # is the object ID number of the clone). Note that the built-in functions being discussed in this section are not renamed.
2. Any statement in the control definition that sets the value of a built-in function (other than **altiaSetObj**) to the object ID number of the original object is modified to use the ID number of the clone instead.

### 3.10.4 Clone Manipulation

Functions manipulating cloned objects:

#### **altiaSetObj:**

Send the object ID of the clone to be manipulated to this animation- should be called first!

#### **altiaMoveClonedObjX:**

Sets the horizontal pixel offset to the assigned/passed value.

#### **altiaMoveClonedObjY:**

Sets the vertical pixel offset to the assigned/passed value.

**NOTE:** Once BOTH offsets are set, the clone identified by the previous call to altiaSetObj moves relative to its current position.

### **altiaDeleteClonedObj:**

Deletes the clone with the assigned/passed object ID number.

### **altiaDeleteAllClones:**

When sent a value of 1, all clones will be deleted.

## **3.10.5 Miscellaneous Functions**

### **altiaFindObj:**

For the Altia Design editor only, send an object ID number to this animation and the editor will select the object with this ID number if it exists.

### **altiaGenListing:**

Send a string to this animation that is a file name (such as listing.txt) or the full path for a file name (such as c:\tmp\listing.txt on Windows). A simple text listing for all objects in the current design is generated and written to the given file name. The listing format is identical to the output from the **Export > Object Listings > Object Listing** option in the Altia Design **File** ribbon. To verify if the action was successful, check the value of **altiaGenListingReply** using an **IF** statement in control or **AtPollEvent()** in the Altia C/C++ API. A value of 1 means the operation was successful. A value of 0 means the file could not be written.

### **altiaGetTimeCount:**

Returns an ever-increasing time count, in milliseconds. The count “wraps” around to 0 when it overflows, and is always positive.

### **altiaGenXRef:**

Generates a name cross-reference of all symbols in the current design to a file: call the function with a file name surrounded by double quotes (e.g., “crossref.txt”). Each symbol name in the generated cross reference is followed by a hexadecimal code in parentheses. Each bit of the code provides information about the symbol:

- bit 0 - if set, indicates symbol is an integer animation name
- bit 1 - if set, indicates symbol is a string animation name
- bit 2 - if set, indicates symbol is referenced by stimulus
- bit 3 - if set, indicates symbol is modified by stimulus
- bit 4 - if set, indicates symbol is reference by a timer
- bit 5 - if set, indicates symbol is modified by a timer
- bit 6 - if set, indicates symbol is referenced by a control statement
- bit 7 - if set, indicates symbol is modified by a control statement
- bit 8 - if set, indicates symbol is a global control variable
- bit 9 - if set, indicates symbol is a local control variable

### **altiaRecordClientEvent:**

Enable or disable recording of client events dynamically (previously, they could only be enabled from a command line option when starting Altia Design or Altia Runtime). If the value for the animation is non-zero, it enables recording of client events. A value of zero (0) disables recording of client events.

### **altiaShellOpen:**

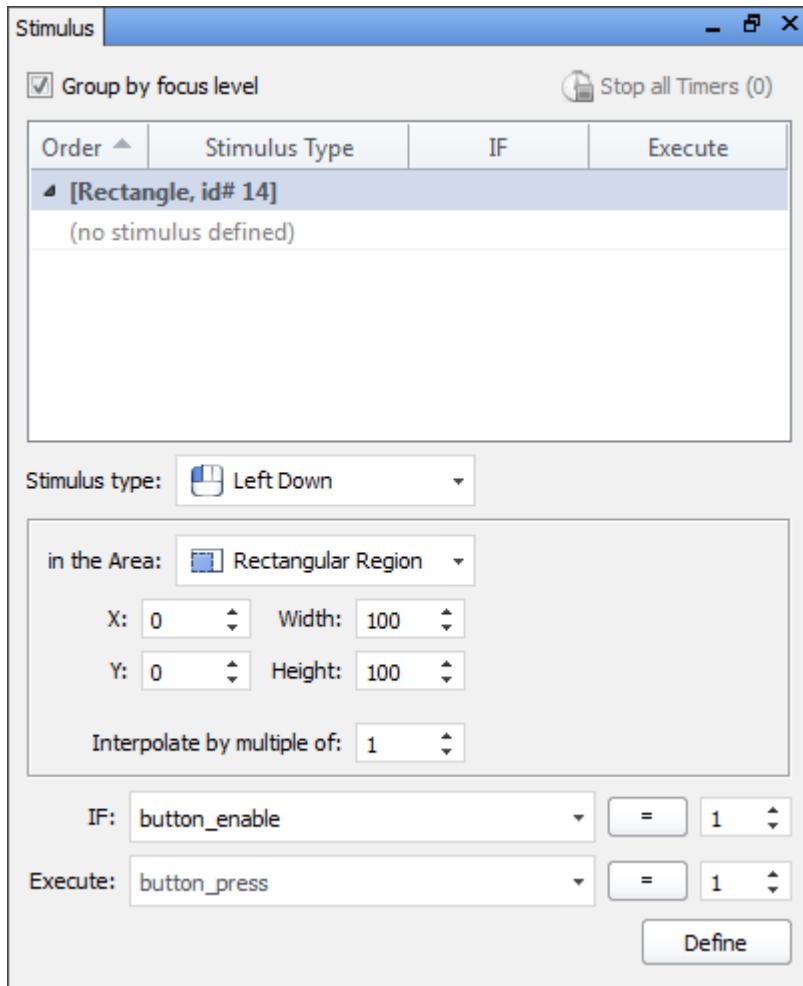
Gives the assigned/passed value (usually a file name surrounded by double quotes) to the system. If the system has an action associated with that type of file, then that action will be performed. For instance, in Windows if a PDF file name is passed to this animation then (if installed and properly associated) your PDF reader will open and load the file.

### **altiaStartProgram:**

Gives the assigned/passed value (typically an executable program name surrounded by double quotes) to the system for execution. Parameters for the executable are allowed.

# Chapter 4: Stimulus Editor

With Altia Design's Stimulus Editor pane you can create input stimulus (mouse clicks/touchscreen presses, keyboard input, etc.) that will drive the object animation you defined in the Animation Editor. Using this powerful feature, you can create robust interactive objects.



## 4.1 Stimulus Editor Layout

The Stimulus Editor is divided into four major sections:

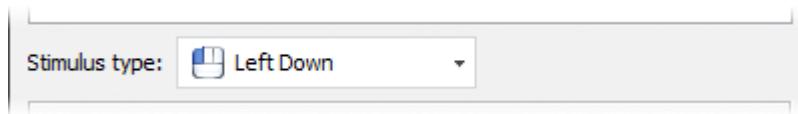
- **List Area:** This section displays all of the defined stimulus for the selected object (it is discussed further in [Section 4.4, Defined Stimulus List](#)). Unlike the List Area in the Animation pane, this list area only shows stimulus for one object at a time. If the object is a group, stimulus associated with child objects is shown and can be selected for inspection, but you must focus into the group if you wish to modify stimulus defined on a child object.

It is very common to have multiple stimulus definitions defined for a single object. For example, if multiple actions are required on a single left (button) down, they would be defined (probably in a specific order) and would appear in the list area in the order they were defined. The order in the list area is also the order in which the stimuli will execute. Multiple stimulus definitions may also be specified for different types of inputs (left down and left up).

Order	Stimulus Type	IF	Execute
▲ [Group, id# 5]			
1	Left Down	4_push = 0	4_push = 1
2	Left Up	4_push <> 0	4_push = 0
3	Leave	4_push <> 0	4_push = 0

List Area

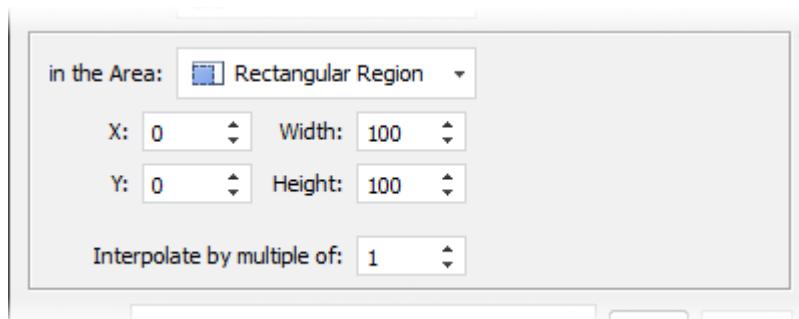
- **Stimulus Type:** The form of input that will trigger execution of a stimulus definition. Choose from: **Left Down, Left Up, Middle Down, Middle Up, Right Down, Right Up, Enter, Leave, Motion, Key Press, Key Release, and Timer.**



Stimulus Type

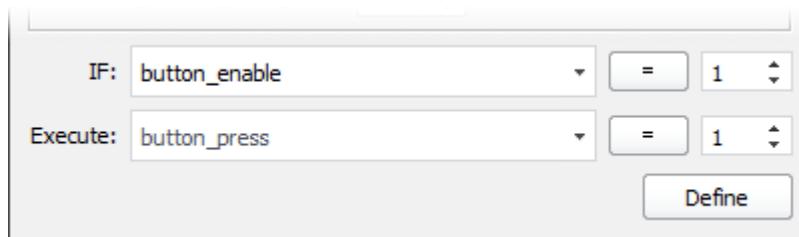
**NOTE:** Pressing a touchscreen's display with your finger corresponds to the **Left Down** and **Left Up** mouse stimulus types. The **Middle** and **Right** mouse event types are not used when designing stimulus for a touchscreen display. Also keep in mind that due to the touchscreen environment the mouse stimulus types of **Motion**, **Enter**, and **Leave** can only be detected during a **Mouse Down**.

- **Stimulus Area:** The region (relative to the center of the object) in which the stimulus must occur. Specify the region by selecting an option from the **Stimulus Type** dropdown. The available options are rectangular (**Rectangular Region**), circular (**Polar Region**), or anywhere within the entire object (**Whole Object**). The interpolation multiple is also set in this portion of the dialog. Stimulus interpolation multiples are covered in detail in [Section 4.7, State Interpolation](#).



**Stimulus Area**

- **Execution Area:** The action that will result when the Stimulus Area receives the appropriate Stimulus Type and all limiting conditions (if any) are met.



**Execution Area**

Use the dropdowns to select an animation in the list or type the animation names in directly. Numeric values can be typed in directly or the increment and decrement arrows are used to increase or decrease the **IF** condition and **Execute** state values.

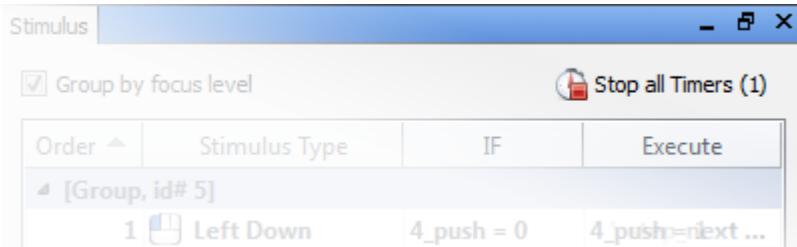
Press the **Define** button to “lock in” the specified **Stimulus Type**, **Stimulus Area**, **Enable Condition** (if any), and **Execute State**. The new definition will appear at the bottom of the order in the list area.

**NOTE:** The Define button will remain disabled until an animation name is entered in the Execute field. The IF condition is optional.

## 4.2 Stimulus Pane Additional Functions

Several useful functions are available within the Stimulus pane.

### 4.2.1 Stop All Timers



Click on the **Stop All Timers** button to immediately stop any active timers for all objects in the design (not just the selected objects). The Stop All Timers button will indicate the number of active timers in the model. If no timers are currently active, the Stop All Timers button will be disabled.

### 4.2.2 Stimulus Pane List Context Menu

Right click on one of the rows in the Stimulus pane list to display the Stimulus pane's right-click context menu.

Some of the options on the Stimulus pane right-click context menu include:

- **Cut Stimulus**
- **Copy Stimulus**
- **Paste Stimulus**
- **Delete Selected Stimulus**
- **Rename Animations...**

#### Cut Stimulus

Delete defined stimuli by highlighting the definition(s) in the List Area and choosing this option. Stimulus that has been cut will be placed in the Stimulus Editor's Paste Buffer.

#### Copy Stimulus

Use this option in conjunction with Paste Stimulus to add previously defined stimulus to an object. As with Cut Stimulus, this option will place the stimulus in the Stimulus Editor's Paste Buffer for later use.

## Paste Stimulus

Select this option to add the last cut or copied stimulus, then edit as necessary. Be sure the object to which you want the stimulus added is selected in the Graphics Editor. Stimulus in the Paste Buffer is pasted in front of items currently selected in the List Area. If no items in the List Area are selected, the new stimulus appears after all existing stimulus.

The order of stimuli in the List Area may be important because definitions of similar types (e.g., left button down) are evaluated in the order in which they appear in the List Area.

## Delete Selected Stimulus

Permanently delete defined stimuli by highlighting the definition(s) in the List Area and choosing this option. Please note that such a deletion *cannot* be undone so be very careful when performing this action.

## Rename Animations...

Change the animation names of defined stimulus by selecting this option and completing the statements in the Rename Window. For more information on the Rename Window, please refer to [Section 3.8, Renaming Animations](#).

## 4.3 Defining Input Stimulus

Defining stimulus to which animated objects will respond is done in three major steps:

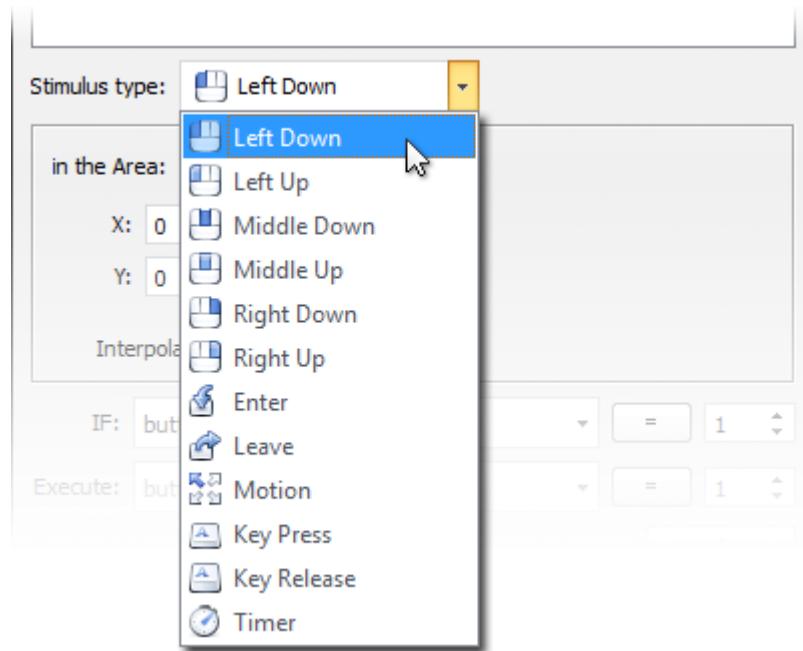
1. Define the type of stimulus that will trigger an action (key press, mouse button down/up, mouse motion, enter/leave, timer).
2. Specify the area in which the stimulus must occur (the whole object, rectangular portion, or polar portion).
3. Define the resulting action and add limiting conditions, if desired.

It should be noted that a “stimulus” function is simply a function that changes state in response to input, but has no animation sequence associated with it. Such a function is created from the Stimulus Editor when an “Execute” function is specified that does not have animation defined for it.

To add stimulus, select an object in the Graphics Editor. This object must be the one that will receive stimulus (but not necessarily the one whose animation will be triggered by the stimulus).

### 4.3.1 Defining the Stimulus Type

Start defining stimulus by making a selection in the **Stimulus Type** dropdown menu.



The following Stimulus Types are available:

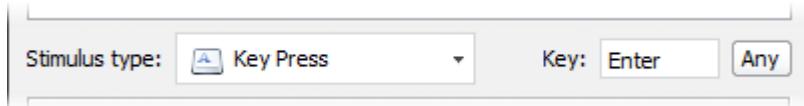
## Mouse Events

- **Left Down, Left Up, Middle Down, Middle Up, Right Down and Right Up.** Choose one of the mouse events if you want such an actions to trigger a stimulus event.
- Choose **Enter** or **Leave** to define stimulus that gets executed when the cursor either enters or leaves the Stimulus Area.
- Choose **Motion** as a Stimulus Type and Altia Design will watch for cursor movement in the Stimulus Area.

**NOTE:** Pressing a touchscreen's display with your finger corresponds to the **Left Down** and **Left Up** mouse stimulus types. The **Middle** and **Right** mouse event types are not used when designing stimulus for a touchscreen display. Also keep in mind that due to the touchscreen environment the mouse stimulus types of **Motion**, **Enter**, and **Leave** can only be detected during a **Mouse Down**.

## Keyboard Events

Choose **Key Press** or **Key Release** to create stimulus that responds to keyboard input. The default state is to respond to "any" keyboard input. If you wish for the stimulus to respond to a single key, click on the **Key** input field and type the key on your keyboard. The Key field will update to display the key pressed. If you wish to return to responding to "any" keyboard input, click the **Any** button.



**NOTE:** If you choose to make the Stimulus Type a Keyboard Event and specify "any" key, the state value when executed will correspond to the ASCII value of the key pressed, if it has one. If the key does not have an ASCII value (such as a function key), the state value when executed will correspond to the value of the key's keysym, shifted to the upper two bytes (to distinguish it from an ASCII value). Keysym values are defined by X11, and their definitions can be found in

`/usr/include/X11/keysymdef.h`

On Windows, keysym values are the same as the Virtual-Key Codes defined by the Microsoft Visual Studio environment.

In addition, stimulus definitions using Keyboard Events in the "any" key mode cannot perform input interpolation.

## Timer

Select **Timer** to create a Timer stimulus that will trigger an object's animation at timed intervals. Timers are started and stopped by animation or stimulus functions. Once started, a timer will wait a specified period of time, then "tick" to drive another function (unless stopped prior to its wait time elapsing). Timers can be "one shots" or repetitive and there can be multiple timer stimuli specifications for a single object.

For more information on the Timer, please refer to [Section 4.8, Timer Stimulus](#).

### 4.3.2 Defining the Stimulus Area

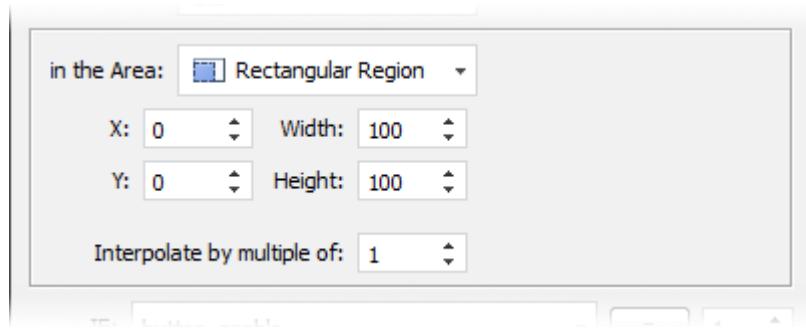
Once the Stimulus Type has been defined, you must define the **Stimulus Area** in which that Stimulus Type will occur. The area of the object that is to be sensitive to the chosen input type is defined in the In the Area dropdown in the Stimulus pane.

Choose from the following Stimulus Areas.

- **Rectangular**
- **Polar**
- **Whole**

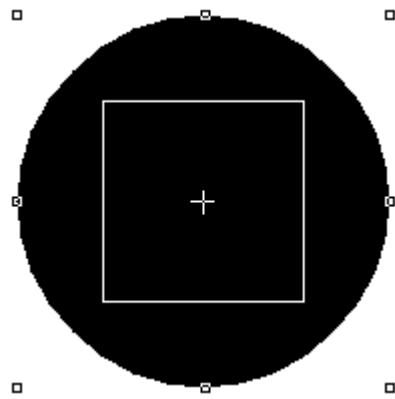
## Rectangular

To change the size of the Stimulus Area (in pixels), adjust the values in the **Height** and **Width** fields by using the increment/decrement buttons or by typing values into the fields.



To change the position of the Stimulus Area, adjust its x and y coordinates. The **X** and **Y** fields define the offset (in pixels) of the center of the area from the center of the object. Therefore, the X and Y coordinates of 0,0 (the default setting) place the center of the Stimulus Area at the center of the object.

While editing the **Width**, **Height**, **X**, and **Y**, a rectangular indicator with the position and dimensions specified will appear on top of the selected object in the Universe view.

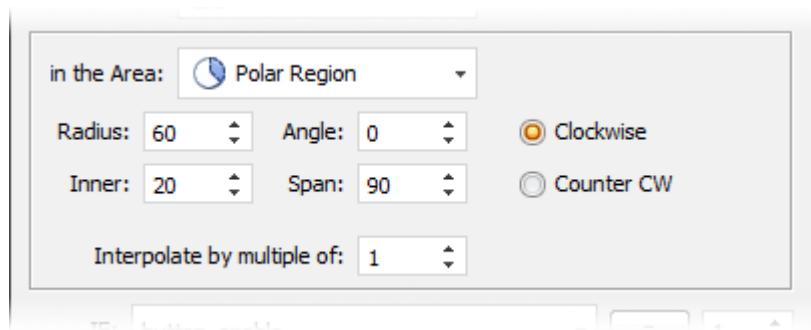


## Polar

Define a circular region or arc ("pie slice") by selecting the **Polar Region** stimulus type.

To change the size of the Stimulus Area, adjust the values for Radius, Inner, and/or Span. The **Radius** field defines the outer radius of the stimulus area (in pixels). The **Inner** field specifies the radius (in pixels) of a

“donut hole” in the center of the circular region that will *not* respond to input. **Span** defines the size of the arc’s angle (in degrees).



Adjust the value for **Angle** to change the offset angle in degrees (with respect to horizontal) of the circular region being defined. An illustration of these four polar variables is shown in **Figure 8.2** below.

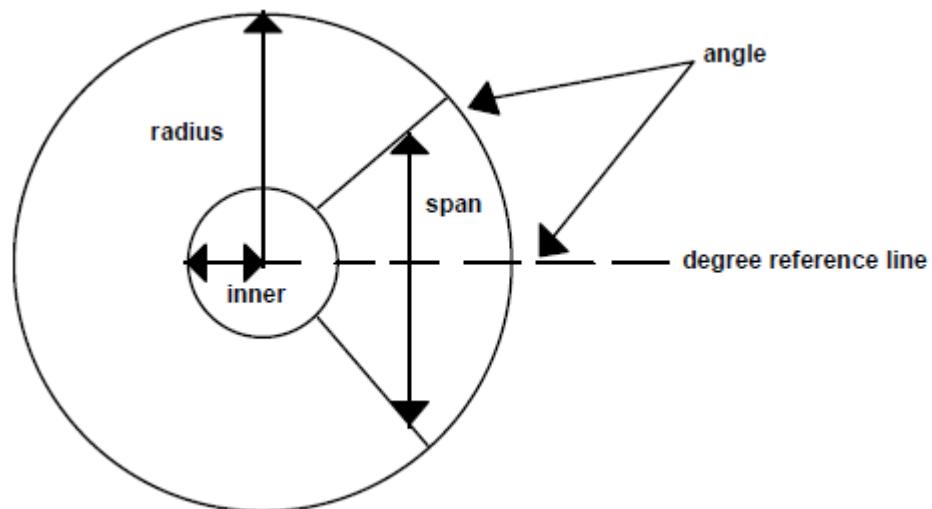
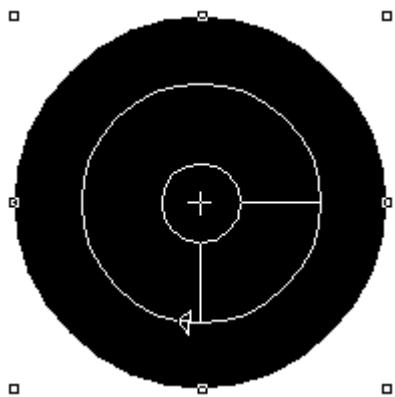


Figure 8-2: Polar Stimulus Area Specifications

Click on the Clockwise or Counterclockwise radio buttons to determine the direction of interpolation for polar areas. Interpolation will proceed from the area with the lowest state value to the area with the next highest state value (determined by the Interpolate by multiples of field) in the direction specified. For more information on input interpolation, please refer to [Section 4.7.2, Polar Area Interpolation](#).

While editing the **Radius**, **Inner**, **Angle**, and **Span** values, an indicator with the position and dimensions specified will appear on top of the selected object in the Universe view.

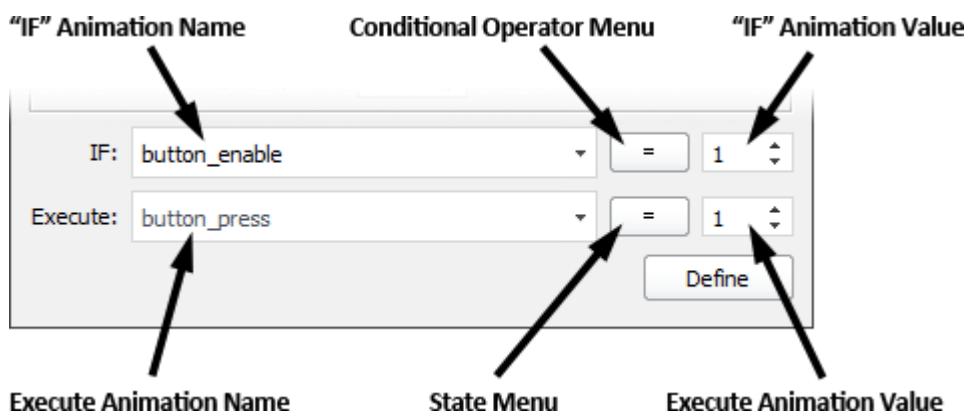


## Whole

Select **Whole** to make the entire object sensitive to input stimulus.

### 4.3.3 Defining Execute State and Enable Condition

The third step in defining stimulus is specifying the Execute State and Enable Condition.



## The IF Condition

Enable, or "IF" Conditions are limiting factors that must be true before an action specified in the Execute field can be performed. Although optional, an IF condition allows for more sophisticated stimulus definitions.

This option is most easily explained by imagining a knob that has the behavior of rotating when the mouse pointer is moved along its perimeter. Typically, this is not quite the desired behavior. A more useful knob could be created by requiring a left mouse button down stimulus on the knob to enable the

motion stimulus for the knob. Moreover, a left mouse button “release” could turn the enable off. This would ensure that the knob only rotates when the left mouse button is depressed.

## Setting the IF Condition

1. Enter the animation name for the condition (e.g., **knob\_enable**) in the **IF Animation Name** field. If no IF condition is required, do not enter a name (or delete the existing name) in this field. If no name exists in this field, the field will display "(undefined)" and no IF condition will be evaluated.
2. Click on the button between the **IF Animation Name** and **IF Animation Value** fields to display the **Conditional Operator** menu. Choose the desired operator condition (<, <=, =, >=, >, or <>) from the menu. This option allows you to choose how the indicated IF condition’s current state relates to the desired state value.
3. Enter a value in the **IF Animation Value** field.

## The Execute Statement

The final element of the stimulus definition is the animation or stimulus function to execute when the previous three conditions are met (input stimulus type in the area of sensitivity while the IF condition is true). The Execute Function is often used to change the state of an animation sequence (see [Chapter 3: Animation Editor](#)), to trigger a “callback” through Altia’s programmatic interface (see the [Altia API Reference Manual](#)), or to trigger a control WHEN block (see [Chapter 6: Control Editor](#)).

## Setting the Execute Statement

1. Enter an animation name that will be affected by the stimulus into the **Execute Animation Name** field. The animation name in this field can be any animation name used in your model (ex. the name of an enable condition used in another stimulus definition or a name used in Control code).
2. Click on the **State Menu** button to select the way in which the value for the Execute Animation Name will be changed. The choices are: set to a value (State); increase by a value (Inc+); or decrease by a value (Dec-).
3. Enter a value by which the Execute Animation Name will change or the value to which it will be set in the **Execute Animation Value** field.

Once all of the conditions are set, press the **Define** button to store the stimulus definition on the object.

## Multiple Definitions for the Same Input

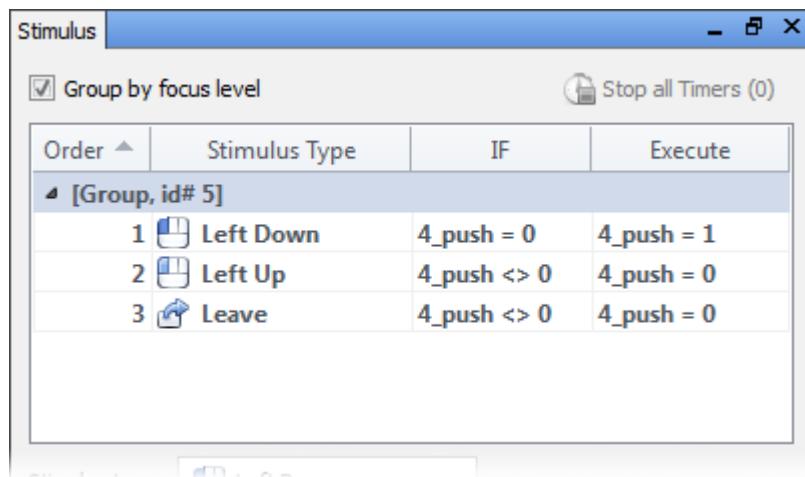
If multiple stimulus definitions respond to the same input, they are evaluated, in sequence, at the time of the input occurrence. All enable conditions are evaluated first and then all Execute statement statements are conditionally performed. If an Execute statement changes the state of a function that is used in one or more of the enable conditions, the change will not have an effect until the next input event.

## Multiple Objects with Similar Input Stimulus

When two or more objects have stimulus definitions that respond to the same input and the stimulus areas overlap, the enable conditions for all objects are evaluated first. All Execute statements are then conditionally performed based on the results of the enable condition evaluations starting with the object in front and finishing with the object in back.

## 4.4 Defined Stimulus List

The input stimulus entities that have already been defined for a selected object appear in a list at the top of the Stimulus Editor. Each line of this horizontally and vertically scrollable list contains information about one stimulus definition. The rows show all the conditions of the stimulus, the Execute Animation Name and State, and the IF condition Name and State. If no IF condition is defined then no value is displayed in the IF column.



The screenshot shows the Stimulus Editor window with a title bar 'Stimulus'. Below the title bar is a toolbar with a checkbox for 'Group by focus level' and a button for 'Stop all Timers (0)'. The main area is a table with four columns: 'Order', 'Stimulus Type', 'IF', and 'Execute'. The table has a header row and a data row labeled '[Group, id# 5]'. The data row contains three entries:

Order	Stimulus Type	IF	Execute
1	Left Down	4_push = 0	4_push = 1
2	Left Up	4_push <> 0	4_push = 0
3	Leave	4_push <> 0	4_push = 0

Note that there may be several definitions in the list because the selected object may have several areas/types of stimulus defined for it.

## Editing, Cutting, and Copying Stimulus

Selecting a line in the list populates all of the fields in the Stimulus Editor with its stimulus definition. This allows you to view the various properties of the definition and possibly redefine it (see the following Redefining Stimulus section).

Stimulus definitions may be deleted by selecting one or more lines in the list and choosing **Delete Selected Stimulus** from the Stimulus pane list's right-click context menu, or by pressing the **Delete (Del)** key.

To copy and paste stimulus (within the same or onto another object), choose the Copy Stimulus or Paste Stimulus menu option from the Stimulus pane list right-click context menu.

## Redefining Stimulus

Any portion of a stimulus definition can be redefined by selecting the definition from the list (which fills in all Stimulus Editor fields), changing the appropriate portion(s) of the definition (type of stimulus, area, enabler, execute function/action/state), then pressing the **Redefine** button (the Define button re-labels itself when an existing stimulus is selected from the list). All lines in the list area must be de-selected to add a new stimulus definition.

## 4.5 Renaming Stimulus

Choose Rename Animations... from the Stimulus pane list's right-click context menu to change one or more names used in previously defined stimuli. The Rename Animations window ([Figure X-X](#)) will appear, allowing you to enter a new name for any of the animation names used by the selected object's stimulus (and all of its children, if any).

**NOTE:** When you open the Rename Animations window, it actually shows you all animation, stimulus, and control names for the selected object (and its children if it is a group). This allows you to simultaneously change any or all names associated with the object.

## 4.6 Slider Example

The following example discusses stimuli for a typical slider bar.

With a mouse button LeftDown over the slider handle, the slider bar should start listening for mouse Motion events. With a Left Up over the slider handle, the slider should stop listening for motion events. Left mouse button events on the slider handle are enablers for the processing of mouse motion events for the slider bar (see [Figure 8-3](#)).

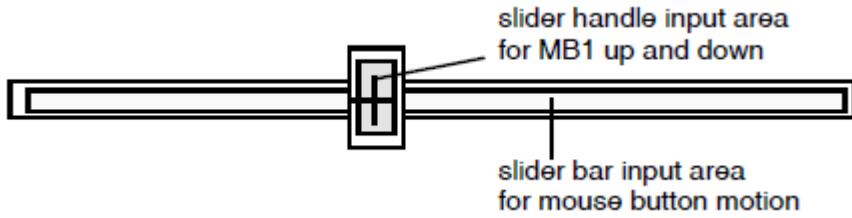


Figure 8-3: Slider Bar Example

A pseudo code representation for the slider handle's stimulus definitions might appear as follows:

```
If MB1 down and slider_enable == 0
then slider_enable 1

If MB1 up and slider_enable == 1
then slider_enable 0
```

**slider\_enable** is both the Enable Name and the Execute Name for the handle. Therefore, the definition for the slider bar could be described as follows:

```
If mouse motion and slider_enable==1
then slider 0 thru 20
```

**slider\_enable** is the enabler, and **slider** is the name of the animation to be executed. The state for **slider** is a value between 0 and 20, depending on the position of mouse motion over the slider bar.

The following would be defined for input stimulus.

Stimulus	AREA	Object	Enable	Execute
LeftButtonDown	whole	slider handle	slider_enable = 0	slider_enable 1
LeftButtonUp	whole	slider handle	slider_enable = 1	slider_enable 0
Motion	left	slider bar	slider_enable = 1	slider 0
Motion	right	slider bar	slider_enable = 1	slider 20

When creating input stimulus, you need to decide if you want an enabler and, if so, what (mouse/key) event should trigger the enable. In effect, you are saying to Altia, “this name (**slider\_enable**) has to be in this state (1), so that another action (**slider** animation with a state between 0 and 20) can occur.”

## 4.7 State Interpolation

As with animation, stimulus definitions can also be interpolated. The interpolation of stimulus automatically creates new Stimulus Areas and Execution States.

To complement the Animation Editor, the Stimulus Editor supports the use of floating point values in its Enable Condition and Execute fields. Hence, integer or floating point values can be interpolated. By using the Interpolate by multiple of field, you can specify the “step size” used to interpolate. For example, you can create a slider that changes value between 0 and 1.0 by multiples of 0.01.

The procedures for defining Rectangular and Polar input areas are similar, but each has its own type of input interpolation. Basically, you define endpoint areas from which in-between input areas will be automatically created (based on the chosen interpolation multiple).

For input interpolation to occur, you must define at least two distinct input areas of the same type (Motion, LeftDown, Enter, Leave, etc.) for an object with identical Execute names. Moreover, the range between the defined Execute State values must be larger than the interpolation multiple. For best results, it is recommended that you use an interpolation multiple that evenly divides into the difference between the two defined endpoints.

### 4.7.1 Rectangular Area Interpolation

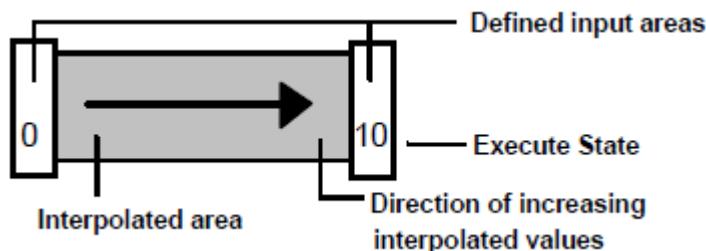


Figure 8-4: Rectangular Area Interpolation Example

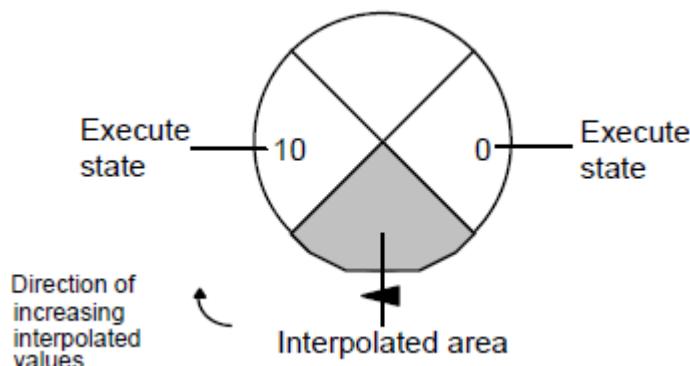
As shown in [Figure 8-4](#), the gray area will be the interpolated input area that will have Execute State values 1-9 (assuming that the interpolation multiple is chosen as 1.0), depending on the location of the stimulus within the area.

### State Interpolation Example

1. In the Editor view, create a long, horizontal rectangle (leave it selected).
2. Open the Stimulus pane and set the input stimulus to **Motion**.
3. Create a rectangular area of sensitivity the same height as the rectangle at the left end of the rectangle (adjust **Height:** and **X:**).
4. Set the Execute function to **test** and the execute state to **0**.

5. Press the **Define** button to define an area at the left end of the rectangle that is sensitive to motion.
6. Increment the execute state to **10**.
7. Move the rectangular outline of the area of sensitivity to the right end of the rectangle by incrementing the **X:** value.
8. Press the **Define** button to define an area at the right end of the rectangle that is sensitive to motion.
9. Open the Animation pane and enter the name **test** in the **Animation Name** field. The Animation pane will be used to monitor state changes for the **test** animation.
10. Put the Editor view in Run mode by selecting the **Run** button. In this mode, all keyboard and mouse events are passed directly to objects - the Altia Design interface ignores them.
11. Move the mouse pointer along the length of the rectangle and observe the state changes that occur to the function test in the Animation Editor. While only states **0** and **10** were explicitly defined, states 1-9 have been interpolated.

#### 4.7.2 Polar Area Interpolation



**Figure 8-5: Polar Area Interpolation Example**

As shown in *Figure 8-5*, the interpolated area will have Execute State values of 1-9 (assuming that the interpolation multiple is chosen as 1.0), depending on the angle of the stimulus location within the interpolated area.

**NOTE: Whole Stimulus Areas do not support interpolation because there is only one Stimulus Area: the whole object.**

## 4.8 Timer Stimulus

A timer stimulus is tied to a specific object and executes animation state changes at regular intervals, which are user-defined. As with mouse or keyboard event stimuli, a timer stimulus specifies a response to an event. However, with timer stimulus, that event is not a direct user action.

When creating timer stimulus, five different aspects must be considered:

- Timer Interval
- Start Condition
- Stop Condition
- Enable Condition (optional)
- Execution

The event which starts the execution of a timer stimulus definition is a state change in an animation or stimulus function that satisfies the Start Condition. In a typical scenario, this state change would actually result from the execution of a conventional stimulus definition for a mouse or keyboard action, control code, or programmatically using the Altia API.

Once a timer stimulus is started, it is evaluated at a specified millisecond, second, or minute Timer Interval. If an IF Condition is specified, the timer will evaluate the state value of the IF Condition name at each interval and perform the Execute Statement only if the IF condition is satisfied.

The timer stimulus will stop when the animation or stimulus function state change specified by the Stop Condition occurs. This state change may be the result of the timer's own execution or it could be triggered by any other method of setting the stop condition's stimulus/animation function (programmatically, using control code, conventional stimulus, etc.).

### Timer Definitions and the List Area

As timer definitions are added to the object, they appear in the Stimulus Editor's list area. If the object has any mouse or keyboard stimulus definitions, they will appear first in the list with timer definitions appearing afterward.

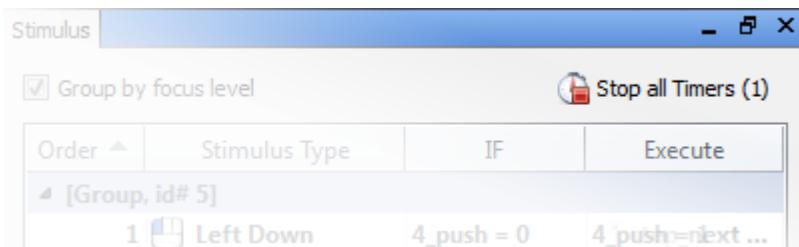
Each timer definition is assigned an index starting with 1 (i.e., you will see **Timer (1)** at the beginning of the first timer definition shown in the list area). If two or more timer definitions have identical **Interval** (Timer event every: field), **Start** (Start when:), and **Stop** (Stop when:) statements, they will all be assigned the same index number. This is a visual clue to the user that the timer definitions are all defined to start on the same event, evaluate at the same interval (in the same sequence that they appear in the list), and stop on the same event.

**NOTE:** When a set of timer definitions have the same index number, they are evaluated, in sequence, at the same time intervals. All Enable Conditions are evaluated first and then all Execute Statements are conditionally performed. If one or more of the Execute Statements change a function's state which would stop all timers,

the remaining Execute Statements are still performed. Also, if an Execute Statement changes the state of a function that is used in one or more Enable Conditions, the change will not have an effect until the next timer interval.

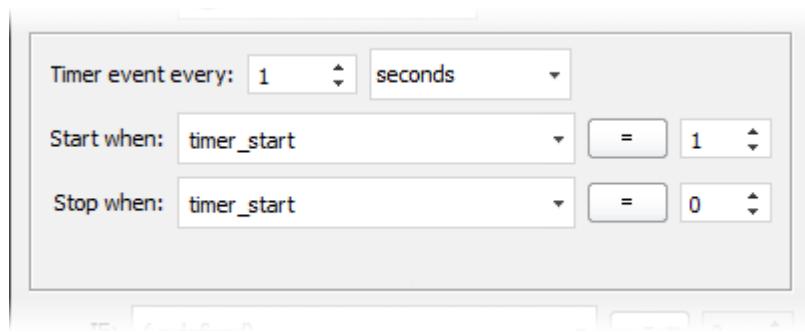
## Runaway Timers

While developing timer definitions, it is not uncommon to make a mistake and set off a timer or sequence of timers which cannot be easily stopped. If this happens, simply click the **Stop All Timers** button on the Stimulus pane. This will stop all active timers immediately for all objects in the design.



### 4.8.1 Creating Timer Stimulus

To create timer stimulus for an object you have selected in the Graphics Editor, open the Stimulus Editor and click the left mouse button on the stopwatch icon. The Stimulus Editor is now in timer stimulus definition mode as indicated by the appearance of the word Timer under the WHEN THE INPUT STIMULUS IS heading. When in timer stimulus definition mode, the Stimulus Area section looks like the figure below.



The body of the Stimulus Editor now contains five sections that, when completed, specify the entire timer definition. You may now begin defining your timer stimulus using the following procedure.

1. **Select the Timer Interval:** The timer interval describes how often the timer definition's **IF** and **Execute** statements should be evaluated after the timer is started. For example, entering a value of "500 msecs" in the **Timer event every:** fields would trigger evaluation 1/2 second after the timer starts and it would continue to repeat the evaluation at 1/2 second intervals until the timer stops.

Enter a non-negative integer in the field next to **Timer event every:** by toggling the increment/decrement buttons or by typing in a number. Then choose a unit of time from the pulldown menu. There are three options: **milliseconds**, **seconds**, and **minutes**.

2. **Set the Start Condition:** A timer waits for a specified animation, stimulus, or control function name state change to occur before it will begin evaluating its “enable” and “execute” statements at the prescribed interval.

In the field following **Start when:** type the appropriate animation, stimulus, or control function name, then choose an operator condition from the pulldown menu (initially labeled =), then enter a state value.

**NOTE:** If the state value for a function meets the START statement conditions when a design is loaded, the timer will not automatically start. A state change resulting from a mouse/keyboard event, control block SET or MATH statement, or from a client application program is necessary for a timer to start. Starting a timer when a design is loaded, however, is possible by setting the START statement to `altialInitDesign = 0`.

3. **Set the Stop Condition:** As with starting the timer, the occurrence of a particular state change is used to stop it. When a timer is stopped, it discontinues the periodic evaluation of its “IF” and “execute” statements.

In the field following **Stop when:** type the appropriate animation, stimulus, or control function name, then choose an operator condition, and enter a state value.

**NOTE:** If the same state change is used to start and stop the timer, it will never start. Be sure to distinguish start from stop by using different names, operator conditions, and/or state values.

4. **Set an Enable Condition (Optional):** The enable, or “IF” condition provides a method for testing the current state for any animation or stimulus function and ignoring the “execute” statement if the current value of the function is not satisfied. For timers, the IF condition is evaluated after each time interval expires.

If you wish to set an IF condition, complete the **IF:** section by typing in the appropriate animation, stimulus, or control function name, choosing an operator condition, and entering a state value. If you do not want to utilize an IF Condition, delete the name in the animation name field.

5. **Set the Execute Statement:** The final element of the timer stimulus definition is the animation, stimulus, or control function name to execute at each timer interval if the enable condition is satisfied.

Specify the state change you want executed by completing the last section, **Execute:**. Once

again, type in the appropriate animation, stimulus, or control function name, choose an option from the pulldown menu (this time, either **Inc+**, **Dec-**, or **State**), and enter a state value.

6. Once the five statements have been satisfactorily edited, push the **Define** button to add the timer stimulus definition to the object.

Please note that, as with all types of stimulus, an existing timer definition can be edited by highlighting it in the list area, making adjustments to one or more of the five statements while it remains highlighted, and pressing the **Redefine** button (which was previously the Define button) to apply the changes to the highlighted definition. For more details on editing, copying, and deleting stimulus definitions, see

[\*\*Section 4.4, Defined Stimulus List.\*\*](#)

## A Simple Flashing Example

Assume that an object exists which is animated to change color. Its animation function is named **Flash** and it has two states (0 and 1). Also assume that a button exists that has left button down stimulus which toggles the stimulus function **StartFlash** between states 0 and 1.

The following shows the timer stimulus definitions (there are 2) required to continuously change **Flash** between 0 and 1 at 1 second intervals while **StartFlash** is in state 1:

### Definition #1

Timer event every:	<b>1</b>	<b>seconds</b>
Start when:	<b>StartFlash</b>	<b>= 1</b>
Stop when:	<b>StartFlash</b>	<b>= 0</b>
IF:	<b>Flash</b>	<b>= 0</b>
Execute:	<b>Flash</b>	<b>= 1</b>

### Definition #2

Timer event every:	<b>1</b>	<b>seconds</b>
Start when:	<b>StartFlash</b>	<b>= 1</b>
Stop when:	<b>StartFlash</b>	<b>= 0</b>
IF:	<b>Flash</b>	<b>= 1</b>
Execute:	<b>Flash</b>	<b>= 0</b>

These 2 definitions could be added to the button object or the object that is changing color – it's up to the designer. As a helpful hint, it is sometimes best to put the timer stimulus on the object that is affected by it rather than on the object that triggers it. This way, the timer stimulus is copied when the object is copied and the method used to trigger the timers can be changed quite easily (e.g., a toggle button is quickly replaced with a switch).

When the 2 definitions are created, you will notice that they are grouped together with the same timer index (index 1 if these are the only timer stimulus definitions). As was discussed before, this is because the 2 definitions have the same **Timer event every:**, **Start when:**, and **Stop when:** statements. This grouping is also very important for ensuring reliable behavior. The definitions are evaluated at exactly the same time as each interval expires. This ensures that the flashing is well synchronized.

## Finding More Timer Stimulus Examples

A Models Library of timer stimulus examples is available to help users get acquainted with this powerful feature. To display the examples in a Models View window, open the **Model Libraries** gallery dropdown on the **Insert** ribbon. In the **Special Functions** category, click on **Timers**.

A Models View window will appear containing the various timer stimulus examples. When you see an example that looks interesting, press the left mouse button over it and drag it into the Editor's drawing area.

Try out the example by putting the Editor view into **Run** mode and clicking on the switch, knob, or buttons associated with the example. To see an object's stimulus, switch to **Edit** mode and select the object. Most of the examples have the timer stimulus attached to the example's group object. In a few cases, you may need to click on **Focus In** and select one or more of the subcomponents of the group to see the timer stimulus definitions.

Definitions can be easily copied from these examples to your own objects using the **Copy Stimulus** and **Paste Stimulus** options from the Stimulus pane list's right-click context menu.

## 4.9 Testing a Stimulus Definition

To test the stimulus definition, the Graphics Editor is placed into Run mode. The stimulus from the mouse is then directly applied to the objects.



You can also use the Animation pane to monitor the effects of the stimulus.

1. Open or activate the Animation pane.
2. In the **Animation** name field of the Animation pane, type the stimulus definition's **IF** or **Enable** name.
3. Place the Editor view in run mode by selecting the **Run** button.
4. Directly apply the stimulus to the object and notice the change in the **State** field of the Animation pane.

# Chapter 5: Properties Editor

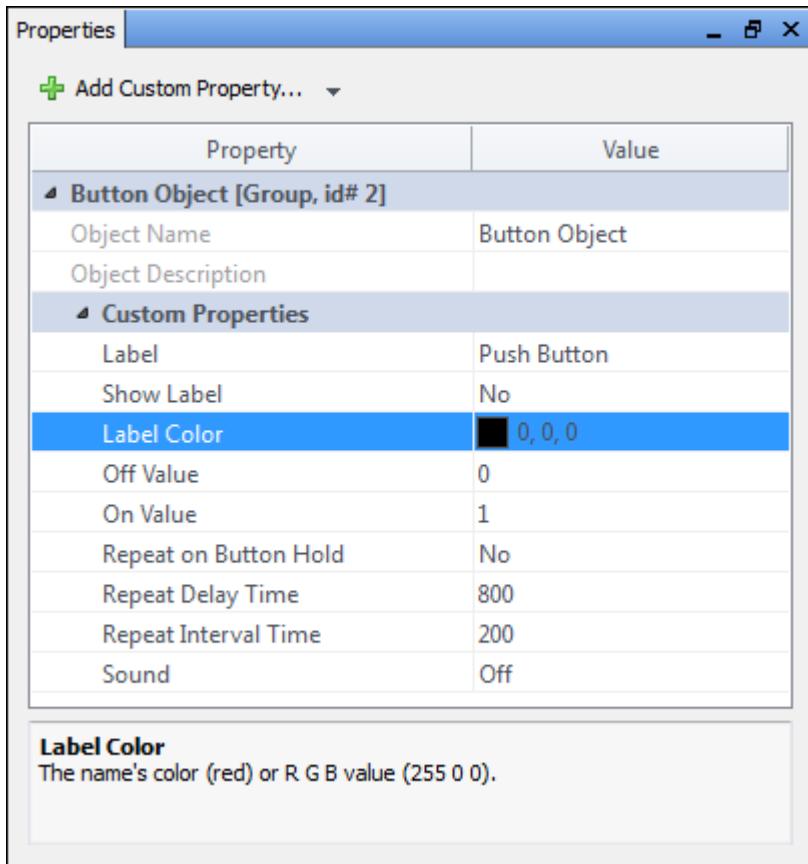
Properties allow users to adjust the animation, stimulus, and control of Altia objects without having to understand the process that went into creating them. When carefully considered and implemented, properties can make even the simplest objects extremely flexible or the most complex object behaviors clear and simple.

An object with properties can be a primitive object (e.g., a rectangle) or a group containing many child objects that have their own animation, stimulus, or control. In the case of a group, properties can manage all of the child objects' behaviors.

## 5.1 The Property Pane

Properties are accessed via the Property pane. A menu button allows you to add new properties and edit current ones. A list displays the properties belonging to the currently selected object. When you select a property in the list, its description is displayed at the bottom of the pane.

If the properties pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Properties** menu option is checked. Alternatively, you can use the keyboard shortcut, **Ctrl+Shift+P**.



### 5.1.1 The Property List

The property list displays all properties that belong to the current selected objects. An object's properties are listed under their respective object group item. The object group item displays the object name if defined, and object type. It also displays the object's unique id.

### 5.1.2 Standard Properties

These optional properties are listed at the top of each object group item in the property list. Because they are standard for any object, you can only change their value.

- **Object Name** - This property is always shown. It is the name of the object and will be displayed in various lists such as the property list and Navigator to easily identify the object.
- **Object Description** - Always shown, this property may be used to describe the object in more detail.
- **Object Layer** - This property assigns the object to a hardware display layer. It is not shown by default. You can create this property using the "Add Hardware Layer Property" item of the dropdown menu button or the context menu.

- **Raster Cache State** - This property assigns the raster cache state policy for a Group, Raster, or Image Object. This property is not shown by default. You can create it using the "**Add Resource Property**" -> "**Raster Cache State**" item of the **Add Custom Property** dropdown menu button.
- **Raster Compress State** - This property assigns the raster compress state policy for a Group, Raster, or Image Object. This property is not shown by default. You can create it using the "**Add Resource Property**" -> "**Raster Compress State**" item of the **Add Custom Property** dropdown menu button.
- **DeepScreen Optimize** - This property allows you to override global DeepScreen memory optimization settings to specifically include or exclude an object and its children from memory optimization in DeepScreen generated code. This property is not shown by default. You can create it using the "**Add DeepScreen Property**" -> "**DeepScreen Optimize**" item of the **Add Custom Property** dropdown menu button.

**NOTE:** Adding a DeepScreen Optimize property for an object and setting it to "Do not optimize" is typically only used under special circumstances with Text I/O objects that need specific dynamic behaviors controlled by external application code or control logic.

For detailed information on DeepScreen optimization settings, refer to the DeepScreen User's Guide and your target's DeepScreen Target User's Guide.

### 5.1.3 Custom Properties

These properties are ones you create specifically for the object. See the [Property Edit Dialog](#) section for a description of how to create custom properties. Library objects that you drag into your design will have custom properties pre-defined.

### 5.1.4 Changing Property Value

The property list displays the property's unique name and its value. To change a value, click it in the list. This will bring up the inline editor allowing you to change the value. The type of editor depends on the value type.

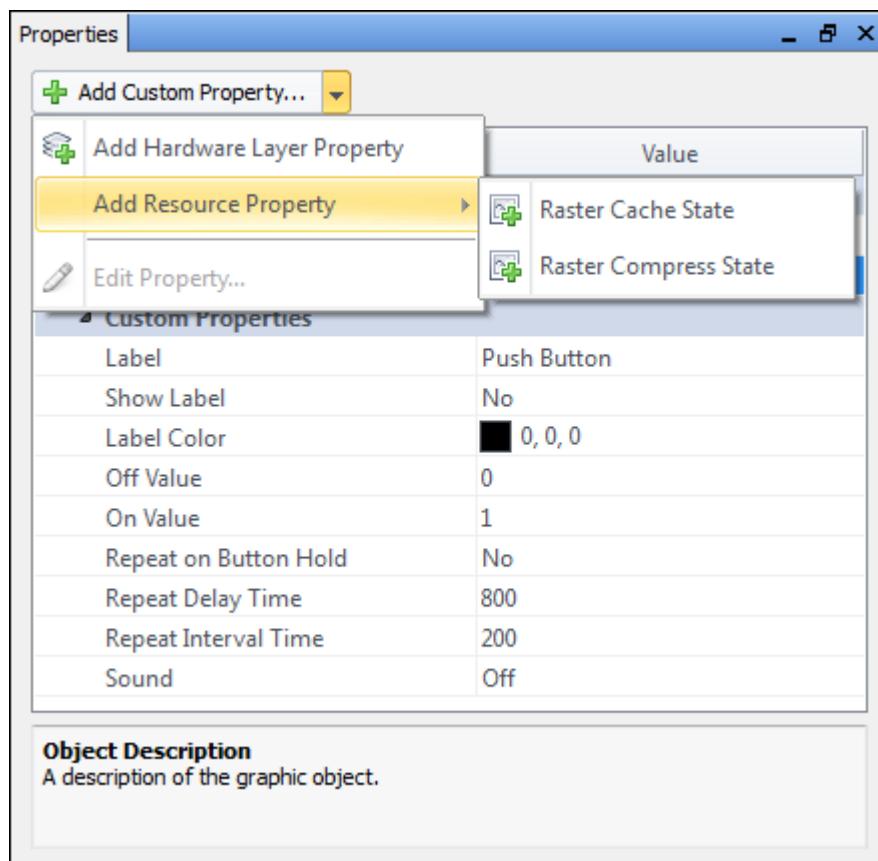
- **Numbers** - A spinner widget allows you to enter a numeric value directly using the keyboard. The up and down buttons allow you to increment or decrement the value. The mouse wheel allows you to do this quickly.
- **Text** - A line edit widget allows you to enter a value using the keyboard. You can press the "..." button to bring up the text edit dialog so longer strings may be entered with greater ease.
- **Lists** - A dropdown widget allows you to select a pre-defined value. Optionally, certain list types will allow you to enter a custom value. See the [Property Input Tab](#) section for a description of how to define such a list.

- **Colors** - A color edit widget allows you to enter color values in three different formats. For example, you may enter the color red as (without quotes): "255,0,0", "#ff0000", or "red". Alternatively, you can press the drop down button to pick colors from a menu.
- **Filenames** - A line edit widget allows you to enter a path to a filename. You can press the "..." button to bring up the file picker dialog. If your design has been saved, the path to the file will be converted to a path relative to your design's directory. If not, an absolute path will be recorded. Once you have saved your design, you may then convert your absolute path to a relative path by selecting "**Change absolute to relative path**" in the context menu.

## 5.1.5 Changing Property Names Inline

By double-clicking a name in the property list, you can change the property name without using the [Property Edit Dialog](#). You must enter a property name that hasn't been used yet in the object. If you do not, an error message is displayed and the name is not changed.

## 5.1.6 The Menu Button



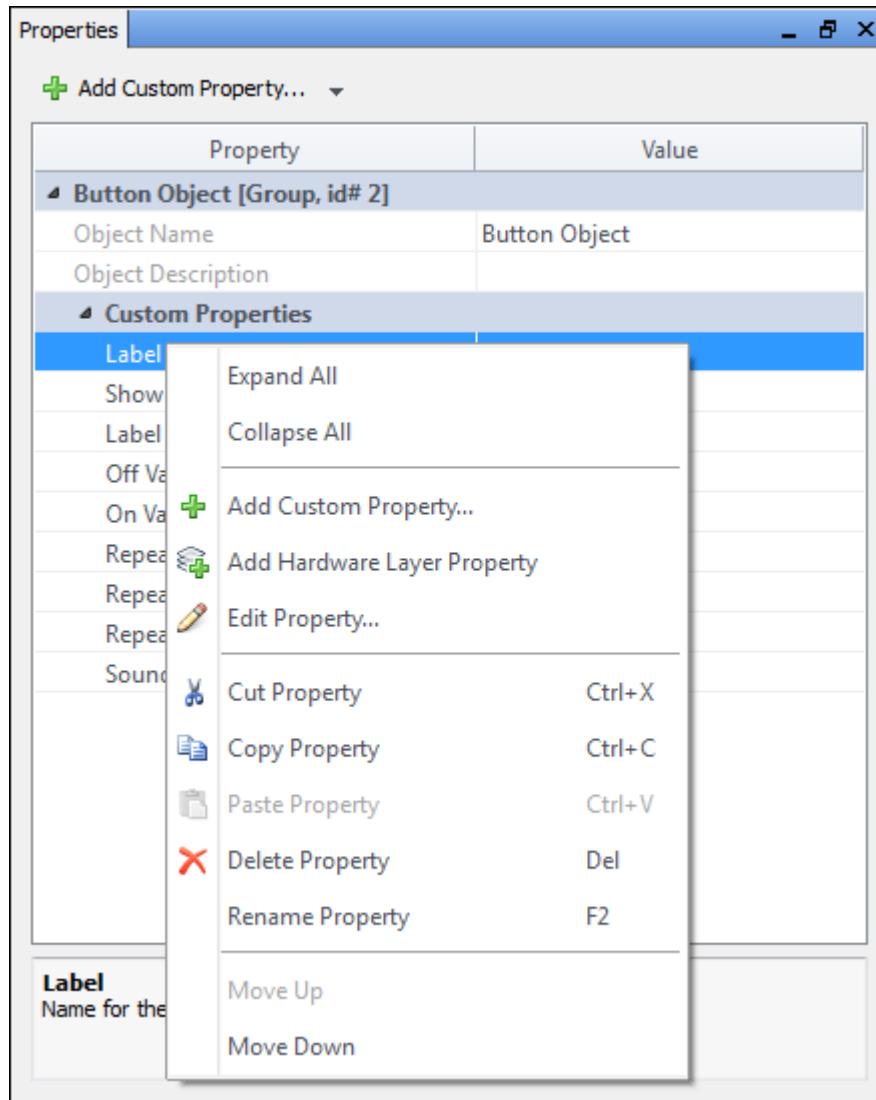
When pressed, the "**Add Custom Property...**" menu button displays the [Property Edit Dialog](#) so you can create a new custom property.

The dropdown portion of the menu button displays these menu items.

- **Add Hardware Layer Property** - Create the "Object Layer" standard property.
- **Add Resource Property -> Raster Cache State** - Create the "Raster Cache State" standard property.
- **Add Resource Property -> Raster Compress State** - Create the "Raster Compress State" standard property.
- **Add DeepScreen Property" -> "DeepScreen Optimize** - Create the "DeepScreen Optimize" standard property.
- **Edit Property...** - Display the [Property Edit Dialog](#) so you can edit a selected custom property.

## 5.1.7 The Context Menu

You can right-click in the property list to display the context menu. Depending on what you select in the property list, certain menu items may be disabled.

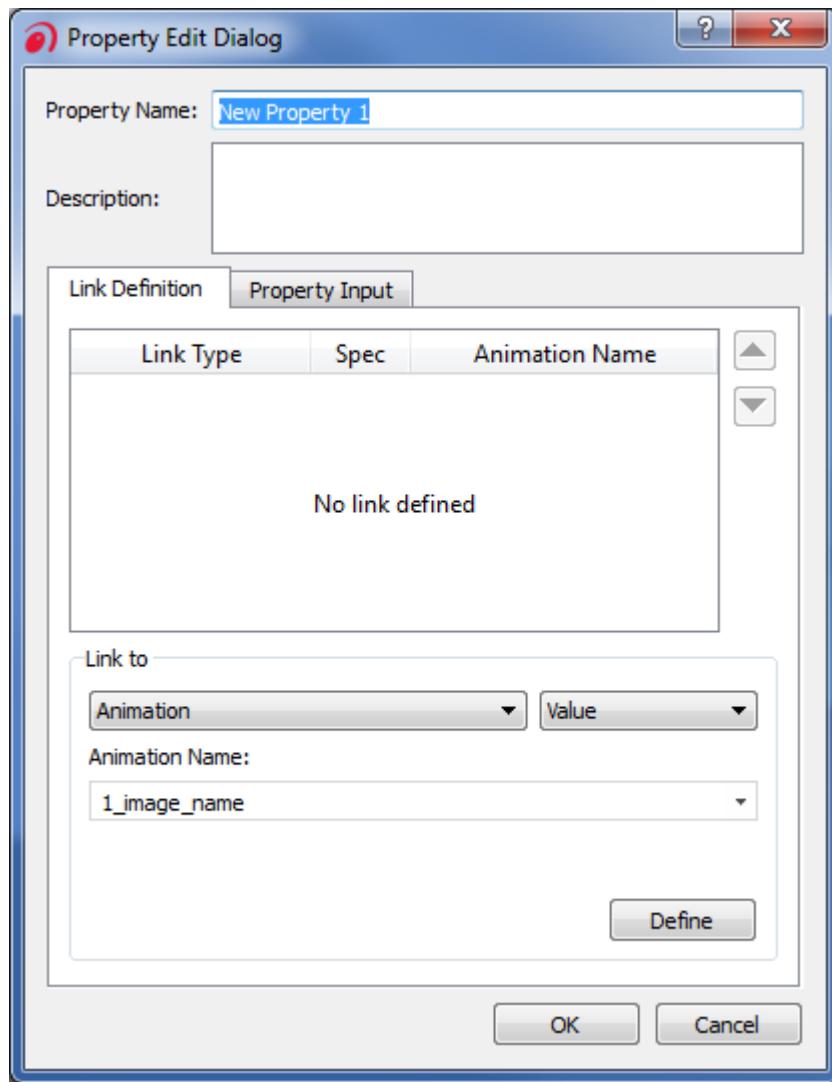


- **Expand All** - Expand to show all items in the property list.
- **Collapse All** - Collapse to hide all items in the property list.
- **Add Custom Property...** - Displays the [Property Edit Dialog](#) so you can create a new custom property.
- **Add Hardware Layer Property** - Create the "Object Layer" standard property.
- **Edit Property...** - Display the [Property Edit Dialog](#) so you can edit a selected custom property.
- **Cut Property** - Remove the selected property. You can paste it back later.
- **Copy Property** - Place the selected property in the copy buffer. You can paste it back later.

- **Paste Property** - Paste the cut or copied property below the currently selected item. The name will be modified to make it unique.
- **Delete Property** - Remove the selected property. You can paste it back later.
- **Rename Property** - Rename the selected property inline.
- **Move Up** - Move the selected property up one position in the list.
- **Move Down** - Move the selected property down one position in the list.

## 5.2 Property Edit Dialog

The property edit dialog allows users to create new custom properties or edit existing ones. It can be brought up through the property menu button or context menu. When creating a new property, the dialog is prefilled with only a suggested unique name. The rest of the fields in the dialog are empty waiting for the user to fill out. When used to edit a current property, the fields in the dialog are filled with data from the selected property.



### 5.2.1 Property Name and Description

The property name uniquely identifies the property. It is pre-filled with a suggested name when creating a new custom property. When the name matches a currently used custom property name or a standard property name, the text will turn red and a tool tip will describe error. If the name is not unique or is empty, the **OK** button will be disabled and the property cannot be created.

With the property description, you can provide details about the property. Some example uses might be to describe what the property does, what the values mean, or how it affects other objects in the design. This description will be displayed at the bottom of the property pane when a property is selected in the property list.

## 5.2.2 Link Definition Tab

What the property does is defined using the link definition tab. A list shows existing link specifications. A dropdown allows the user to choose a link specification type. Depending on the specification type, various options are available. Once the options are selected, press the **Define** button to create the link specification.

More than one link specifications may be created for a property. A property value change is issued to all specifications in the order they are listed. You can change the order by selecting the specification and pressing either the **Up** or **Down** arrow button.

You can create a property without a link specification but you will be presented with a warning.

### Context Menu

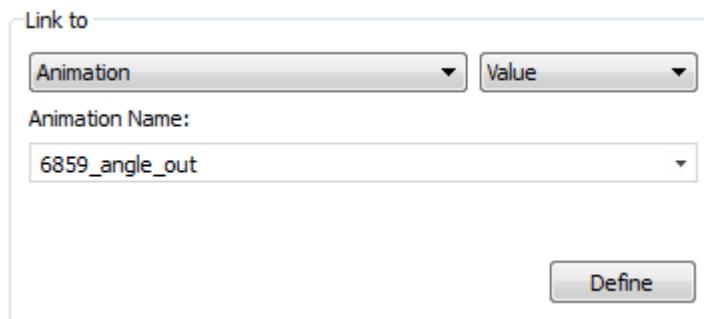
The link specification list context menu contains these actions:

- **Cut Link Definition** - Remove the selected specification. You can paste it back later.
- **Copy Link Definition** - Place the selected specification in the copy buffer. You can paste it back later.
- **Paste Link Definition** - Paste the cut or copied specification below the currently selected item.
- **Delete Link Definition** - Remove the selected specification. You can paste it back later.
- **Move Up** - Move the selected specification up one position.
- **Move Down** - Move the selected specification down one position.

### Animation Specification Type

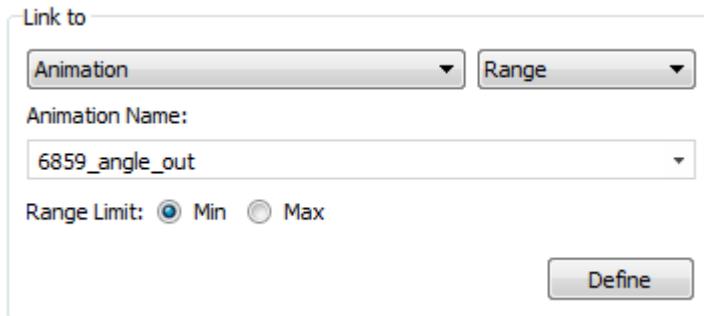
The Animation Specification Type allows you to manipulate animations on your objects through properties in numerous ways. A secondary dropdown to the right gives you this choice.

- **Value** - The Value option lets you vary an animation's current state value through a property.



- **Range** - The Range option lets you vary an animation sequence's Min or Max value through a property.

For example, the animation Slide on an object has a Low defined state of 0 and a High defined state of 100. A Property Specification called Min Value of type Range(Min) for Slide has been created for the object. When a user enters a value of 12 into the Min Value property, the Slide animation will now have a Low defined state of 12 and a High defined state of 100. The physical location and behavior of the object for its Low state have not changed, but the value of the low state has. We have adjusted its Range.



- **State** - The State option allows you to redefine the value of an object animation's explicitly defined state (without regard for its previous numerical order) through a property.



- **Initial** - The Initial option lets you vary the Initial Value of an object's animation through a property.

Link to

Animation	Initial
Animation Name:	<input type="text" value="6859_angle_out"/>
<b>Define</b>	

## Stimulus Specification Type

The Stimulus Specification Type allows you to manipulate stimulus definitions on your objects.

- **Enable** - The Enable option lets you vary an Enable Condition's compare value through a Property.

Link to

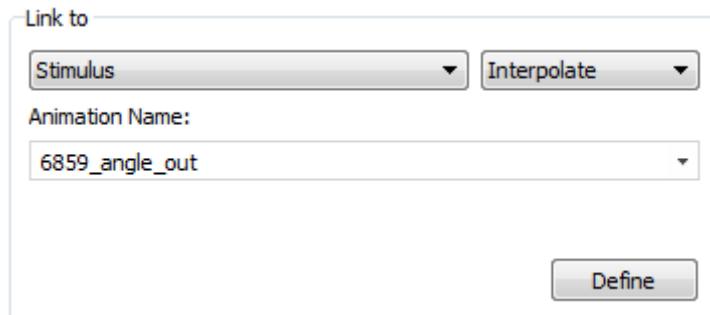
Stimulus	Enable
Animation Name:	State Value:
<input type="text" value="6859_angle_out"/>	<input type="text" value="0"/>
<b>Define</b>	

- **Execute** - The Execute option lets you vary the Execute value of a Stimulus definition through a Property.

Link to

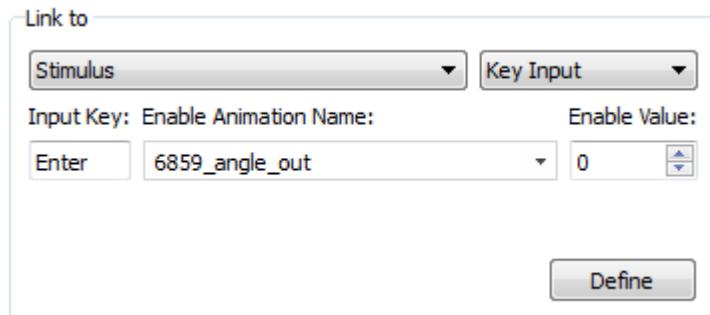
Stimulus	Execute
Animation Name:	State Value:
<input type="text" value="6859_angle_out"/>	<input type="text" value="0"/>
<b>Define</b>	

- **Interpolate** - The Interpolate option lets you set the step value for interpolated Stimulus definitions based on a Property value.

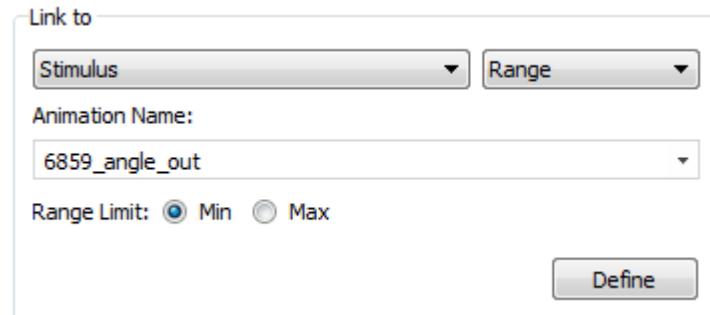


- **Key Input** - The Key Input option is applicable only to KeyPress or KeyRelease stimulus definitions. This property allows you to change which key triggers the stimulus. Non-printable keys are supported with the following pre-defined names: **Enter, Space, Esc, Insert, Delete, Home, End, Page Up, Page Down, Left Arrow, Up Arrow, Right Arrow, Down Arrow, and Backspace**.

For a demonstration of the Key Input property, please see the **hotkeys/hotkeys.dsn** file in the Altia software demos directory.



- **Range** - The Range option, like the Range option for Animation Property Specifications, allows for adjusting the Min or Max defined Execute values on interpolated Stimulus.



## Timer Specification Type

The Timer Specification Type allows you to manipulate Stimulus timer definitions on your objects.

- **Enable** - The Enable option lets you vary an Enable Condition's compare value through a Property.

Link to

Timer  Enable

Animation Name:  State Value:

- **Execute** - The Execute option lets you vary the Execute value of a Timer Stimulus definition through a Property.

Link to

Timer  Execute

Animation Name:  State Value:

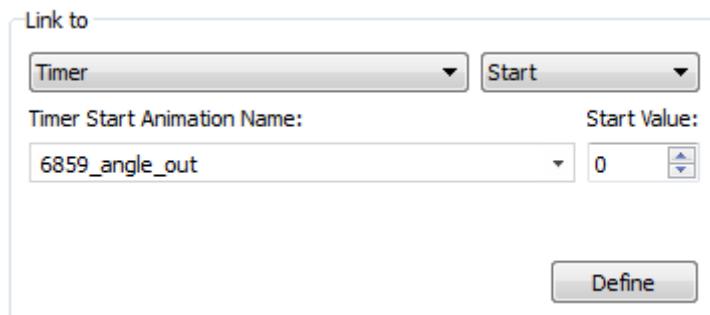
- **Interval** - The Interval option lets you vary the interval of a Timer Stimulus definition through a Property.

Link to

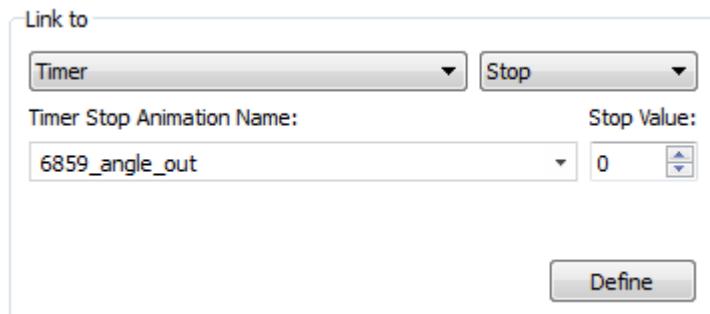
Timer  Interval

Timer Start Animation Name:  Start Value:

- **Start** - The Start option lets you vary the value of the Start Condition of a Timer Stimulus definition through a Property.



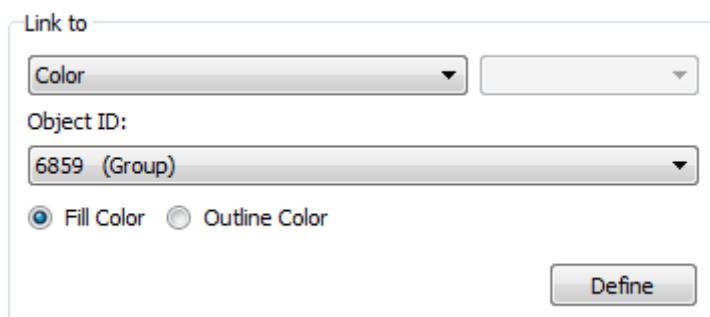
- **Stop** - The Stop option lets you vary the value of the Stop Condition of a Timer Stimulus definition through a Property.



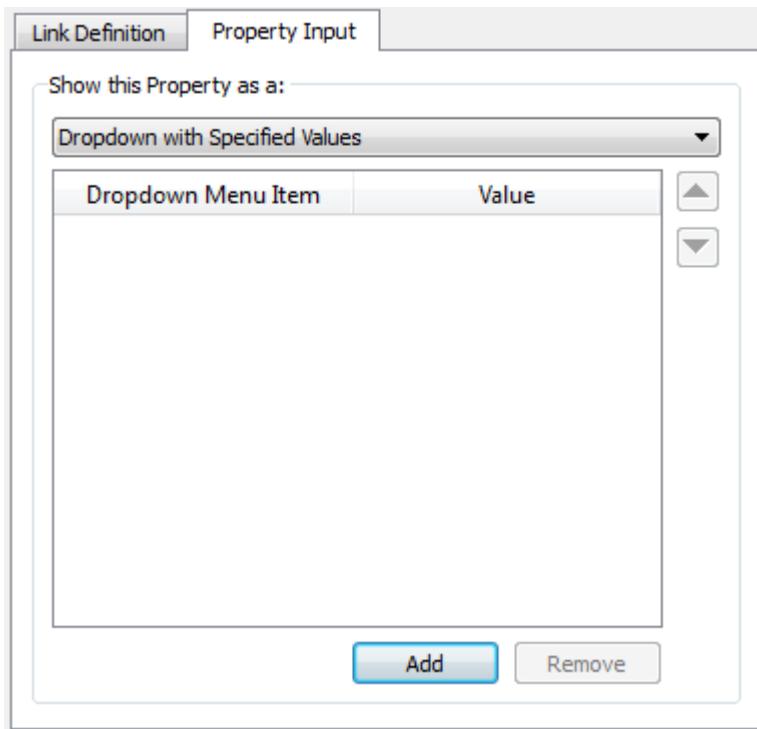
## Color Specification Type

The color of an object can be set through a Property.

The unique objectID for the object must be used, and objects with animations may override a color set with a Color Property definition.



### 5.2.3 Property Input Tab



How property values are displayed and changed can be defined using the property input tab. There are several display options.

- **Default** - The value type is depended on the first link specification.
- **Dropdown** - A list of values are defined and displayed as the property's values.
- **Dropdown Editable** - Similar to the Dropdown display option, but this option allows the user to enter a value not in the list.
- **Dropdown with Specified Values** - Similar to the Dropdown display option, but the displayed values are mapped to actual values.

Press the **Add** button to create a new item in the display list. You can immediately enter the value of the list item. Press the **Remove** button to remove a selected list item. To change the order of the list items, select an item and press the **Up** or **Down** arrow button. The context menu allows you to **Cut**, **Copy**, and **Paste** items.

## 5.3 Learning More About Properties

The standard models library (which you can view by choosing the **Insert** ribbon) is home to the basic Altia component libraries. These libraries contain full featured reusable components which have properties that allow you to customize their behavior. As such, they are an excellent resource for those who want to learn how to design advanced properties.

As an experiment, drag one of these library components (e.g., a slider) into the Altia graphic editor, select it, and press **Ctrl-Shift-P** to display the Property Pane. The pane will show you all of the properties for the selected component. By using the Property Edit Dialog, examine the properties of components like a slider and a meter, much can be discovered about the proper way to implement properties.

# Chapter 6: Control Editor

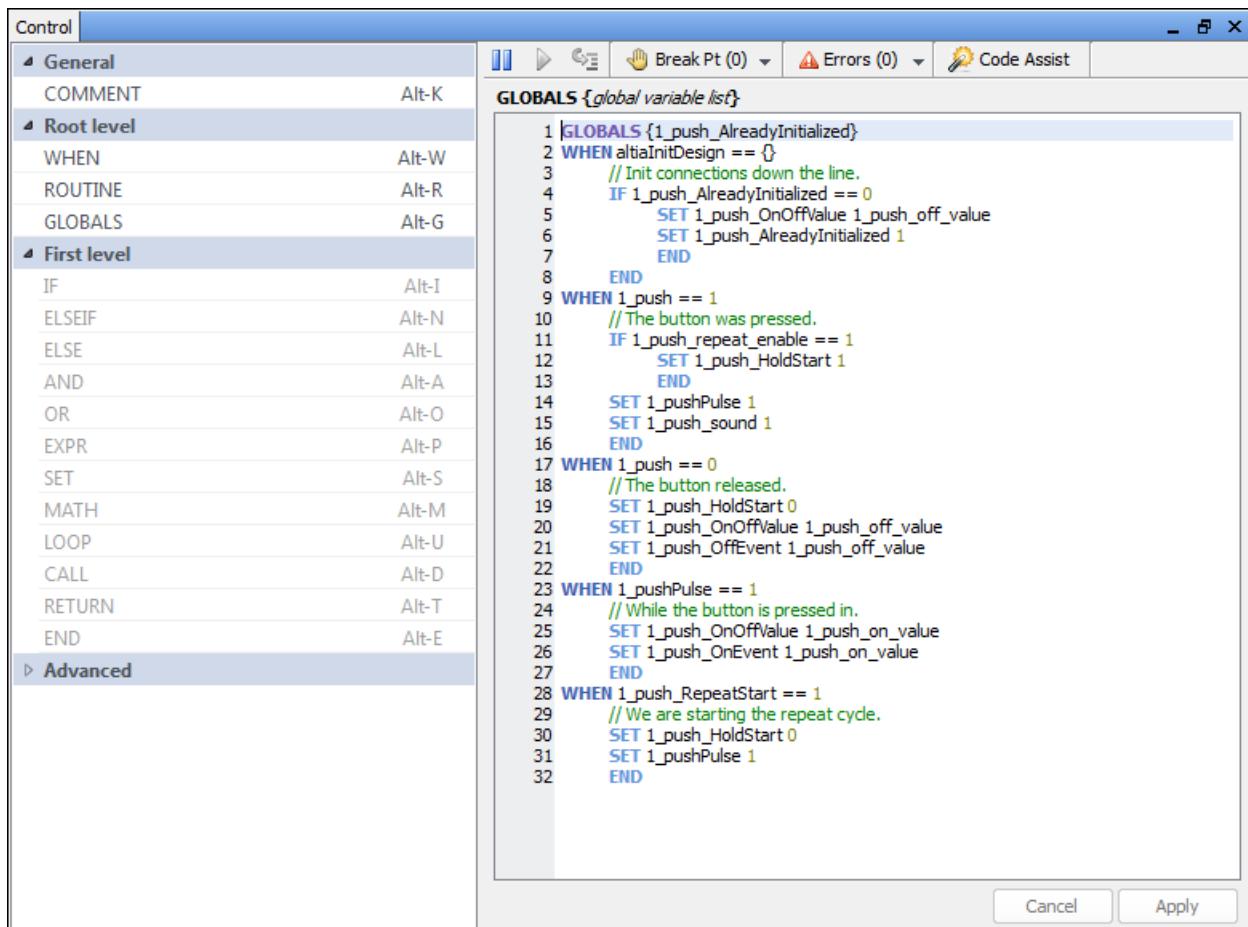
Altia Design's Control Editor may be used to specify sophisticated logic that governs how Altia components behave. While simple cases such as a flashing light are readily handled by the Stimulus Editor, more involved scenarios may be created with the Control Editor.

Like animation and stimulus, control is attached to objects. Control is displayed for the object currently selected in the Graphics Editor. Whenever you cut, copy, move, or paste an object, all of the animation, stimulus, and control statements that have been assigned to that object also get cut, copied, moved, or pasted.

Control code is event driven and can use animation and stimulus previously defined in your designs. It allows you to specify what to do when certain events happen. The events can be any animation or stimulus function name, originating from user input, Altia's code connection, or the Control Editor itself. The resulting action ("what to do") could be setting the value of an animation/stimulus function name; reading/writing file information; or making more complicated logical decisions based on other animation/stimulus/control name values.

The Control Editor is a free-form text editor. Altia uses a set of control code statements (see [Section 6.6, Control Code Statements](#)) which follow a specified syntax. The Control Editor provides a **Code Assist** feature so there is no need to learn and remember the language syntax. It guides you through the entire process by only allowing the addition of syntactically valid statements. All you need to do is fill in the necessary animation/stimulus/control names, values, and operators inside the statements.

If the Control pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Control** menu option is checked.



## 6.1 Control Code Overview

The simplest element of control code is a statement, which can be one of several types, as described in [Section 6.6, Control Code Statements](#).

The Control Editor is used to organize multiple statements into control blocks, which allow for sophisticated manipulation of animation. These control blocks provide a level of logic beyond that of stimulus definitions.

A control block always begins with a WHEN statement to specify the event that will trigger execution of the block. In Altia Design, an event always has a name and a value and can originate from one of several sources:

## Execution of a stimulus definition

When a stimulus definition is executed, an event is generated based on the Execute Name and Value fields of the definition. If the name and value match the specification in a WHEN statement, the control block associated with the WHEN statement will execute. Both input stimulus and timer stimulus can generate events that trigger the execution of one or more control blocks. To learn more about defining stimulus, please refer to [Chapter 4: Stimulus Edotor](#).

## Execution of certain statements in a control block

SET, EXPR, and MATH control statements (described later in this chapter) generate events when they execute. These events typically change animation states for one or more objects. If the name and value for the event trigger the specification in another WHEN statement, the control block associated with the WHEN statement will execute after the current WHEN block completes its execution.

## Execution of an external program

Altia Design supports communication with C/C++, Java, and Visual Basic application programs. These programs can generate events by calling the **AtSendEvent()** or **AtSendText()** functions provided by Altia Design's application programming interface (API) library. Refer to the [Altia API Reference Manual](#) for more details.

In most cases, an event generated by any of the above sources has a name and value that match an animation state of some object(s) and/or match a specification for a WHEN statement. The event will cause a visual change in the object when it has a name and value that match the object's animation state. If the event has a name and value that match a WHEN statement specification, it will activate that WHEN control block.

Each time an event occurs from any of the above sources, the name and value associated with the event are remembered (stored in computer memory). The value in memory is updated for each occurrence of a new event with the same name. Control statements such as IF, AND, OR, SET, EXPR, and MATH can access the newest remembered value by simply referring to the name. If an event has never occurred for a particular name, the value defaults to 0 when the name is referenced.

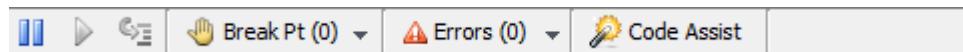
## 6.2 User Interface

The Control Editor consists of the following components:

- **Toolbar**
- **Statement Palette**
- **Code Editor**
- **Message Area (when a message is active)**
- **Syntax Helper (not visible when Code Assistant is active)**
- **Code Assistant**

These components are described in further detail in the following sections.

## 6.3 Control Editor Toolbar

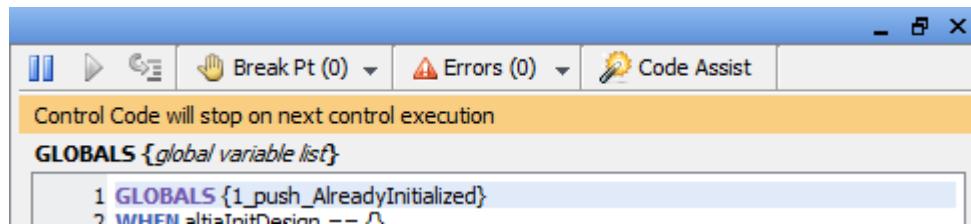


The Control Editor Toolbar has controls for debugging statement execution as well as for managing breakpoints and errors. In addition it provides access to the optional Code Assist interface.

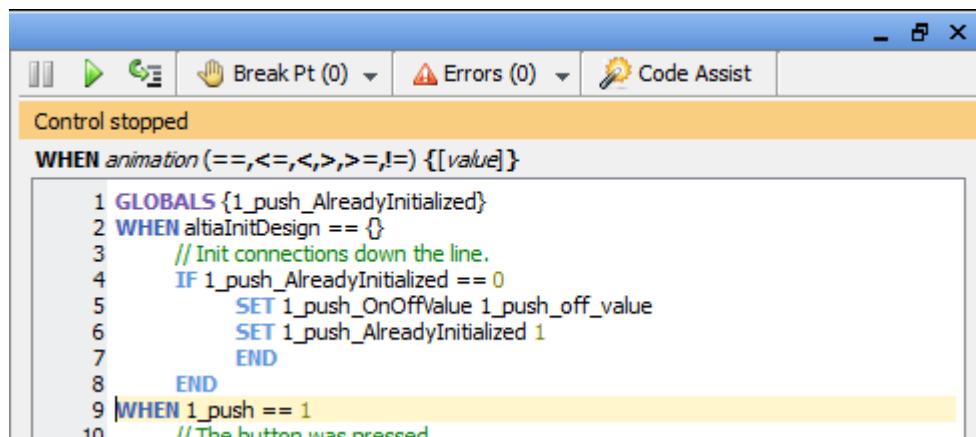
### 6.3.1 Debugging Controls

#### Pause

The first button for the Debugging Controls is the Pause button. Pressing this button will stop control code execution at the very next statement. Since control code executes in blocks determined by events, the code may not stop until such an event occurs. The following message will appear in the Message Area after the Pause button is pressed:



When the control code execution does stop, the Code Editor will highlight the stopped line and the Message Area will display a notification of the stop:



The screenshot shows a software interface with a toolbar at the top containing icons for Stop, Run, Break Pt (0), Errors (0), and Code Assist. Below the toolbar is a message bar that says "Control stopped". The main area is a code editor with the following pseudocode:

```
WHEN animation (==,<=,<,>,>=,!=) {[value]}

1 GLOBALS {1_push_AlreadyInitialized}
2 WHEN alitaInitDesign == {}
3 //Init connections down the line.
4 IF 1_push_AlreadyInitialized == 0
5   SET 1_push_OnOffValue 1_push_off_value
6   SET 1_push_AlreadyInitialized 1
7 END
8 END
9 WHEN 1_push == 1
  //The button was pressed
```

The line "9 WHEN 1\_push == 1" is highlighted in yellow, indicating it is the current statement being executed.

The Pause button will be disabled while the control code is not running.

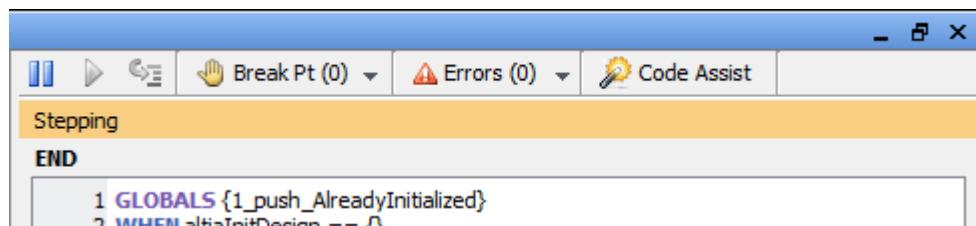
## ▶ Resume

The second button for the Debugging Controls is the Resume button. Pressing this button will resume control code execution from the statement where execution was previously stopped.

The Resume button will be disabled while the control code is running.

## ▶ Step

The third button for the Debugging Controls is the Step button. Pressing this button will step control code, executing the current statement and then stopping. If the current statement is at the end of a block (i.e. WHEN or ROUTINE) then code will not stop until an event is received which causes a new block to execute. The following message will appear in the Message Area when this situation occurs:



The screenshot shows a software interface with a toolbar at the top containing icons for Stop, Run, Break Pt (0), Errors (0), and Code Assist. Below the toolbar is a message bar that says "Stepping". The main area is a code editor with the following pseudocode:

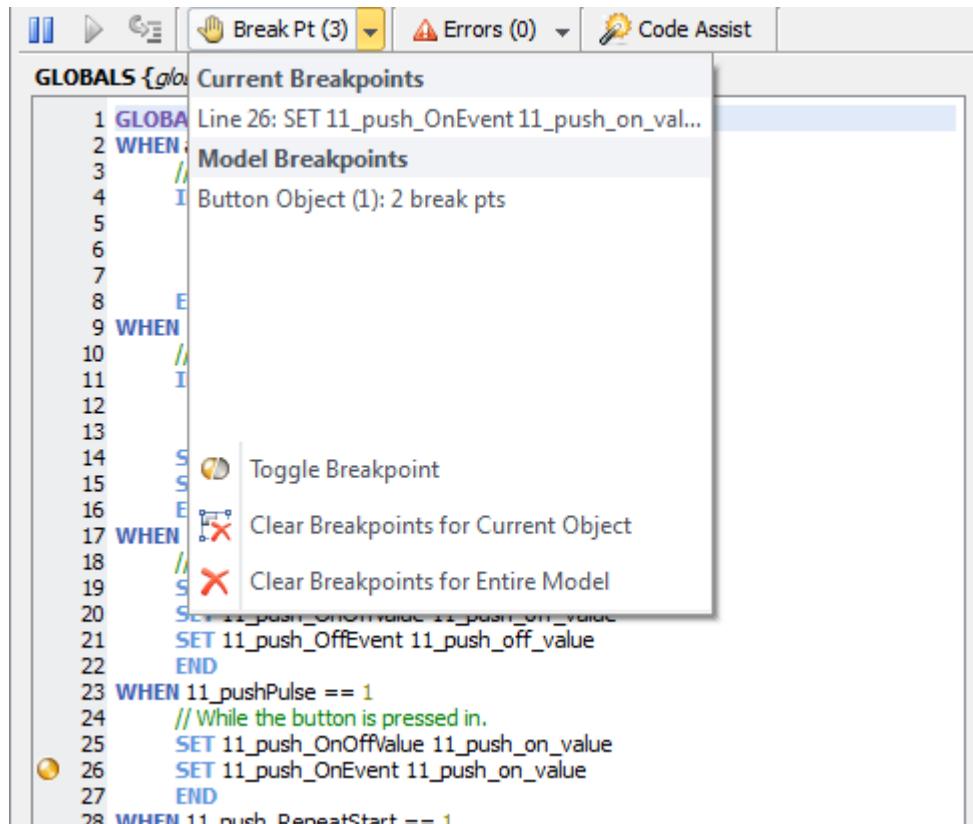
```
END

1 GLOBALS {1_push_AlreadyInitialized}
2 WHEN alitaInitDesign == {}
```

The Step button will be disabled while the control code is running.

## 6.3.2 Breakpoint Controls

The Breakpoint button will show the total number of breakpoints currently active in the Altia Design project. Pressing the Breakpoint button will toggle a breakpoint on the current line in the Code Editor. Pressing the drop-down arrow on the button will show the Breakpoint Menu:



### Current Breakpoints

The **Current Breakpoints** lists all the breakpoints for the currently selected object in Altia Design. Each breakpoint is shown using the line number and the statement text for that line. Clicking on a breakpoint in this list will move the text cursor in the Code Editor to the line for that breakpoint.

### Model Breakpoints

The **Model Breakpoints** lists all the objects in the Altia Design project which contain at least one breakpoint. The number of breakpoints is shown for each object. An object will only appear once in the list regardless of how many breakpoints are set for the object. Clicking on an object in this list changes the currently selected object in Altia Design to the object clicked. The text cursor for the Code Editor will change to a line with a breakpoint for that object.

## **Toggle Breakpoint**

Selecting **Toggle Breakpoint** from the Breakpoint Menu will toggle a breakpoint on the current line in the Code Editor.

**NOTE:** The Toggle Breakpoint item will be disabled if the code statement for the current line does not yet exist. This occurs when editing code in the Code Editor. Control code must be "applied" using the Apply Button before physical statements will be created.

## **Clear Breakpoints for Current Object**

Selecting **Clear Breakpoints for Current Object** from the Breakpoint Menu will remove all breakpoints for the currently selected object in Altia Design. These are the breakpoints currently shown in the Code Editor.

After clearing the breakpoints for the current object, those breakpoints will no longer appear in the Breakpoint Menu.

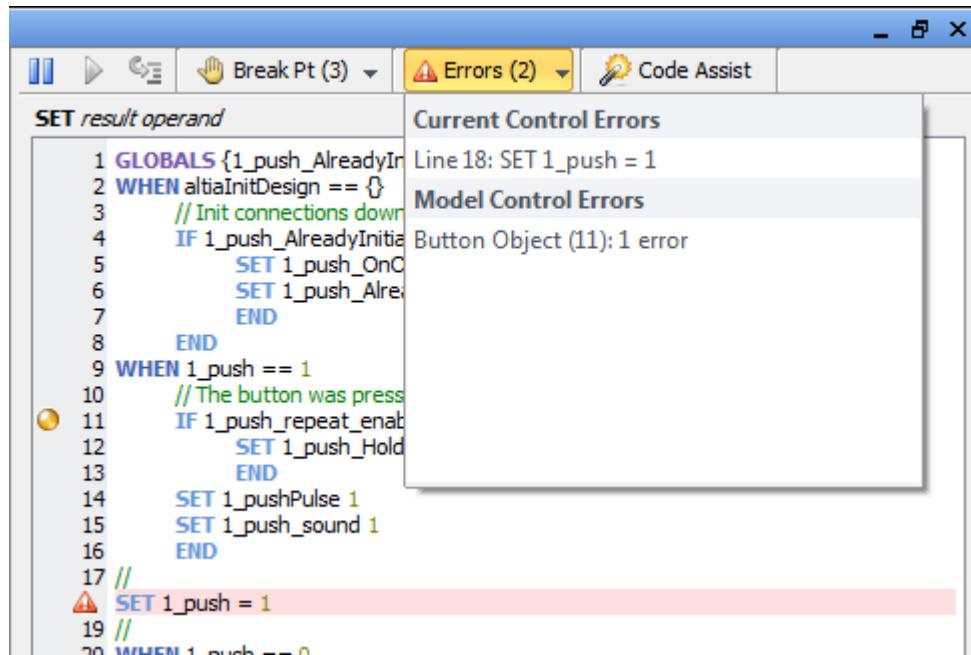
## **Clear Breakpoints for Entire Model**

Selecting **Clear Breakpoints for Entire Model** from the Breakpoint Menu will remove all breakpoints in the entire Altia Design project.

After clearing the breakpoints, no breakpoints will appear in the Breakpoint Menu.

### 6.3.3 Error Controls

The Error button will show the total number of errors currently detected in the Altia Design project. Pressing the Error button will show the Error Menu:



**NOTE:** The only way to clear an error is to fix the control code for that line. Mousing over the Error icon in the Code Editor will show a tooltip with details on the error.

### Current Control Errors

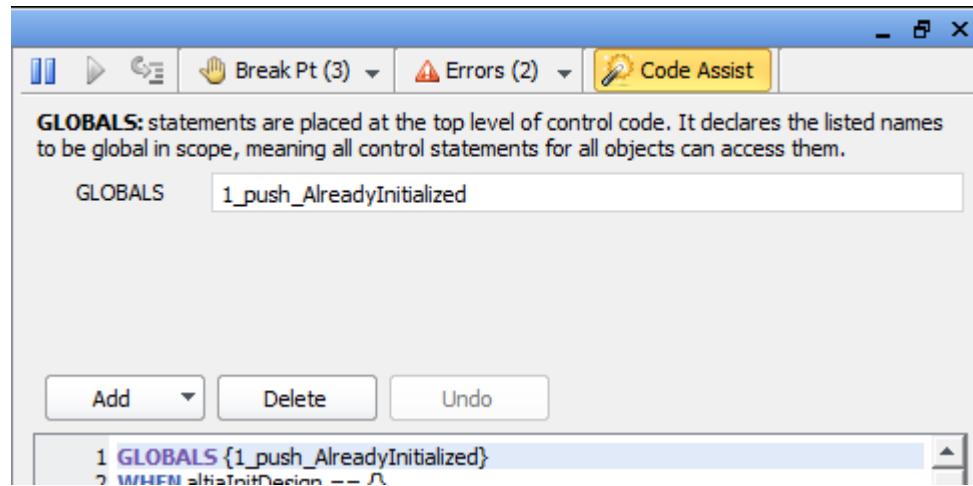
The **Current Control Errors** lists all the errors for the currently selected object in Altia Design. Each error is shown using the line number and the statement text for that line. Clicking on an error in this list will move the text cursor in the Code Editor to the line for that error.

### Model Control Errors

The **Model Control Errors** lists all the objects in the Altia Design project which contain at least one error. The number of errors is shown for each object. An object will only appear once in the list regardless of how many errors are present for the object. Clicking on an object in this list changes the currently selected object in Altia Design to the object clicked. The text cursor for the Code Editor will change to a line with an error for that object.

### 6.3.4 Code Assist

The Code Assist button will toggle the Code Assistant interface. When active the Code Assist button will become highlighted and the Code Assistant interface will be visible:



**NOTE:** The Code Assistant will hide itself when no object is select in the Altia Design project. The Code Assist button will remain highlighted in this situation to indicate that the mode is still active. The Code Assistant will reappear as soon as a single object is selected in the Altia Design project.

## 6.4 Statement Palette

COMMENT	Alt-K
WHEN	Alt-W
ROUTINE	Alt-R
GLOBALS	Alt-G
IF	Alt-I
ELSEIF	Alt-N
ELSE	Alt-L
AND	Alt-A
OR	Alt-O
EXPR	Alt-P
SET	Alt-S
MATH	Alt-M
LOOP	Alt-U
CALL	Alt-D
RETURN	Alt-T
END	Alt-E
OPEN	
WRITE	
READ	
CLOSE	
OPEN_VIEW	
CLOSE_VIEW	
OPEN DESIGN	
CLOSE DESIGN	
EXTERN	

The Statement Palette shows a list of the possible control code statements used by Altia Design. The statement types are described in [Section 6.6, Control Code Statements](#).

The list will dynamically update to show only the allowed statements for the current line in the Code Editor. If a statement is disabled (light gray text), then the statement is not allowed for the current line.

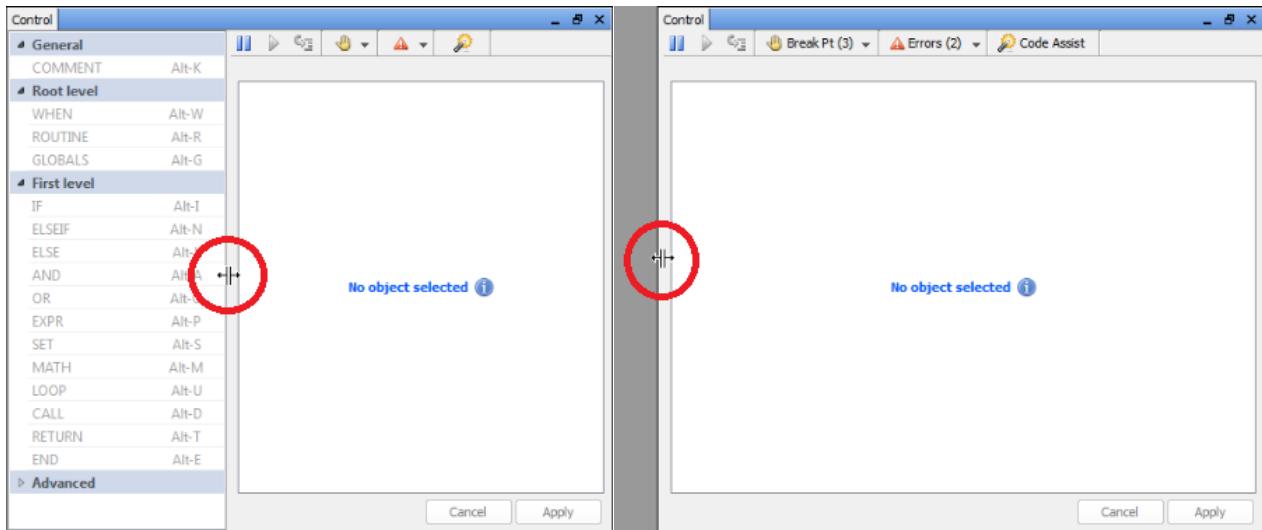
Clicking an enabled statement in the palette will automatically insert it into the Code Editor. If the cursor in the editor is on a blank line, the statement will be inserted on that line. If the cursor in the editor is on a line with text, a new line will be appended and the statement will be put onto the new line.

The commonly used statements have a shortcut key-sequence (i.e. ALT-W for WHEN). The shortcut can be pressed on the keyboard while using the Code Editor. The result is exactly the same as clicking an enabled statement in the editor.

**NOTE:** The shortcuts will not function if the statement is currently disabled in the palette.

### 6.4.1 Resizing the Statement Palette

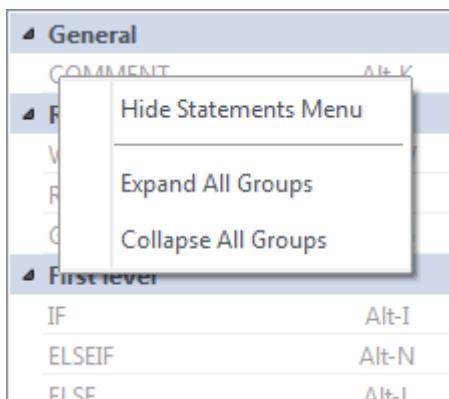
The Statement Palette can be resized by left-click-and-dragging the empty region just to the right of the palette. There is a minimum width for the palette, but the palette can be hidden by dragging the size to the left (nearly to the edge of the Control Editor window itself):



If hidden, the palette can be shown again by left-click-and-dragging the empty region on the left edge of the Control Editor window. The palette will reappear after dragging a width that's large enough to show the palette.

## 6.4.2 Statement Palette Context Menu

Right-clicking in the Statement Palette will show the context menu:



The **Hide Statements Menu** item will hide the Statement Palette (similar to dragging its size to zero as described in the previous section).

The **Expand All Groups** item will expand every group heading in the palette, showing all possible statements.

The **Collapse All Groups** item will collapse every group heading in the palette, hiding all statements.

## 6.5 Code Editor

The Code Editor is a text based editor used for creating and modifying control code statements for an object in Altia Design:

```
GLOBALS {global variable list}
1 GLOBALS {1_push_AlreadyInitialized}
2 WHEN altiaInitDesign == {}
3 // Init connections down the line.
4 IF 1_push_AlreadyInitialized == 0
5 SET 1_push_OnOffValue 1_push_off_val
6 SET 1_push_AlreadyInitialized 1
7 END
8 END
9 WHEN 1_push == 1
10 // The button was pressed.
11 IF 1_push_repeat_enable == 1
12 SET 1_push_HoldStart 1
13 END
14 SET 1_pushPulse 1
15 SET 1_push_sound 1
16 END
17 WHEN 1_push == 0
18 // The button released.
19 SET 1_push_HoldStart 0
20 SET 1_push_OnOffValue 1_push_off_value
21 SET 1_push_OffEvent 1_push_off_value
```

No object selected ⓘ

### Syntax Helper

The top of the Code Editor displays a syntax hint based upon the statement for the current line. This helper is not visible when using the Code Assistant. If the statement for the current line is not recognized, and "Invalid" message will appear.

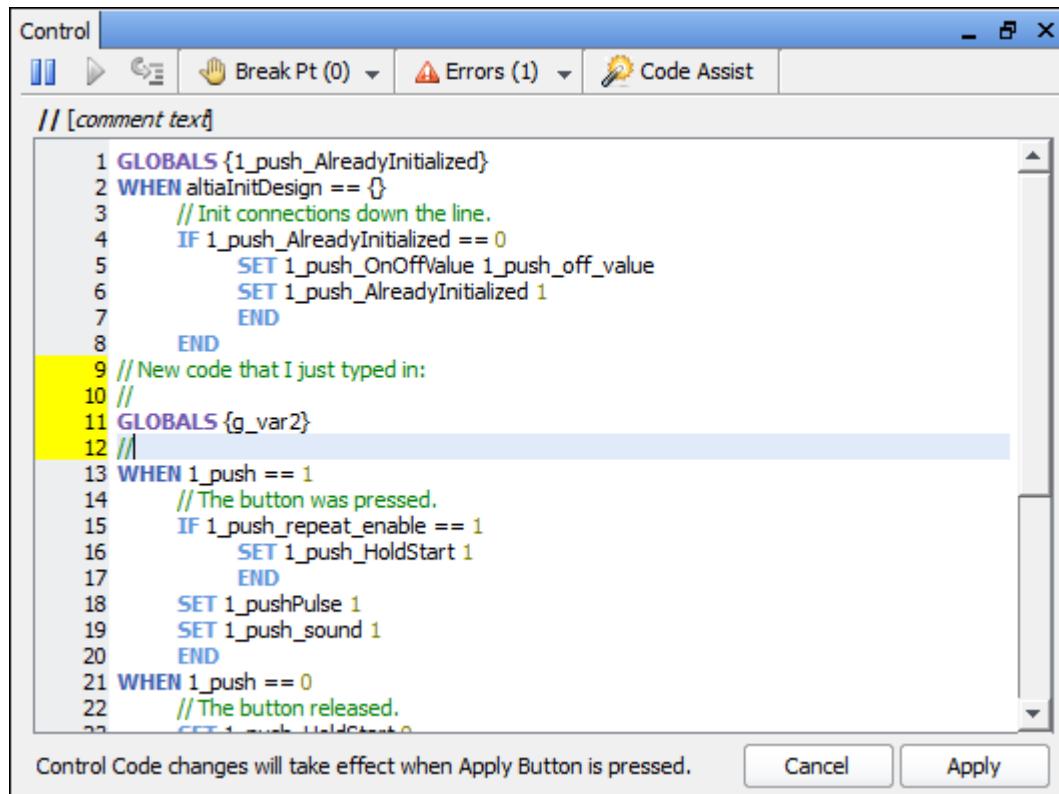
### Line Number Region

The left side of the Code Editor contains a line number region. This region can be clicked to set a breakpoint for a specific line. If a line has an error, an error symbol will appear in place of the line number.

**NOTE:** Clicking in the Line Number Region will not toggle breakpoints for modified lines. Changes must be "applied" using the Apply Button before breakpoints can be created.

## Text Editor

The center of the Code Editor is used for creating and editing control code statements. The text in the editor is colorized based upon syntax with statements in blue, numbers in gold, text strings in red, and plain text in black. The current line in the text editor will be highlighted in light blue:



The screenshot shows the Altia Control Code Editor window titled "Control". The menu bar includes "File", "Edit", "Break Pt (0)", "Errors (1)", and "Code Assist". The main area contains the following control code:

```
// [comment text]
1 GLOBALS {1_push_AlreadyInitialized}
2 WHEN altiaInitDesign == 0
3   // Init connections down the line.
4   IF 1_push_AlreadyInitialized == 0
5     SET 1_push_OnOffValue 1_push_off_value
6     SET 1_push_AlreadyInitialized 1
7   END
8 END
9 // New code that I just typed in:
10 //
11 GLOBALS {g_var2}
12 //
13 WHEN 1_push == 1
14   // The button was pressed.
15   IF 1_push_repeat_enable == 1
16     SET 1_push_HoldStart 1
17   END
18   SET 1_pushPulse 1
19   SET 1_push_sound 1
20 END
21 WHEN 1_push == 0
22   // The button released.
23   SET 1_push_HoldStart 0
```

Below the code editor, a message states: "Control Code changes will take effect when Apply Button is pressed." There are "Cancel" and "Apply" buttons at the bottom.

## Apply and Cancel Buttons

When editing control code statements, the **Cancel Button** and **Apply Button** will become enabled. During this time the control code is in a transient state. Edits will not take effect until the changes are applied by clicking the **Apply Button**. Changes can be cancelled by clicking the **Cancel Button**.

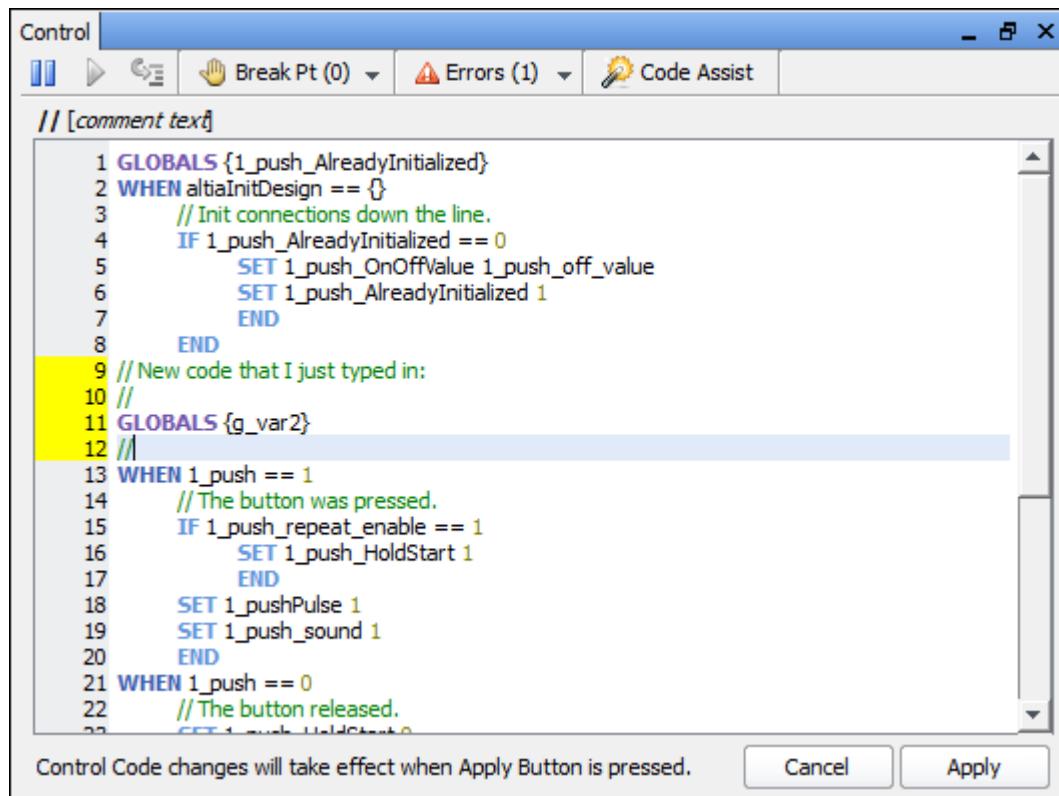
**NOTE:** Code changes will automatically be applied when changing the selection in Altia Design.

## Invalid Selection in Altia Design

Control code can be edited only when a single object is selected in Altia Design. If no objects are selected, or if multiple objects are selected, the Code Editor will hide itself and show the appropriate status message.

## 6.5.1 Editing Code (Code Assistant OFF)

When editing code (with Code Assistant OFF), modified lines will have a yellow indication in the **Line Number Region**. In addition the **Apply Button** and the **Cancel Button** will become enabled:



## Auto Formatting

Control code is automatically formatted by the editor after the **Apply Button** is pressed. This includes capitalizing the statement name and creating the indentation structure. It is not necessary to put in the desired indentation while editing.

Pressing ENTER while editing code will proceed to a new line. The editor will automatically insert tabs in the new line to match the preceding line. This indentation is temporary because all the code will be automatically formatted after the **Apply Button** is pressed.

**NOTE:** Empty or blank lines will automatically be converted to comments. Every line must contain a valid control code statement in Altia Design.

## Tooltips

Mousing over any line will display a tooltip showing syntax help for the statement on that line:

```
9 WHEN 16_push == 1
10 // The button was pressed.
11 IF 16_push_repeat_enable == 1
12     SET 16_push_HoldStart 1
13 END
14 SET 16_push_
15 SET 16_push_sound 1
16 FND
```

## Undo and Redo

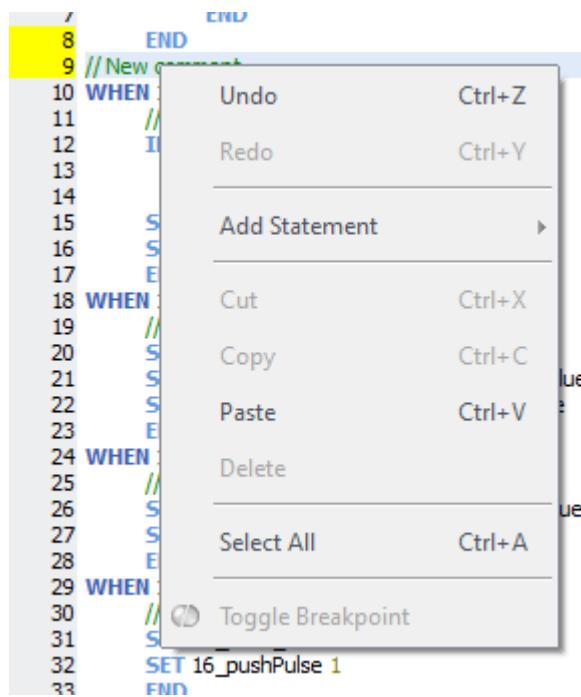
The shortcut key-sequence of CTRL-Z will undo the last change. This shortcut can be used repeatedly to undo more than one previous operation. The shortcut will have no result if there are no previous operations to undo.

The shortcut key-sequence of CTRL-Y will redo the last undone change. This shortcut can be used repeatedly to redo multiple undone operations. The shortcut will have no result if all previously undone operations have been redone.

**NOTE:** The Undo and Redo stacks are cleared after code changes are applied using the Apply Button or after changing the current selection in Altia Design.

## Context Menu

Right-clicking on a line in the Code Editor will move the cursor to that line and show the context menu for that line (appearance with Code Assistant OFF):



### Undo and Redo

Same has shortcut key-sequence. Enabled if operation is valid.

### Add Statement

Shows the menu of the Statement Palette (see [Section 6.4, Statement Palette](#)).

### Cut, Copy, Paste, Delete, Select All

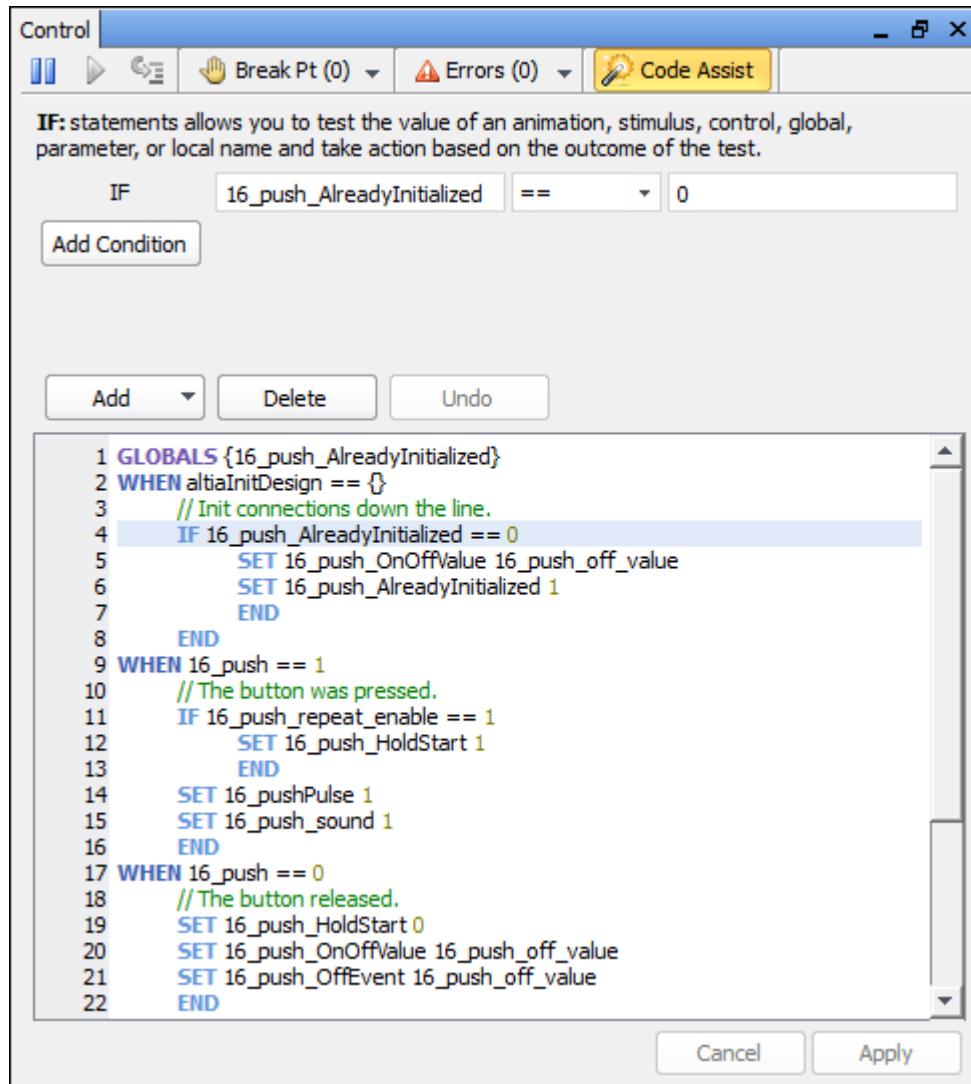
Standard text editor commands.

### Toggle Breakpoint

Same as in the Breakpoint Toolbar command (see [Section 6.3.2, Breakpoint Controls](#)).

## 6.5.2 Editing Code (Code Assistant ON)

When editing code (with Code Assistant ON), text entry in the Code Editor is locked. Adding, deleting, and changing control code statements is performed using only the Code Assistant:



### Adding and Deleting Control Code Statements

The **Add Button** will show a popup menu of the Statement Palette (see Section 10.4). Clicking a statement in the palette will add the statement in the Code Editor in the same fashion as using the Statement Palette. The current line will change to the new statement, and the Code Assistant will show text entry fields for the new statement.

The **Delete Button** will delete the current line. This operation can be undone with the **Undo Button**.

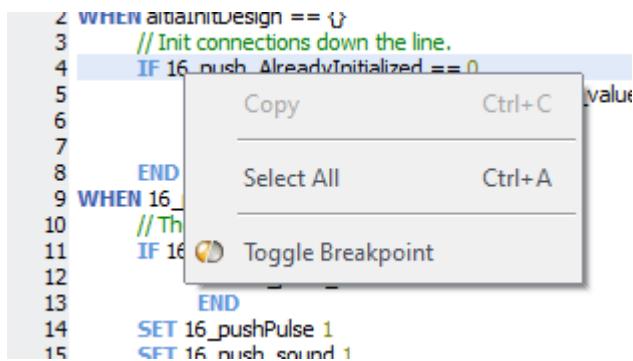
## Undo and Redo

The **Undo Button** will undo the last change. This button can be clicked repeatedly to undo more than one previous operation. The button will be disabled if there are no previous operations to undo.

The **Redo Button** will redo the last undone change. This button can be clicked repeatedly to redo multiple undone operations. The button will be disabled if all previously undone operations have been redone.

## Context Menu

Right-clicking on a line in the Code Editor will move the cursor to that line and show the context menu for that line (appearance with Code Assistant ON):



A screenshot of a Code Editor window showing a context menu. The menu is open over line 4 of the code, which contains the text "IF 16\_alreadyInitialized == 0". The menu items are: "Copy" (Ctrl+C), "Select All" (Ctrl+A), and "Toggle Breakpoint". The "Select All" item is highlighted with a blue selection bar. The background shows lines of code from 2 to 15, including "WHEN 16", "END", and "SET 16\_pushPulse 1".

```
2 WHEN aMainInDesign == 0
3 // Init connections down the line.
4 IF 16_alreadyInitialized == 0
5
6
7
8 END
9 WHEN 16
10 // Th
11 IF 16
12
13 END
14 SET 16_pushPulse 1
15 SET 16_push_sound 1
```

### Copy, Select All

Standard text editor commands.

Cut and Paste are not available because the text entry is locked when the Code Assistant is ON.

### Toggle Breakpoint

Same as in the Breakpoint Toolbar command (see [Section 6.3.2, Breakpoint Controls](#)).

## 6.6 Control Code Statements

This section details the different control code statements used in Altia Design.

### 6.6.1 Code Structure

To add control to an object in Altia Design it must be the only object selected. Use the Control Editor to create control code statements for the object (see [Section 6.5, Code Editor](#)).

The first time you create control code for an object, **WHEN**, **ROUTINE**, **Globals**, and **COMMENT** will be the only statements available in the Statement Palette. All control statement blocks must begin with a **WHEN** or **ROUTINE** statement. The **Globals** statement allows you to define global variables accessible to all control blocks and the **COMMENT** statement allows you to type in your own comments.

To add additional statements, use the Statement Palette again. The palette list will always reflect which statements can be added after the currently selected statement.

### Parent Statements

The following statements can have children: **WHEN**, **ROUTINE**, **IF**, **ELSEIF**, **ELSE**, and **LOOP**. The children will be executed if the statement evaluates to "true" when executed. Example:

```
WHEN 1_push == 1
  // The button was pressed.
  IF 1_push_repeat_enable == 1
    SET 1_push_HoldStart 1
    END
  SET 1_pushPulse 1
  SET 1_push_sound 1
  END
```

**NOTE:** Parent statements must be terminated with an END statement after the last child for the parent.

## Condition Statements

The following statements can have conditions: **IF**, **ELSEIF**, **LOOP**. The conditions appear on the same line, appended to the end. Multiple conditions can be specified for one of these statements. Example:

```
IF 65_knob_NewAngle < 65_knob_MinAngle OR 65_knob_NewAngle > 65_knob_MaxAngle
IF 65_knob_LastAngle == 65_knob_MinAngle OR 65_knob_LastAngle == 65_knob_MaxAngle
    SET 65_knob_NewAngle 65_knob_LastAngle
    END
ELSEIF 65_knob_NewAngle < 65_knob_MinAngle
    SET 65_knob_NewAngle 65_knob_MinAngle
    END
ELSE
    SET 65_knob_NewAngle 65_knob_MaxAngle
    END
END
```

The **AND** and **OR** statements will only become enabled in the Statement Palette when editing a line for a statement that allows conditions (see above).

## Syntax Help

Each statement has a help syntax which can be viewed in the Control Editor by mousing over a statement in the text editor. In addition, the help syntax will be shown in the Syntax Helper at the top of the Code Editor (when Code Assistance is OFF).

An example of help syntax for the **EXTERN** statement:

**EXTERN** *name {parameter list} library (STRING/INT/NONE) result*

Rules:

- Text in **BOLD CAPS** should be entered exactly as written.
- Text in *italics* should be replaced with the appropriate data like a number, text string, animation name, global name, etc. Unless specified as optional (see below), something must appear where the italicized text is. Remember an empty parameter can be shown using a set of empty curly braces: {}
- Curly brackets {} are explicit and should be used around groups of items. When in doubt, use curly brackets around the different parameters in the statement syntax.
- Parenthesis () show a set of mandatory choices where one \*must\* be picked. In the example above, one (and only one) of either STRING, INT, or NONE must be used. The parenthesis should NOT appear in the final code statement.
- Square brackets [] show a set of optional choices where one \*may\* be picked, or none at all. The brackets should NOT appear in the final code statement.

Completed Examples (note that something exists for each and every parameter in an EXTERN statement):

```
EXTERN "ShowDataFunc" {anim1 anim2} "myDLL.dll" NONE {}
EXTERN funcanim {"foo"} dllanim INT resultanim
EXTERN @refanim {12 "bar" 34.56} "myDLL.dll" STRING resultanim
```

## Code Assistance

Each statement has a Code Assistant which is shown in the Control Editor when the Code Assist feature is ON.

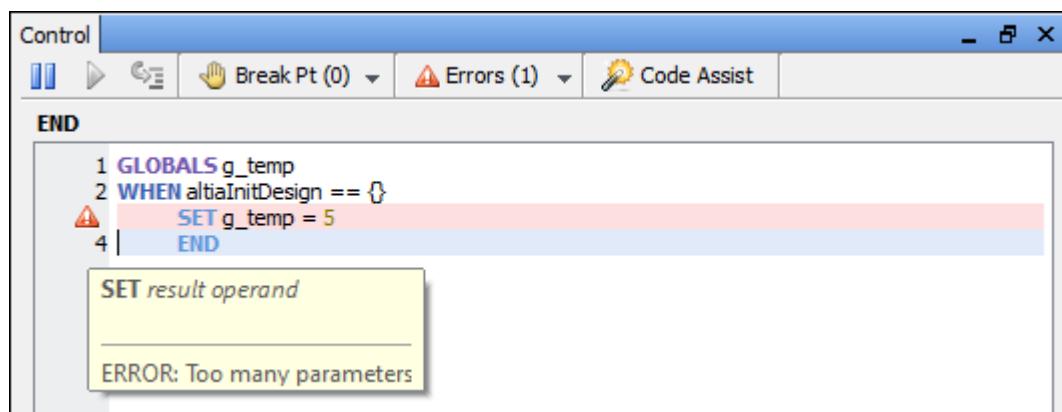
An example of the Code Assistant for the **EXTERN** statement:



The fields in the Code Assistant will be described in the details for the specific statement.

## Errors

The Control Editor has a simple error checking algorithm. When an error is detected, an error symbol will appear in the Line Number Area for the erroneous line. Mousing over the symbol will show a tooltip describing the error condition:



Currently only the following errors are detected:

- Too many parameters for a statement
- Too few parameters for a statement
- Invalid parameters for statements that use comparisons (i.e. ==, <, >, etc.)

**NOTE:** Code structure errors are not currently detected. This includes missing END statements, extra END statements, missing Root Level statements, etc. In addition, invalid animation names will not be trapped.

## 6.6.2 General Statements

### COMMENT (Alt + K)

Syntax Help: `// [comment text]`

Code Assistant: `COMMENT`

COMMENT statements can be placed anywhere in the control code. They are used to insert helpful information and do not affect control execution. Comments generally document what is being done and why it was done the way it was.

## 6.6.3 Root Level Statements

### WHEN (Alt + W)

Syntax Help: `WHEN animation (==,<=,<,>,>=,!=) {[value]}`

Code Assistant: `WHEN`

At the top-most level, you can specify what conditions to look for with one or more WHEN statements. Each WHEN statement consists of an animation/stimulus name, a value, and a relationship between the two. The available relationships are == (equals), < (is less than), <= (is less than or equal to), > (is greater than), >= (is greater than or equal to), and != (does not equal). The value in the second field may be numeric or another animation/stimulus name.

Once the WHEN is defined, Altia Design will watch for an event that satisfies this relationship. When the relationship is true, the control statements under that WHEN will be executed.

Completing a WHEN Statement:

1. Type into the field next to WHEN the name for the event that should execute the control block.
2. Click on the drop-down combo field labeled "==" to access the allowed conditional operators choose the desired comparison (==, <, <=, >, >=, !=).
3. Type into the field next to the conditional operator a value. The conditional operator chosen in step 2 will be used to compare the value of the event with the value in this field. If a name is entered, the value for the name at the time of execution is used. If the name's value has never been set, 0 will be used. If this field is left empty, it implies that the WHEN should execute for all possible event values of the name given in the first field.

The reserved animation name **altiaInitDesign** is a special, pre-defined animation name. Each time an Altia design is loaded, an altiaInitDesign event is generated with a value equal to 0 in Edit mode. In run mode the event is generated with the "design number" (see the OPEN\_DESIGN statement later on in this chapter for more information about design numbers). Creating a WHEN statement to catch this event provides an opportunity to do any initialization the design requires. For instance, to create a WHEN block that always executes when the design is opened in the editor, use:

```
WHEN altiaInitDesign == {}
```

The **value** field can one of several types:

- Any value represented by empty curly brackets { }
- An integer value, like 5
- A floating point value, like 1.234
- A string value in double quotes, like "Hello World"
- Another animation/stimulus/control name with one of the above types
- Another Text I/O object's text animation name preceded by @ (i.e., @input\_text), meaning use that name's value as the name whose value is to be substituted

## ROUTINE (Alt + R)

Syntax Help:	<b>ROUTINE</b> <i>name</i> { <i>parameter list</i> } { <i>local variable list</i> }									
Code Assistant:	<table><tr><td><b>ROUTINE</b></td><td>Name:</td><td><input type="text"/></td></tr><tr><td></td><td>Parameters:</td><td><input type="text"/></td></tr><tr><td></td><td>Locals:</td><td><input type="text"/></td></tr></table>	<b>ROUTINE</b>	Name:	<input type="text"/>		Parameters:	<input type="text"/>		Locals:	<input type="text"/>
<b>ROUTINE</b>	Name:	<input type="text"/>								
	Parameters:	<input type="text"/>								
	Locals:	<input type="text"/>								

A ROUTINE control statement is created at the top level. A ROUTINE is a named sequence of control statements for accomplishing a sub-task. Creating a ROUTINE encapsulates the subtask so it can be invoked (or CALLED) without duplicating the sequence. ROUTINE statements are listed at the top level of control along with WHEN statements, but they only execute when explicitly invoked using a CALL statement.

To create a ROUTINE, you fill in the routine name and, optionally, any routine parameters and locals. Parameters are names or values that represent values given to the ROUTINE when it is called. Locals are names needed to hold temporary values inside the routine when it executes. All control statements necessary to accomplish the sub-task are then added underneath the ROUTINE statement.

A ROUTINE **Name** can be anything as long as it does not conflict with the name given to an existing ROUTINE block. As with other names in Altia Design, ROUTINE names cannot contain space (' ') characters.

ROUTINE **Parameters** are one or more names or values separated by space characters. The number of items in the Parameters line must match exactly with the number of **Parameters** items in each CALL statement that references the same ROUTINE name.

ROUTINE **Locals** have the same capabilities and usage rules as described for GLOBALS except that locals only persist during the execution of the ROUTINE block in which they are declared. As soon as a ROUTINE block completes, the locals and their values are destroyed. Locals are declared as one or more names separated by space characters.

## GLOBALS (Alt + G)

Syntax Help:

**GLOBALS {global variable list}**

Code Assistant:

**GLOBALS**

The GLOBALS statement is placed at the top level of control code. It declares the listed names to be global in scope, meaning *all* control statements for *all* objects can access them. Multiple names are declared in a single GLOBALS statement by separating the names with space (' ') characters. Globals are storage elements for holding arithmetic results or strings required by WHEN and/or ROUTINE blocks.

**NOTE:** The names appearing in a GLOBALS statement are only recognized within control and are not visible outside of control (i.e., they are not visible for animation, stimulus definitions, or external client applications).

**NOTE:** All of the usage and example information that follows is applicable to ROUTINE block locals as well as globals. The few differences between the two storage types are duly noted.

Control globals are not given a formal type when they are declared in a GLOBALS statement. Instead, a global dynamically takes on a type based on the type of the value assigned to it from a SET, MATH, CALL, or EXPR statement. A global can be assigned one of the following:

- An integer value such as 5 or a value from another name that refers to an integer value (i.e., an animation, stimulus, global, routine local, or routine parameter name).
- A floating point value such as 1.234 or a value from another global, routine local, routine parameter, animation name, or stimulus name that refers to a floating point value.
- A string value such as "this is a string" or the string from a global name, routine local name, routine parameter name, or a text I/O object's text animation name.

A global name can also refer to an array of items by using the name with subscripts ([]]) during assignment operation as illustrated in the following examples:

```
GLOBAL value
...
SET value[0] 3.14
SET value[name] "foo"
SET value[@name] 0
SET value["last"] 0
```

As shown in this example, the subscript identifier can be: an absolute number, a name (in which case the value of the name is used as the subscript), an indirect reference through a name (in which case the value of the name is used as a name and that name's value is the subscript identifier), or even a string.

The first assignment to a global determines whether it is a single item or an array of items. After the first assignment, the global name must be used consistently throughout control or a syntax error will result (for runtime, illegal assignments are simply ignored).

For example, the following pair of statements would create a syntax error during execution:

```
SET value 5  
SET value[0] 6
```

When an assignment is performed on a global, its name and value are not routed to graphical objects and external clients as is the case with animation and stimulus names. For this reason, an assignment to a global cannot change any object's animation states, cannot start or stop stimulus timers, and cannot be received by a client application. The intent of a global is to provide integer, float, or string storage outside of the animation and stimulus name space. When a global is assigned to an existing animation or stimulus name or to a name that has never been declared as either a global or local, then a route will occur for the name that is taking on the value of the global.

A global is typically used to hold a temporary arithmetic result during a WHEN block execution and/or between WHEN and ROUTINE block executions until it is appropriate to assign the result to an animation name. This is in contrast to a local which is similar in capability and usage to a global but only persists during execution of the ROUTINE block in which the local is declared.

Finally, conflicts between parameter/local, global, and animation/stimulus names can occur and are resolved by scope - a parameter/local name has priority over a global with the same name and a global name has priority over an identical animation or stimulus name. If a reference is made to a name in control and it does not refer to a parameter or local, and no global is found with the same name, then the name is automatically assumed to be an animation or stimulus name.

As is the case for animation, stimulus, parameter, and local names, global names are character case sensitive (i.e., you must use character case consistently to ensure proper references).

## 6.6.4 First Level Statements

After adding a root statement such as WHEN or ROUTINE, the first level statements become available in the Statement Palette. The first level statements are:

- IF
- SET
- MATH
- EXPR
- END
- LOOP
- CALL

### IF (Alt + I)

Syntax Help:	IF <i>animation</i> (==,<=,<,>,>=,!=) <i>value</i> [OR,AND ...]		
Code Assistant:	IF <input type="text"/> == <input type="button" value="▼"/>		

An IF statement allows you to test the value of an animation/stimulus/control/global/parameter/local name and take action based on the outcome of the test. The IF permits conditional execution of control based on whether the relationship between the name and the value is true. The IF statement appears indented below the WHEN statement because the IF statement is subject to the WHEN statement.

Fill in the animation/stimulus/control/global/parameter/local name and the value to test against. Select the relationship by pressing the left mouse button down on the button labeled with a relational operator ( $==$ ,  $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $\neq$ ) and then selecting the appropriate one from the menu. If the name typed into the first field is new and, therefore, has no current value, its value will be 0 for the purposes of the IF statement execution.

## ELSEIF (Alt + N)

Syntax Help: **ELSEIF** *animation* ( $==$ ,  $\leq$ ,  $<$ ,  $>$ ,  $\geq$ ,  $\neq$ ) *value* [OR,AND ...]

Code Assistant: **ELSEIF**   $==$

The ELSEIF statement provides an easier way of specifying an ELSE followed by an IF. If the preceding IF condition is false, the test of the ELSEIF condition will take place. If true, the control statements under the ELSEIF will be executed. The ELSEIF statement is only available immediately after an END statement that completes an earlier IF or ELSEIF block. Example:

```
IF color == 1
  ...
  END
ELSEIF color == 2
  ...
  END
ELSE
  ...
END
```

## ELSE (Alt + L)

Syntax Help: **ELSE**

Code Assistant: **ELSE** (no parameters for this statement)

When END completes an IF sub-block, ELSE appears in the Statement Palette. An ELSE statement handles the alternative case where the IF statement above it fails. An ELSE sub-block gives your design the opportunity to execute a different sub-block should the IF condition not be met. Based on the test condition, either the control statements under the IF or those under the ELSE will be executed. The ELSE has no fields to populate since the matching IF defines the test.

## AND (Alt + A), OR (Alt + O)

Syntax Help: **(AND/OR)** *animation* ( $==$ ,  $\leq$ ,  $<$ ,  $>$ ,  $\geq$ ,  $\neq$ ) *value*

Code Assistant:

Add Condition	AND	==	X
	OR	==	X

You can immediately add one or more AND or OR statements immediately after an IF, ELSEIF, or LOOP statement. The conditions are placed on the same line as the parent statement. The AND/OR statements permit the creation of more complicated IF conditions. If AND/OR conditions are present but not satisfied, the control sub-block will not execute.

**NOTE:** Evaluation of the conditions occurs from left to right starting with the parent statement. All AND evaluations must be true for the sub-block to execute. Only one OR evaluation must be true.

When using the Code Assistant, the conditions are presented in a tabular list. An IF, ELSEIF, and LOOP statement can have an arbitrary number of conditions. To add a new condition to one of these statements, press the **Add Condition** button which will append a new, empty condition to the list of current conditions. To remove a condition, press the red "X" for the row to remove.

Conditions follow the same rules as IF statements when setting the values to compare.

## EXPR (Alt + P)

Syntax Help:

**EXPR** animation {expression statement}

Code Assistant:

EXPR	=	
------	---	--

The EXPR statement is used to compute more complicated expressions than those possible with a SET or MATH statement.

For mathematical operations, the calculations are performed in floating point if at least one variable or constant value in the calculation has decimal digits. To force floating point calculations, add .0 to at least one constant in the right-hand (bottom) operand. If there are no constants, put \*1.0 at the end of the right-hand operand.

The EXPR statement allows setting an animation, stimulus, global, routine local, or routine parameter name to the result of almost any mathematical or logical expression. Legal operands in the expression field are:

- an integer or floating point value
- an animation/stimulus name (whose integer/floating point value will be used)
- a global, routine local, or routine parameter name
- a string enclosed in double-quotes (e.g., "foo") if a string is appropriate for the operation
- a library function, specified by placing a \$ in front of the name (e.g., \$sin)
- an interpreter command enclosed in brackets [] (e.g., \$format which is described later)

Legal operators for the expression field are as follows:

Legal operators grouped in decreasing order of precedence	
-;~,!	unary minus, bit-wise NOT, logical NOT. Valid for all numeric operands except for bit-wise NOT which only applies to integers.
* ,/,%	multiply, divide, remainder. Valid for all numeric operands except for remainder which only applies to integers.
+,-	add, subtract. Valid for all numeric operands.
<<, >>	left shift, right shift. Valid for integer operands only.
<,<=,>,>=	boolean less, less than or equal, boolean greater, greater than or equal. Each produces a 1 if true, 0 if false. Valid for all numeric and string operands (in which case string comparison is used).
==,! =	boolean equal, not equal. Each produces 1 if true, 0 if false. Valid for all numeric and string operands.
&	bit-wise AND. Valid for integer operands only.
^	bit-wise exclusive OR. Valid for integer operands only.
	bit-wise OR. Valid for integer operands only.
&&	logical AND. Produces 1 if both operands are non-zero, 0 if not. Valid for all numeric operands (i.e., not strings).
	logical OR. Produces 1 if either operand is non-zero, 0 if not. Valid for all numeric operands (i.e., not strings).

Evaluation is left to right and "lazy" meaning operands are only evaluated if needed to determine the final result.

**NOTE:** Precedence can also be controlled by grouping sub-expressions with parentheses, as in  $5+(3/4)$  vs.  $(5+3)/4$ .

Numerous library functions are available for use in the expression field of the EXPR statement. Most take integer or floating point values as parameters and return floats unless otherwise noted. The available library functions are as follows (in alphabetical order):

Available Library Functions	
Name of Function	Description
<b>\$abs(x)</b>	absolute value of x
<b>\$acos(x)</b>	arccosine of x
<b>\$asin(x)</b>	arcsine of x
<b>\$atan(x)</b>	arctangent of x
<b>\$atan2(y,x)</b>	arctangent of y/x
<b>\$ceil(x)</b>	smallest integer not less than x
<b>\$cos(x)</b>	cosine of x
<b>\$cosh(x)</b>	hyperbolic cosine of x
<b>\$double(x)</b>	convert integer x to floating point value
<b>\$exp(x)</b>	e raised to the x power
<b>\$floor(x)</b>	largest integer not greater than x
<b>\$fmod(x,y)</b>	floating point remainder of x divided by y
<b>\$hypot(x,y)</b>	square root of $((x*x)+(y*y))$
<b>\$int(x)</b>	floating point value x truncated (returns integer)
<b>\$log(x)</b>	natural logarithm of x
<b>\$log10(x)</b>	logarithm base 10 of x
<b>\$pow(x,y)</b>	x raised to the y power
<b>\$round(x)</b>	floating point value x rounded (returns integer)

<b>\$sin(x)</b>	sine of x
<b>\$sinh(x)</b>	hyperbolic sine of x
<b>\$sqrt(x)</b>	square root of x
<b>\$tan(x)</b>	tangent of x
<b>\$tanh(x)</b>	hyperbolic tangent of x

Two interpreter commands which are accessed similarly to library functions are worthy of special note:

- **\$format** - provides output capability similar to the C language library function **sprintf()**.
- **\$scan** - provides input capability similar to the C language library function **sscanf()**.

Common conversion characters for **\$format** and **\$scan** are **%d** for decimal (integer) value, **%c** for character, **%s** for string, and **%f** for floating point value.

Example (notice the brackets [ ] around the entire command which are *not* used around regular library functions, but are REQUIRED for **\$format** and **\$scan**):

```
EXPR output_text {[ $format "%s times %s is %d" "two" "five" 10 ]}
```

The example above constructs the string "two times five is 10" and writes it to the animation "output\_text" which must be the text animation of a Text-IO object. The **\$format** command *always* returns a string even if the format specification is simply a conversion to an integer or float.

Example (second example):

```
WHEN input_done == 1
    EXPR match_count {[ $scan input_text "%s %s %s %s %f" #a #b #c #d
#e ] }
    END
```

The example above assigns the first space delimited token in "input\_text" to the name "a", the second token to "b", the third token to "c", etc. The pound sign character ('#') in front of each destination name is required in the **\$scan** statement. It indicates that each destination is being passed by reference so that it can be modified by the **\$scan** command. A destination without a pound sign character ('#') preceding it is passed by value which means the command only gets the absolute value for the name and not a reference to the name. This will cause the **\$scan** to fail.

**NOTE:** Notice that both example use curly brackets around the third parameter!

The **\$format** and **\$scan** commands take format strings in the same syntax as the C language library functions **sprintf()** and **sscanf()**. If you are interested in all of the possible options, please see a [C Library Reference Manual](#) for **sprintf()** and **sscanf()**. Likewise, all of the supported library functions described above follow the conventions of their C Library equivalents.

**NOTE:** Be careful when using EXPR to assign the value of one text string to another. If the string to be assigned consists solely of numbers, then EXPR will try to turn that string into a number. This can cause problems for very large numbers. The workaround is to use the SET statement.

## SET (Alt + S)

Syntax Help: **SET** *result operand*

Code Assistant: **SET**  =

A SET statement lets you set the value for an animation/stimulus/control/global/parameter/local name. If the name is not a global and not a routine parameter/local name, this can change the state of an animation and generate an event that can trigger the execution of other control blocks or start/stop timer stimulus. To create a SET statement:

1. Insert a name in the first field.
2. Insert a value or name into the second field.

The = is static; it does not indicate a pull-down menu. If a name is entered into the second field, the current value for that name is assigned to the name in the first field when the statement executes. If the name in the second field has no value, 0 is used.

**NOTE:** The = is not used in the text editor. It is not part of the syntax.

## MATH (Alt + M)

Syntax Help: **MATH** *result operand1 (+,-,\*,/,%) operand2*

Code Assistant: **MATH**  =  **+ ▾**

The MATH statement is used to set an animation/stimulus/control/global/parameter/local name to a simple mathematical expression (usually involving another animation/stimulus/control name). Instead of setting a name's value to a specific number, a MATH statement can change a name's value by a particular formula (e.g., increasing the value by 1). As with the SET statement, execution of this statement generates an event for the name with a new value if the name is not a global or routine local. To create a MATH statement:

1. Insert a name into the first field.
2. In the second field, insert a name or a value. It can be the same name entered in Step 1. If the name has no value at the time of the statement's execution, 0 will be used.
3. Choose a math sign from the pull-down menu initially labeled +. Choices include: + Plus, - Minus, \* Times, / Divided by, % Remainder, or No Operator.
4. In the third field, insert a name or a value, as in step 2. It can be the same as the name entered in step 1 or 2.

**NOTE:** The = is not used in the text editor. It is not part of the syntax.

## LOOP (Alt + U)

Syntax Help: **LOOP** animation (==,<,>,>=,!=) value [OR,AND ...]  
Code Assistant: **LOOP**  ==

Use the LOOP statement to repeat a section of control while a test condition is true. A LOOP statement is used in conjunction with an END statement. The first time a running control block encounters a LOOP statement, it checks the WHILE condition to see if it is true. If so, Altia Design will execute each statement in the LOOP control block. When it reaches the end of the LOOP control block, Altia Design will check the WHILE condition to see if it is still true. If so, the whole LOOP control block will again be executed.

**NOTE:** When a LOOP is executing, new stimulus events are allowed to execute on each iteration of the loop. This is not the case with other types of control blocks.

## CALL (Alt + D)

Syntax Help: **CALL** name {parameter list} result  
Code Assistant: **CALL**  Name:   
Parameters:   
Result:

A CALL statement executes the named routine, giving it the values for the listed parameters. If filled in, the name in the RESULT field will be given the value returned by the routine. To use the CALL statement, fill in the routine name to be called, any required parameters, and the name to receive the routine's result. The next two examples demonstrate the parameter passing convention used by the CALL statement.

The following examples use this ROUTINE:

```
ROUTINE test {message num_times} {}
    LOOP num_times > 0
        SET Change = num_times
        MATH num_times = num_times - 1
        END
        SET output_text = message
    RETURN 999
END
```

Example of CALL:

```
CALL test {"testing..." 10} value
```

In this example of CALL invoking the ROUTINE **test**, parameter **message** will have a value of "Testing..."

(which is a string) and **num\_times** will have a value of **10** (which is an integer). Upon returning, **999** will be assigned to the name **value**. In this example, the parameters are absolute values instead of names. They can also be names in which case the name itself is substituted as the parameter when the ROUTINE executes. This is referred to as passing a parameter by reference versus by value.

Example of CALL:

```
SET count = 10
CALL test {"testing..." count} value
```

In this example of CALL invoking the ROUTINE **test**, parameter **num\_times** will reference the name **count**. The name **count** will always have a value of **0** upon returning from the CALL. This is because **count** is automatically substituted as the second parameter in the ROUTINE and the ROUTINE decrements its second parameter until it has a value of 0.

## RETURN (Alt + T)

Syntax Help:	RETURN <i>result</i>
Code Assistant:	RETURN

The RETURN statement allows a ROUTINE block to return to its caller prior to the end of the routine and (optionally) return a value that will be assigned to the name in the matching CALL statement's **Results** field. The RETURN statement can appear anywhere within a ROUTINE block any number of times with different return values. The RETURN value can be a name (animation, stimulus, global, routine local or routine parameter) in which case the value associated with the name is the return value.

## END (Alt + E)

Syntax Help:	END
Code Assistant:	END (no parameters for this statement)

An END statement is used to either finish a control block (WHEN or ROUTINE) or finish a control sub-block (IF, ELSE, ELSEIF, or LOOP). For example, when you insert an IF statement, all statements entered after it will be indented underneath it and contingent upon it. When you insert an END statement to finish an IF sub-block, the next statement entered will have the same left margin as the IF statement and will not be contingent upon it, unless it is an ELSE or ELSEIF statement.

## 6.6.5 File I/O Advanced Statements

The File I/O Advanced Statements are also first level statements which can be placed in WHEN or ROUTINE blocks. These statements are:

- OPEN
- WRITE
- READ
- CLOSE

Advanced Control statements are accessed through the **Advanced** group in the Statement Palette. This group contains different advanced command groups. The File I/O statements are contained in the **FILE I/O** group.

### File Names

File names used with the File I/O statements may be:

- a string value in double quotes, like "myfile.txt"
- a control name (parameter/local/global) containing a string
- a Text I/O object's text animation name whose string is to be used
- a control name or Text I/O object's text animation name preceded by '@' (i.e., @name), meaning use that name's value (which should be a string) as the name whose value (also a string) is to be substituted.

### OPEN

Syntax Help:	OPEN <i>filename (r,w,a+)</i>		
Code Assistant:	OPEN	File Name:	READ ▾

With an OPEN statement, you can access a text file to read/write. A READ statement also opens a file if it is not already opened. To complete an OPEN Statement:

1. Type in the file name (enclosed in double quotation marks) you wish to open.
2. From the pull-down menu (initially labeled a+), choose the way in which the file will be used, either r/Read to read from the file; w/Write for writing to the beginning of the file; or a+ Append for writing to the end of the file.

## WRITE

Syntax Help: **WRITE** animation (TAB,SPACE,RETURN) filename

Code Assistant:

WRITE	File Name:	<input type="text"/>
	Write:	<input type="text"/>
	Terminate with:	RETURN <input type="button" value="▼"/>

Use a WRITE statement to copy information received in, or generated by, your design to a text file. WRITE writes the given value to the given text file, appending the specified delimiter. To complete a WRITE Statement:

1. Type in the first field either a name or a text string enclosed in double quotation marks (e.g., "Hello World"). If you insert a name, the value for the name at the time the WRITE statement is executed will be written to the indicated file in its textual form.
2. From the pull-down menu labeled Tab, choose a means of delimiting the text, either Space, Tab, Return, or None.
3. Type into the second field the name of the file to which the text will be written. The file name must be between double quotation marks.

## READ

Syntax Help: **READ** animation (TAB,SPACE,RETURN) filename

Code Assistant:

READ	File Name:	<input type="text"/>
	Read To:	<input type="text"/>
	Read until first:	RETURN <input type="button" value="▼"/>

Use a READ statement to set the given animation/stimulus/control/global/parameter/local name to the value read from the file. The text in the file should be numeric or a double-quoted string, delimited by the chosen delimiter. Like the SET and MATH statements, execution of this statement generates an event for the name with the new value if the name is not a global or routine local. To complete a READ Statement:

1. Type into the first field the name that will receive the input as a value.
2. From the pull-down menu labeled Tab, choose a means of delimiting the text, either Space, Tab, Return, or None.
3. Type into the second field the name of the file from which the value or string will be read. The file name must be within double quotation marks (e.g., "myfile.txt").

## CLOSE

Syntax Help:	<b>CLOSE</b> <i>filename</i>
Code Assistant:	<b>CLOSE</b> File Name: <input type="text"/>

A CLOSE statement is used to close a text file that you have accessed in a control block. To complete a CLOSE Statement simply type the name of the file to be closed into the CLOSE field. The file name must be within double quotation marks (e.g. “myfile.txt”).

**NOTE:** You may use an Open and one or more Write statements in a control block to access a file and write to it. However, if you want to read that data from the file later in a control block, you must insert a Close statement at some point before the first Read statement to ensure that the information at the beginning of the file will be available for reading.

### 6.6.6 Views Advanced Statements

The Views I/O Advanced Statements are also first level statements which can be placed in WHEN or ROUTINE blocks. These statements allow the opening and closing of additional views (i.e., windows) into the current design. In addition, there are statements to open and close new Altia Design files. These statements are:

- OPEN\_VIEW for opening a new view into the current design.
- CLOSE\_VIEW for closing a view.
- OPEN DESIGN for opening a new design file in the current Altia session.
- CLOSE DESIGN for closing a design file.

Advanced Control statements are accessed through the **Advanced** group in the Statement Palette. This group contains different advanced command groups. The Views statements are contained in the **VIEWS** group.

## OPEN\_VIEW

Syntax Help:

```
OPEN_VIEW id x y (Root,Main,ID) {[winid]} width height winX winY  
(Toplevel,Borderless,Message,3DBorder,Dialog,FullScreen) magnification  
viewname
```

Code Assistant:

The image shows a software interface for creating an OPEN\_VIEW. It has a title bar 'OPEN\_VIEW'. Below it are several input fields: 'ID:' with a dropdown arrow, 'Name:' with a dropdown arrow, 'X:' and 'Y:' with dropdown arrows, 'Width:' and 'Height:' with dropdown arrows, 'Window X:' and 'Window Y:' with dropdown arrows, 'Magnification:' with a dropdown arrow, 'Style:' with a dropdown arrow containing 'Toplevel', and 'Window:' with a dropdown arrow containing 'Root' and 'Window ID:' with a dropdown arrow.

OPEN\_VIEW opens a new view into the current design. The new view is a completely separate window and usually displays a different portion of the design.

A unique identification number (id) is assigned to the new view. If id is the number of an existing view, that view is updated according to the remaining values in the OPEN\_VIEW statement. The ID cannot be 0 because this ID is reserved for the main view.

The area of the design that should appear in the view is specified with x and y, the design file coordinates that should appear at the lower left corner of the view. If unspecified, the defaults for the main view are used.

A new view is positioned relative to an existing reference window. The reference window can be the Root Window, which is the screen itself, the Main Window, which is the main Altia view, or another view window that has a specific ID number. Use the ref win menu to select the appropriate option. The relative position of the new view to the reference window's lower left corner is specified by setting ref x and ref y. If unspecified, the center of the reference window is chosen.

The new view's width and height are given in screen pixels. If left empty, the width and height of the main view are used.

The window style for a new view is specifying a style from the following types:

**Toplevel** A window with a Toplevel style is independent from its reference window

**Borderless** This style of window has no borders and it stays on top of the main Altia window. The user cannot move it, resize it or close it.

**Message** This style of window has a title and border and it stays on top of the main Altia window. The user can move it, but cannot resize it or close it.

<b>3DBorder</b>	This style of window has no title, just a border, and it stays on top of the main Altia window. The user cannot move it, resize it or close it. On same as the Borderless style because a border without a title is not Solaris or Linux running the X11 Windowing System, this style is supported by most X11 window managers.
<b>Dialog</b>	This style of window has a title and close option and it stays on top of the main Altia window. The user can move it and close it, but cannot resize it.
<b>FullScreen</b>	This style of window takes over the entire display. It has no title or borders. The user cannot move it, resize it or close it.

The magnification factor for the view is specified by setting mag (which can be a floating point value, such as .75). If empty, the magnification of the main view is used.

The name field specifies the name of the view. Enter text between double quotes (e.g., "Next View") or an animation/stimulus/control/global/local/parameter name. If unspecified, the view will have a name of the form "UserView#[id]".

Please note that the id, x, y, width, height, ref x, ref y, and mag fields may be numeric or any animation/stimulus/control/global/local/parameter name whose current value is numeric.

## CLOSE\_VIEW

Syntax Help:	<b>CLOSE_VIEW</b> <i>id</i>
Code Assistant:	<b>CLOSE_VIEW</b> ID: <input type="text"/>

**CLOSE\_VIEW** closes a view previously opened with **OPEN\_VIEW**.

The id field is the identification number of the view to close. It must be an ID number for a view previously opened with an **OPEN\_VIEW** statement. An ID of 0 is ignored because it refers to the main view. To close an Altia Runtime session, use a **SET** statement that looks like:

```
SET altiaQuit 1
```

## **OPEN DESIGN**

Syntax Help: **OPEN DESIGN** *id filename*

Code Assistant: **OPEN DESIGN** File Name:   
ID:

**OPEN DESIGN** opens a new design file in the current Altia session.

A unique identification number (*id*) is assigned to the new design. If *id* is the number of a design already open, that design is replaced by the new one and all its views are updated appropriately.

The FILE field may be a string between double quotes or any global/local name whose current value is a string. In addition, a text I/O object's text animation name can be used, as can @name, meaning use that name's value as the name whose value is to be substituted.

After a design is opened with **OPEN DESIGN**, additional views into the new design can be opened using the **OPEN VIEW** statement.

A Note On the **altiaInitDesign** animation:

After all objects in a new design have been loaded and initialized, Altia automatically generates an event with the name altiaInitDesign with a value equal to [id]. Creating a WHEN block to catch this event provides an opportunity to do further initialization of the design. For example, the WHEN block might contain one or more SET statements to clear text I/O objects in the new design. A design can have any number of WHEN blocks for altiaInitDesign. Also remember that the value for the event is equal to [id]. This allows different WHEN blocks to execute when different design files are loaded. When the main design is loaded, the value for the event is 0. To execute such a WHEN for any design file load, leave the second field of the WHEN statement empty.

A Note On the **altiaSetDesignId** animation:

**OPEN DESIGN** sets an internal Altia animation named altiaSetDesignId to the value of [id]. Subsequent **OPEN VIEW** statements check this value when they execute to decide which design a new view belongs to. Another Views statement should be available to change the active design ID value at any time, but is not yet available. Instead, the internal altiaSetDesignId animation must be changed with a SET statement that looks like (example changing the value of [id] to 15):

```
SET altiaSetDesignId 15
```

## CLOSE\_DESIGN

Syntax Help: **CLOSE\_DESIGN** *id*

Code Assistant: **CLOSE\_DESIGN** ID:

**CLOSE\_DESIGN** closes a design previously opened with **OPEN\_DESIGN**.

If one or more views are opened for the design with the identification number *id*, the views are closed automatically. An ID of 0 is ignored because it refers to the main view. To close an Altia session, use a **SET** statement that looks like:

```
SET altiaQuit 1
```

## 6.6.7 Miscellaneous Advanced Statements

The Miscellaneous I/O Advanced Statements are also first level statements which can be placed in WHEN or ROUTINE blocks. These statements are:

- EXTERN

Advanced Control statements are accessed through the **Advanced** group in the Statement Palette. This group contains different advanced command groups. The Miscellaneous statements are contained in the **MISC** group.

## EXTERN

Syntax Help: **EXTERN** *name {parameter list} library (STRING/INT/NONE) result*

Code Assistant: **EXTERN** Parameters:   
Library:   
Function:   
Result:  INTEGER ▾

This statement is used to execute a routine residing in a **DLL** (Dynamically Linked Library). The **DLL** must be a 32-bit type. Calling 16-bit **DLL** routines is not supported and may result in unpredictable behavior. External DLL functions must be **\_\_stdcall** type functions. When the DLL is linked, you must use a **.def** file or the **/EXPORT:myFunc** compile-time option.

The EXTERN statement executes the named routine from the named **DLL** library. One or more parameters may be passed to the routine by placing values or names in the PARAMS field using spaces to separate the parameter items. If filled in, the name in the RESULT field will be given the value returned by the external routine. The type of the return value from the routine must be provided to ensure a correct conversion to an animation, stimulus, global, local, or parameter name type. Possible types are String, Integer, or None.

If you do not properly match the number of parameters with the number expected by the external routine, or a parameter is holding the wrong type of data (e.g., a string instead of an integer) at the time of the EXTERN statement execution, the outcome is unpredictable!

The following EXTERN example will call the **altiaSleep()** in the Altia supplied 32-bit **altdde32.dll**, to sleep 500 milliseconds:

```
EXTERN "altiaSleep" {500} "altdde32.dll" NONE {}
```

A full path is not required with the **DLL** file name in this example because the **altdde32.dll** file is located in the Windows directory after a normal Altia Design product installation. By default, this directory is part of the search path for **DLL** files.

# Chapter 7: Connections

## 7.1 Connections Overview

Connections are a quick and easy way for Altia objects to communicate with each other or with the outside world (simulation models and code). Connections allow objects to send events to one another without the objects needing to be aware of each other's animation, stimulus, or control code names. The connection acts as a "translation" layer between the objects or the outside world. For example, the stimulus name that executes when the mouse is dragged over a knob object can have an output connector defined for it. It can be linked (connected) to a meter object's input connector that is tied to the meter's needle movement animation.

**NOTE:** Creating connections in Altia Design 11 is performed in a slightly different manner than in older versions of Altia Design.

In prior versions of Altia Design, each object had its own Connection list window. In Altia Design 11, all selected objects are displayed in a single list within the Connections pane. Connecting two objects together is as easy as before - simply highlight an output (plug) and an input (socket) using Ctrl + Click, then click the Connect button.

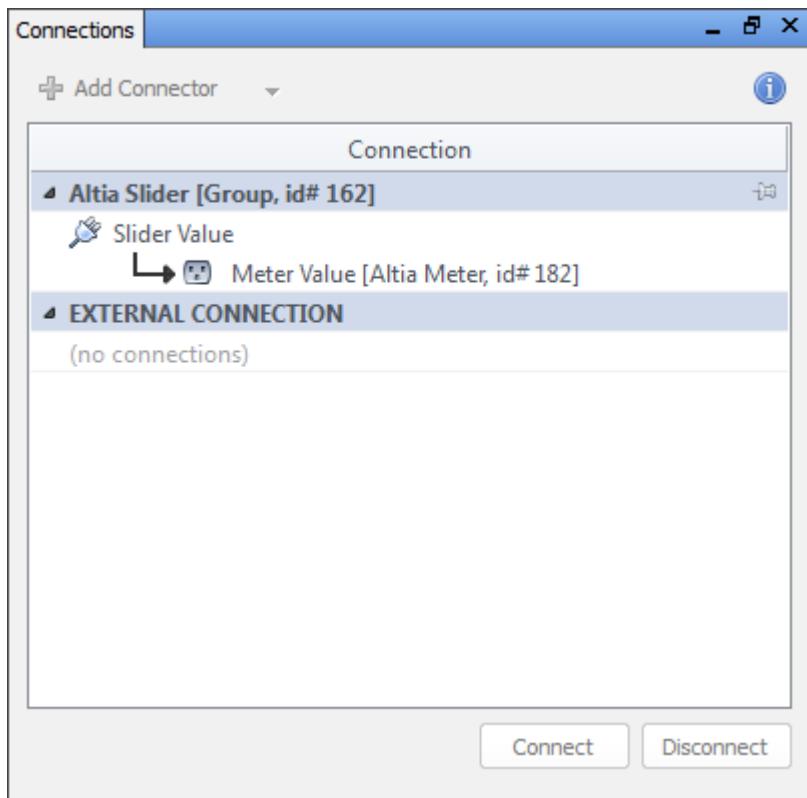
If the objects you wish to connect together are not at the same focus level, you can pin an object in the Connections pane list. Pinned objects will remain displayed at the top of the list. To make a connection, simply pin the first object, then navigate within the Universe window to the second object and select it. Now both objects will be visible in the Connections pane list and you can select the Connectors you wish to link to create a Connection.

The **Connections** pane provides options to view, modify, and create connections between selected object(s), or external (e.g., client application program) signals. If an object has input  or output  connectors defined for it, they appear in the list area of the Connections pane. If an input or output is linked to another object's output or input or an external output or input, this is also shown in the list area. From the Connections dialog, inputs and outputs can be linked together, created, or modified.

**NOTE:** We're aware that the metaphor of the electrical plug (output) icon and socket (input) icon isn't technically accurate, but it's easy to understand that plugs can only connect to sockets and vice versa.

In Altia connections, the direction data travels goes from the plug (sender) to the socket (receiver).

## 7.2 Using the Connections Pane



### 7.2.1 Overview

The objects to be connected must have input connectors and/or output connectors defined for them. If an object does not have any inputs or outputs, the list area for that object will indicate "(no connections)". Adding connectors to an object is described later in this chapter. If multiple objects are selected in the Editor view, each object (indicated by the blue heading row) will display its input and output connections as shown below.

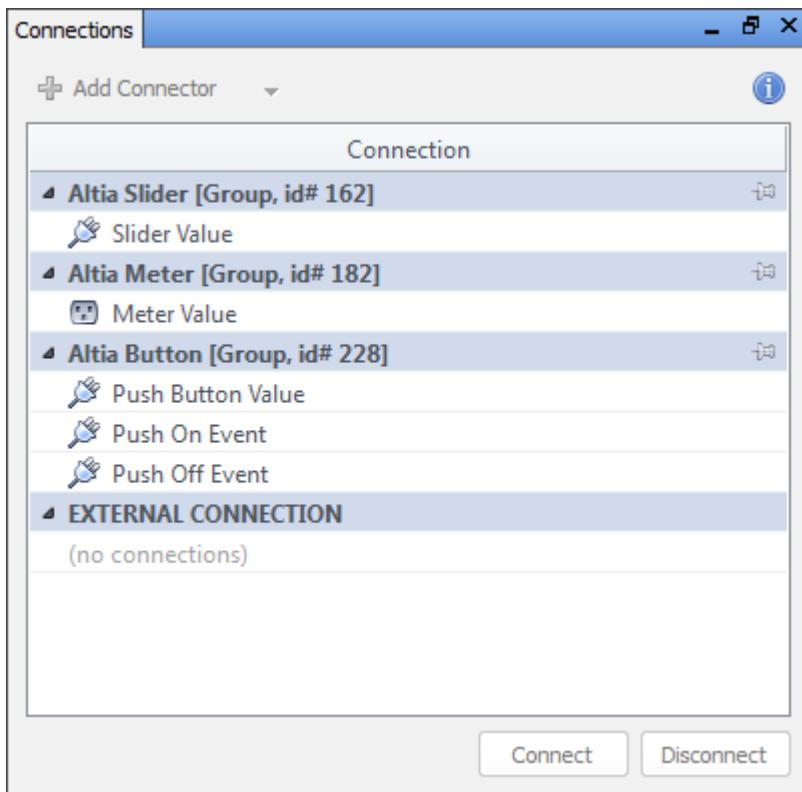


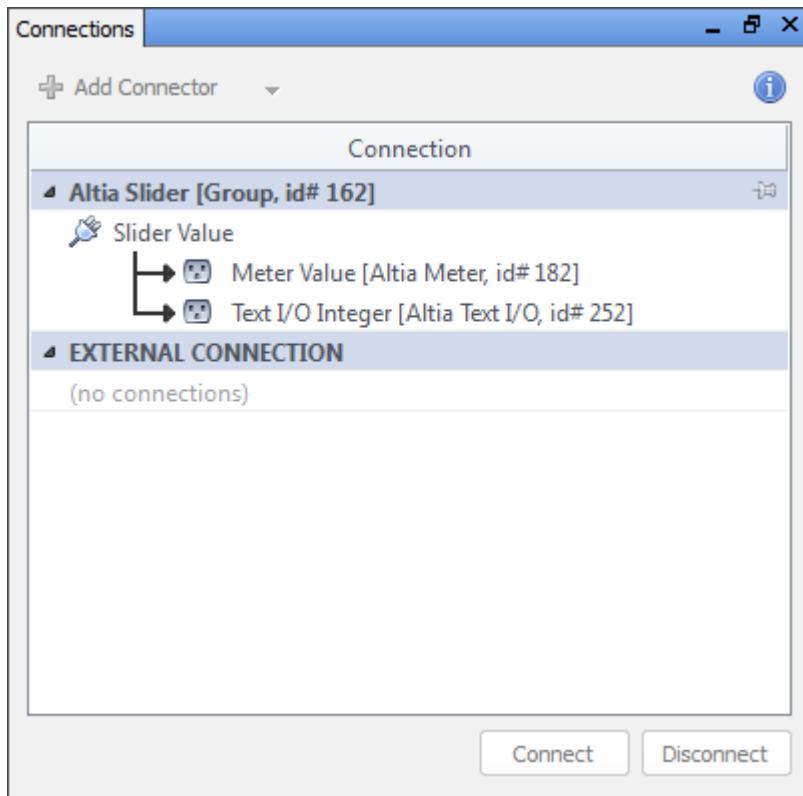
Figure 9-1: Connections pane when multiple objects are selected in the Editor view

The Connections pane displays two basic types of information for each selected object. The rows that start with an input or output icon list the inputs and output connectors defined for that object. These rows also display the input/output's name.

The indented rows that display an arrow describe other input/outputs that are connected to the above input/output and indicate the direction of the data flow. We call these arrows links since they represent data links to other objects. The link arrows list the name of the input/output that is connected and the name and id number of the object that owns the connector. If the object in the above row is not connected to another object, no link row will be displayed. If the object's connection is to the external world then "[external]" is displayed.

The Connections pane list also displays all External connectors and their connections (if any) below the selected objects. If there are no External connectors defined for the current model, the Connections pane list will display "(no connections)" under the EXTERNAL CONNECTION heading row. See [Figure 9-1](#) above.

An object may link to multiple objects. For example, you may wish to have a slider object linked to both a meter object and a text I/O Integer display object (as shown below).



## 7.2.2 Linking Two Objects Using Connections

To connect two objects at the ***same hierarchy (focus) level*** together, perform the following steps.

1. In the Editor view, select the two objects you want to connect.
2. Both objects and their connectors will appear in the Connection pane's list.
3. Click on the first object's input or output connector row. Ctrl-click on the second object's input/output connector row. Note that an input connector can only be connected to an output connector and vice versa.
4. Both rows will now be highlighted and the **Connect** button at the bottom of the Connections pane will become active.
5. Press the **Connect** button and the objects will be connected.

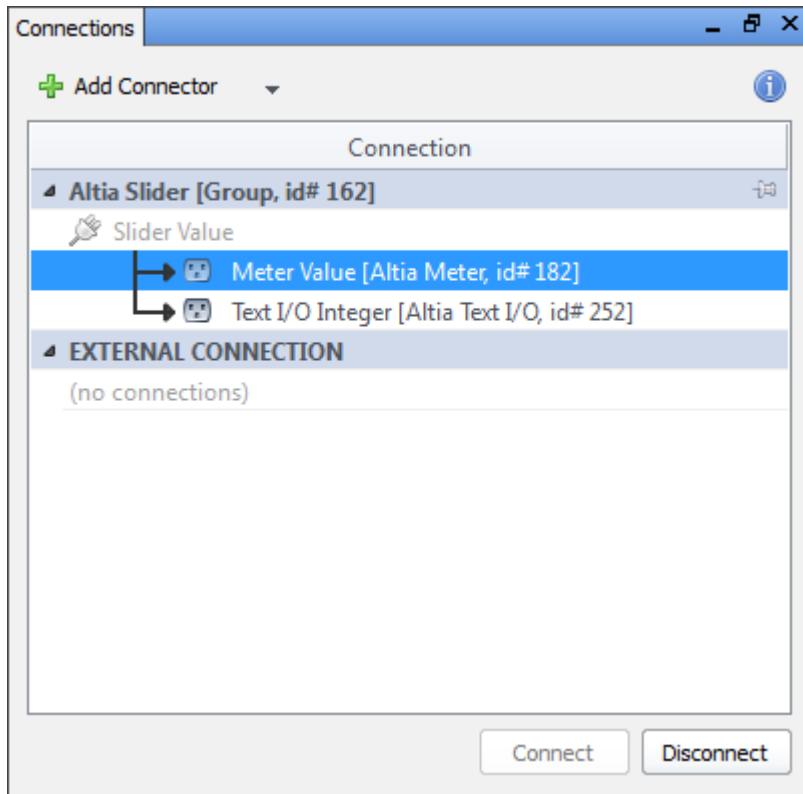
To connect two objects at ***different hierarchy (focus) levels*** together, perform the following steps.

1. In the Editor view, select the first of the two objects you want to connect.
2. That object and its connectors will appear in the Connection pane's list.
3. In the Connection pane list, on the right-hand side of the objects blue heading row, click the pin icon to temporarily pin that object to the Connection pane's list.
4. Navigate to the second object you wish to connect, and select it in the Editor view.
5. The second object will appear below the pinned first object in the Connection pane's list.
6. Click on the first object's input or output connector row. Ctrl-click on the second object's input/output connector row. Note that an input connector can only be connected to an output connector and vice versa.
7. Both rows will now be highlighted and the **Connect** button at the bottom of the Connections pane will become active.
8. Press the **Connect** button and the objects will be connected.

## 7.2.3 Unlinking a Connection Between Two Objects

To disconnect two connected objects, perform the following:

1. In the Editor view, select either of the objects you wish to disconnect.
2. Select the “link” row that you want to break (links are denoted by the indented row with the arrow). Be sure to select the link row and not the connector row (as shown below).
3. Press the **Disconnect** button. This will disconnect the two objects and remove the links.



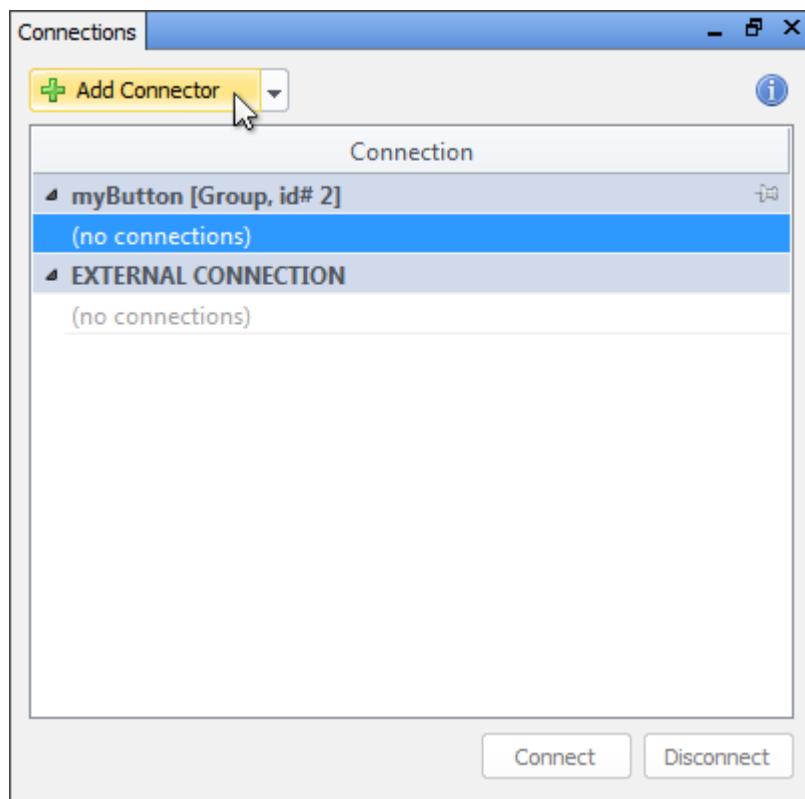
## 7.2.4 Showing the Objects Associated With a Link

To quickly select the object to which a link is connected to, right-click on the link row in the Connection pane's list and select **Show Connected Object**. This will instantly jump to and select the connected object, even if it exists at a different level in the focus hierarchy.

## 7.2.5 Adding a New Connector to an Object

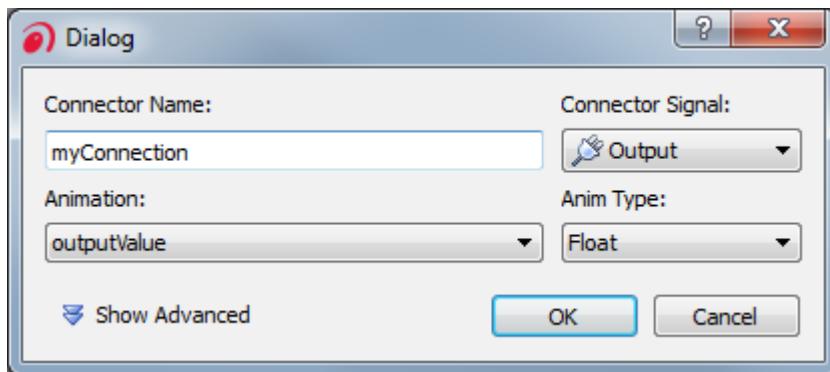
To add a new connector to an object, that object must have an existing animation, stimulus, or control code name defined on that object or on one of that object's children.

1. Select the object and its connectors (if any) will appear in the Connection pane's list. If the object does not have any connectors already defined, "(no connections)" will be displayed under the object's heading row in the Connection pane's list.
2. Select either an existing connector row, or the "(no connections)" row below the object's heading in the Connection pane's list.
3. Click on the **Add Connector** button



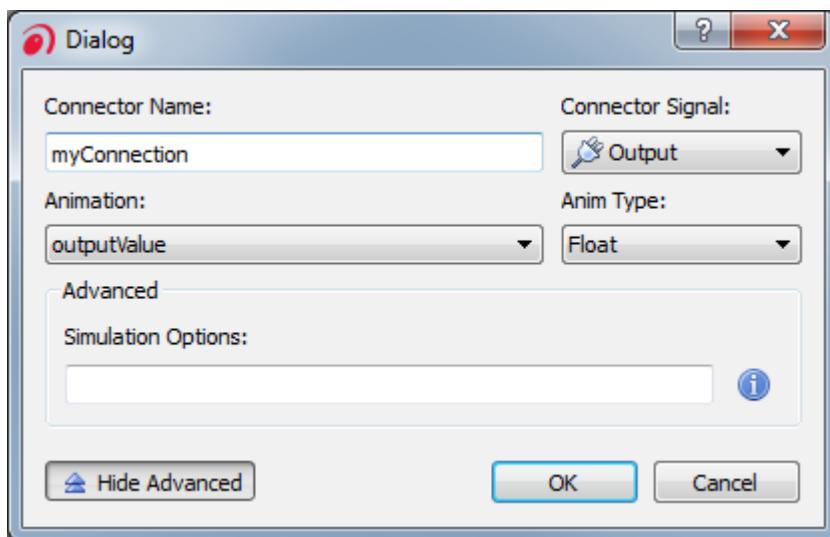
**NOTE:** If you wish to create a new External Connection connector, follow the same sequence as described above, but in Step 2, instead of highlighting the row below the object's heading row, select a connector row or the "(no connections)" row below the EXTERNAL CONNECTIONS heading.

The Add/Edit Connection dialog will then be displayed. Enter a name for the new connector, and select the animation, stimulus or control code name to associate with this connector from the Animation dropdown. You can create input connections that set the value for an animation/control/stimulus name; or you could create output connections that receive the value of an animation/control/stimulus name.



Next, indicate whether the connection is an input or an output using the Connector Signal dropdown. You can also set the type for the connection (integer or floating point). If this is an ordinary object with animation, stimulus, or control created directly from the Animation, Stimulus or Control Editor, the type may be either Integer or Float. However, you should always choose Float if the name can generate float values (e.g., 1.23 is a floating point number while 1, 2, 5, 10 are integers). Some newer special Altia objects have built-in animations that only accept or generate floating point numbers. If the name in the Animation field refers to one of these animations, then the Float type should be chosen.

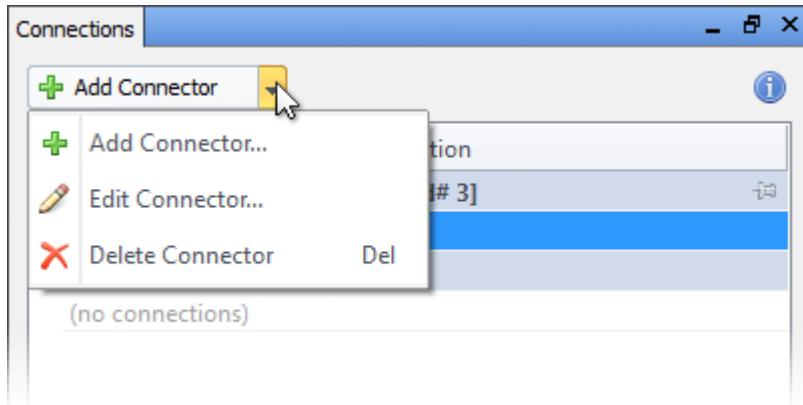
Click **Show Advanced** to reveal the **Simulation Options** field. This field is only used for the special needs of specific simulation product connections supported by Altia. Please refer to specific simulation connection documentation for further details. If you are not connecting your model to an external simulation product you will not need to enter anything in this field.



When you are satisfied with the settings for the new connector, press the **OK** button. The new connector will now appear below the object's heading row in the Connection pane's list.

## 7.2.6 Viewing, Modifying, or Deleting a Connection Definition

Highlight a connector row in the Connection pane's list, then select **Edit Connection...** option from the Connection pane dropdown menu to open the Edit Connector dialog. You can view the connector's settings and make changes to it if necessary.



To delete a connector, highlight it in the Connection pane's list and choose **Delete Connection** from the Connection pane dropdown menu. Please note that such a deletion *cannot* be undone so be very careful when performing this action.

## 7.3 External Connections

External Connections can be defined for use by external client programs and allow you to link these connections to specific Altia objects. Creating an External Connections connector is done in the same manner as creating a connector for an object. See [Section 7.2.5, Adding a New Connector to an Object](#).

Altia sells external connection packages custom tailored to the needs of various simulation products. These packages allow you to connect your Altia design to leading simulation software tools with the simple point and click connection dialog. No programming is necessary. Please visit our web page at [www.altia.com](http://www.altia.com) for a current listing of available Connection Packages.

Establishing links between Altia objects and external connectors (whether to simulation models or custom code) is similar to linking two objects together. When a link is established in Altia, signals generated by the Altia object are sent to the listening external object (and vice versa).

Instead of referencing a specific Altia object's animation, stimulus, or control name, a client program can reference an external connection's animation name. That is to say, a client program can send events to the external connection's animation name (e.g., using **AtSendEvent()** or **AtSendText()**) if the external connection is defined as an output. Similarly, the client program can receive events for the animation name (e.g., using **AtSelectEvent()** with **AtNextEvent()**) or by setting up a callback function with **AtAddCallback()**) if the external connection is defined as an input.

### 7.3.1 How External Connections Work

If an external connection is an output, Altia listens for events with the defined animation name. When such an event occurs, it is translated into one or more new events if the external connection is linked to one or more specific object connections. Each new event has the same value as the original, but the name is changed to match a linked connection's animation name. If the external input is linked to only one object connection, then only one translated event is generated. If there is more than one link, a translated event is generated for each link.

### External Connection Example

The clearest way to describe how external connections work is by using an example. Let's say we have a slider object from the **sliders.dsn** library. It has an output connection named **Slider Value** that is defined for the animation name **slide**. This connection is an output because slide events are generated by the object's stimulus definitions.

For our example, we will assume that an external connection has been defined, its name is **Desired Volume**, its animation name is **desired\_volume**, and it is an input (i.e., an input to a client program). We will also assume that the **Slider Value** connection has been linked to the external **Desired Volume** connection.

When Altia is in Run mode and the slider handle is moved with the mouse, the slider object generates slide events with specific values. Each time one of these events is generated, an event with the name `desired_volume` is also generated with the same value.

We can link another specific object connection to the same external connection. Let's say the second object is a knob from the `knobs.dsn` library and it has a connection for the animation `knob`. In this case, each knob event will generate a `desired_volume` event, but it will *not* generate a slide event. This is an important difference from the traditional approach where the client listens for slide events directly, the knob's animation is renamed to slide, and the knob rotates when the slider handle is moved and vice versa.

## 7.3.2 External Connection Animation Names

It is very important to understand that the animation names used in external connection definitions share the same name space with all other Altia object animation, stimulus, and control names. Hence, it is perfectly valid, but maybe not desirable, to define an external connection with an animation name that matches the name for a specific object animation.

As an example, you can have a meter object with a meter animation and also have an external connection output defined that uses meter as its animation name. If a client program sends a meter event to Altia, it will go to the meter object as well as the object(s) linked to the external connection. One way to avoid such a situation is to always use a special prefix, such as `extern_`, for your external connection animation names (for example, `extern_meter`) and avoid the use of this prefix for object animation, stimulus, or control names.

An advantage to the single name space is that you can easily monitor the value of an external connection animation (e.g., `extern_meter`) by simply typing the name into the Animation Editor's Name field and watching the value change in the State field. You can also easily simulate events from the client by simply changing the value in the State field. This monitoring and controlling via the Animation Editor only works if you avoid using spaces in your external connection animation names (for example, use `extern_meter` and not `extern meter`). Unlike object animation, stimulus, and control names, external connection animation names can contain spaces. This is certainly a way to guarantee uniqueness between names used by objects and external connection animation names, but the ability to monitor and control an external connection animation's value from the Animation Editor is lost because the Animation Editor does not allow spaces between characters entered into the Name field.

### **7.3.3 Advantages to Using External Connections**

An advantage to using external connections is that doing so hides specific object animation, stimulus, and control names from the client program. Instead of accessing individual animations directly, the client program is written to send and receive generic external connection animation names. These external connections can then be linked to any object connections independent of the actual animation names associated with the object connections. When an external connection's link needs to be changed (i.e., unlinked from one Altia object and linked to another), there is no need to do any animation renaming or code modification.

Another advantage of using external connections is that the client program's access to the design is defined in a central location. If you use external connections for all Altia/client communication, you can open the External Connections dialog and see what the links are between the client and the objects in your design. Using the Connected Object option from the Show menu of the External Connections dialog, you can also locate the specific object associated with a link.

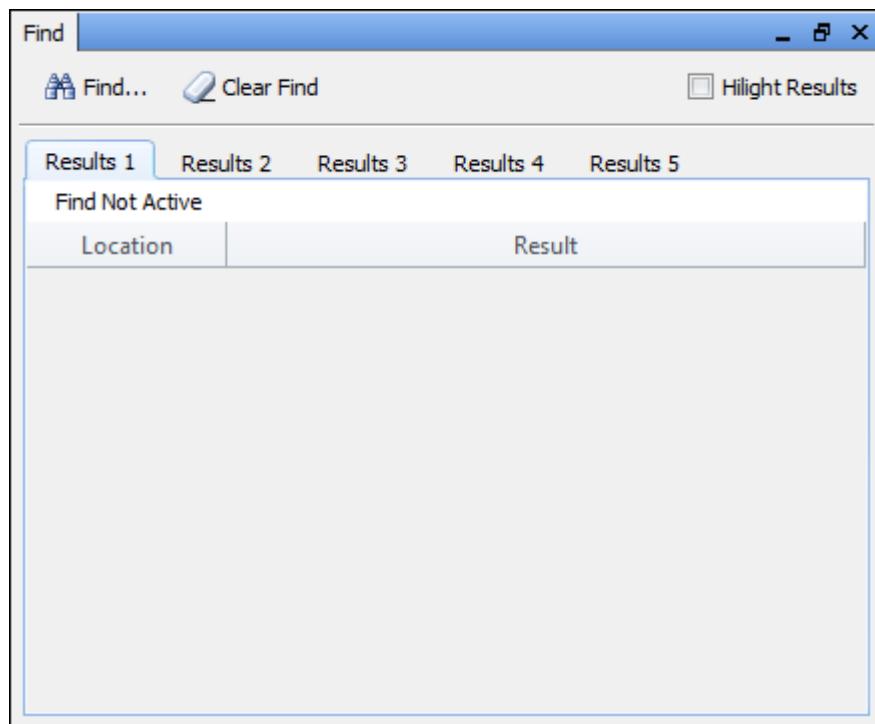
### **7.3.4 Using External Connections with Cloned Objects**

External connections cannot be used with cloned objects (as described in the [Altia API Reference Manual](#)) since clones are created dynamically and can therefore have no predefined links to external connections.

# Chapter 8: Find Pane

The Find pane is used to create custom searches in Altia Design for text, fonts, or objects. The search criteria can be saved to file and loaded at a future date.

If the Find pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Find** menu option is checked.



## 8.1 Find Overview

The Find Editor allows the User to create and execute up to five different find operations. Find operations tell the Find Editor what to search for in the Altia Design project. Each operation is independent from the other operations and displays its results in a dedicated **Results** tab.

### Defining a Find Operation

A find operation is defined used the **Criteria Dialog** by pressing the **Find...** button on the **Toolbar**. The following type of find operations are supported:

- **Text Find**
- **Font Find**
- **Object Find**

### Find Results

The results for a find operation are displayed in the **Results** tab. There are a few possible status messages when viewing the results:

- **Find Not Active** - There is no defined criteria for this **Results** tab.
- **Find Criteria Too Vague** - The Text Find criteria is too ambiguous because it matches whitespace.
- **Find X (no results found)** - The Find criteria did not yield any results where **X** is the type of find (text, font, object)
- **Find X (Y results in Z objects)** - Shows a summary of find results where **X** is the type of find (text, font, object), **Y** is the number of results found, and **Z** is the number of objects found.

A list of results will be displayed by object. Each result can be clicked which will make the object associated with the result the current selection. The clicked result will also be shown in the associated editor (i.e. Control Editor, Animation Editor, etc.).

### 8.1.1 Text Find

A text find operation will search the Altia Design project for objects that have matching text strings. The text can be located within an object including animation names, animation values, object identifier, property values, etc. The operation can be limited in scope to:

- Text string to search for (uses regular expressions with wildcards)
- Text case and whole word match
- Location within objects (i.e. control code, animation names, property values, etc.)

- Specific types of objects (i.e. decks, images, rectangles, etc.)
- Current selection or the entire project

### **8.1.2 Font Find**

A font find operation will search the Altia Design project for objects that have a matching font definition. The operation can be limited in scope to:

- Font family name to search for (uses regular expressions with wildcards)
- Font characteristics (i.e. bold, italic, size, etc.)
- Specific types of objects (i.e. decks, images, rectangles, etc.)
- Current selection or the entire project

### **8.1.3 Object Find**

An object find operation will search the Altia Design project for specific objects. The operation can be limited in scope to:

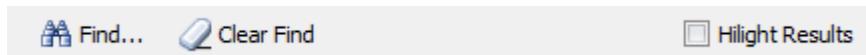
- Types of objects (i.e. decks, images, rectangles, etc.)
- Current selection or the entire project

## **8.2 Find Pane User Interface**

The Find pane consists of the following components:

- **Toolbar**
- **Results List**
- **Criteria Dialog**

## 8.3 Find Editor Toolbar



The Find Editor Toolbar has controls for starting and clearing a find operation as well as for managing how find results are displayed in the various editors.

### 8.3.1 Find Controls



The **Find...** Button will show the **Criteria Dialog**. The dialog allows the User to set the find criteria for the currently selected **Results** tab.



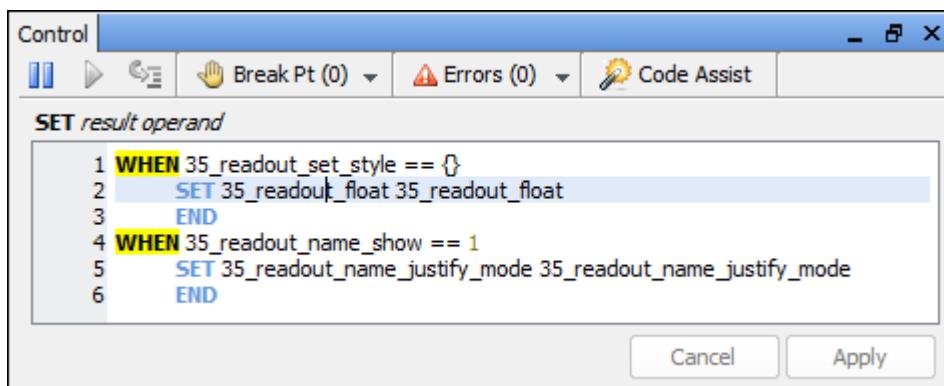
The **Clear Find** Button will erase the find criteria for the currently selected **Results** tab. The results for the currently selected **Results** tab will be removed.

### 8.3.2 Result Controls



The **Highlight Results** Checkbox toggles inline highlighting of find results in the different Altia Design Editors. For example, matching results in the Control Editor will be highlighted yellow when this checkbox is checked.

Example showing highlighted results in the Control Editor:



## 8.4 Find Results List

The Find Results list details the status of the current find operation.

Results 1		Results 2	Results 3	Results 4	Results 5
Find Text: when (20 results in 10 objects)					
Location	Result				
▲ Altia Readout (#35)					
Control Code	Line 1: WHEN 35_readout_set_style == {}				
Control Code	Line 4: WHEN 35_readout_name_show == 1				
▲ Altia Readout (#27)					
Control Code	Line 1: WHEN 27_readout_set_style == {}				
Control Code	Line 4: WHEN 27_readout_name_show == 1				
▲ Altia Readout (#19)					
Control Code	Line 1: WHEN 19_readout_set_style == {}				
Control Code	Line 4: WHEN 19_readout_name_show == 1				
▲ Rectangle (#17)					
Control Code	Line 4: WHEN {} == {}				

Up to five different find operations can be utilized at the same time. Each find operation has a tab control which can be used to activate it.

## Result Tabs

Results 1	Results 2	Results 3	Results 4	Results 5
-----------	-----------	-----------	-----------	-----------

The **Results** tabs specify which find operation is active. Changing tabs will change the active find operation. The results for the associated find operation will be shown in the status and list areas.

**NOTE:** Selecting a different Results tab will change the active find criteria. All the associated editors (i.e. Control, Animation, etc.) will update to reflect the criteria associated with the newly selected tab.

## Result Status

Find Text: when (20 results in 10 objects)
--

The status area shows the status of the find operation for the currently selected **Results** tab. The find operation may be inactive (cleared), invalid (too vague), or active (with results).

## Result List

Location	Result
Altia Readout (#35)	
Control Code	Line 1: WHEN 35_readout_set_style == {}
Control Code	Line 4: WHEN 35_readout_name_show == 1
Altia Readout (#27)	
Control Code	Line 1: WHEN 27_readout_set_style == {}
Control Code	Line 4: WHEN 27_readout_name_show == 1
Altia Readout (#19)	
Control Code	Line 1: WHEN 19_readout_set_style == {}
Control Code	Line 4: WHEN 19_readout_name_show == 1
Rectangle (#17)	
Control Code	Line 4: WHEN {} == {}

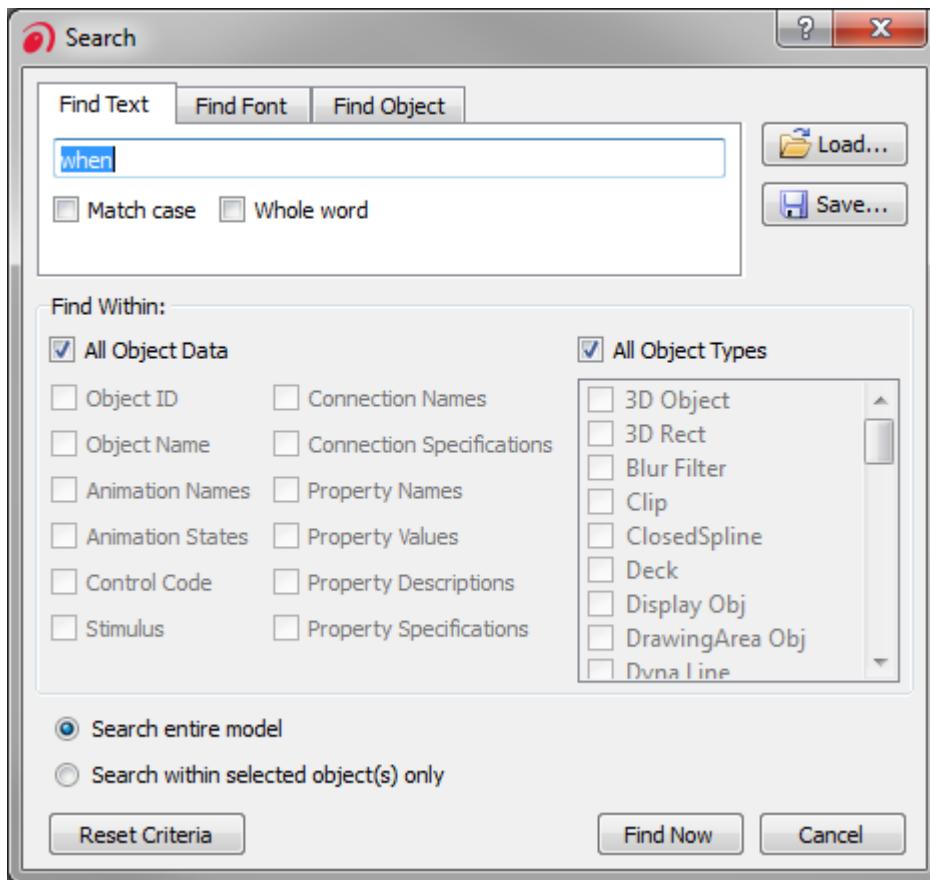
The result list shows all the matches found by the find operation. The list is grouped by object found in the Altia Design project. Multiple matches may be found for a single object as there are many locations to search for within a single object (i.e. animation names, property values, control code, etc.).

Each result can be clicked which will make the object associated with the result the current selection. The clicked result will also be shown in the associated editor (i.e. Control Editor, Animation Editor, etc.).

**NOTE:** The results are not "live" and will not update as the Altia Design project is changed. The results are those at the time the search was performed. To "refresh" the find results after changing the Altia Design project, rerun the find operation using the Find Criteria Dialog.

## 8.5 Find Criteria Dialog

The Find Criteria Dialog allows for the definition of a find operation.



### Find Type Tabs

The top Find Criteria Dialog contains a tab control to select one of three find operation types. This includes text search, font search, and object search. Refer to [Section 8.1, Find Overview](#) for details on the different find operation types.

### Find Criteria

For **Text** searches this is a regular expression to match. The match can optionally be whole-word and case-sensitive. For **Font** searches this is a regular expression for matching font family names. The match can optionally be tied to bold, italic, font size, DPI, and character set. For **Object** searches, there is no criteria expression to match.

See [Section 8.6, Find Expressions](#) for guidelines on constructing regular expressions.

## Search Locations

For **Text** searches this identifies where to search for text matches. The locations are:

- **All Object Data** - searches everywhere (disables specific search locations below)
- **Object Id** - searches the unique numerical ID for an object
- **Object Name** - searches the object's name property or the default assigned name.
- **Animation Names** - searches animation names for an object
- **Animation States** - searches animation values (numerical and string)
- **Control Code** - searches all control code for an object
- **Stimulus** - searches stimulus definitions for an object
- **Connection Names** - searches connection names for all connections of an object
- **Connection Specifications** - searches the animation names used in a connection definition
- **Property Names** - searches the names of all properties for an object
- **Property Values** - searches the values of all object properties (numerical and string)
- **Property Descriptions** - searches the descriptions of all object properties
- **Property Specifications** - searches the animation names used in all object property definitions

The **Font** and **Object** searches do not use a search location.

## Object Types

If the **All Object Types** checkbox is checked, the search will examine every object in the search. If the checkbox is not checked, then only the checked objects in the object list will be examined.

## Search Set

The find criteria can search the current selection or the entire Altia Design project.

## Load and Save

The find criteria can be loaded from or saved to a data file for future use, or to share with other Altia Design Users. Searches from previous versions of Altia Design can also be loaded (i.e. from the Navigator Find Panel in Altia 10.x).

## Reset

Pressing this button will reset the search criteria to default values.

## Find Now

Pressing this button will hide the Find Criteria Dialog and execute the find operation. A progress bar will appear while the find is in progress. The find could take a bit of time for large designs with broad search criteria. After the find operation is completed, the progress bar will disappear and the results will be shown in the **Results List**.

**NOTE:** The find criteria associated with each Results tab is NOT saved with the Altia Design project. The criteria will NOT persist from session to session in Altia Design. To save criteria for future use, use the Load/Save feature in the Find Criteria Dialog.

## 8.6 Find Expressions

Find expressions used in Text and Font find criteria are built up from groups of text, quantifiers, and assertions.

The simplest text is a character, e.g. **x** or **5**. The text can also be a set of characters enclosed in square brackets. **[ABCD]** will match an **A** or a **B** or a **C** or a **D**. This same text can be written as **[A-D]**. Text to match any capital letter in the English alphabet is written as **[A-Z]**.

A quantifier specifies the number of occurrences of an expression that must be matched. The quantifier **{1,1}** means match "one and only one". The quantifier **{1,5}** means match "at least one, but no more than five".

An assertion does not match characters. Instead an assertion matches positions in a text string. The assertion **^** means the "start of the string". The assertion **\$** means the "end of the string".

### Example:

To match integers in the range 0 to 99:

The expression **[0-9]{1,1}** matches a single digit exactly once which will find integers in the range 0 to 9. The expression **[0-9]{1,2}** will match integers from 0 to 99, but it will also match integers that occur in the middle of strings.

The expression **^[0-9]{1,2}** will match integers from 0 to 99 that are at the **start** of a string.

The expression **^[0-9]{1,2}\$** will match integers from 0 to 99 that comprise the entire string from start to end.

Some sets of characters and some quantifiers are so common that they have been given special symbols. The text **[0-9]** can be replaced with the symbol **\d**. The quantifier **{1,1}** can be replaced with the expression itself, i.e. **x{1,1}** is the same as **x**. The quantifier **{0,1}** can be replaced with a question mark, i.e. **x{0,1}** is the same as **x?**.

### Example:

To match integers in the range 0 to 99:

The expression `^\d{1,2}$` will match one or two digits that comprise the entire string from start to end. The expression `^\d\d{0,1}$` will match one digit at the start of the string, followed immediately by 0 or 1 digits at the end of the string. The expression `^\d\d?$` will behave the same as `^\d\d{0,1}$`.

The qualifier symbol `|` will join groups of text together in an **or** operation. Parentheses group text together and identify a part of the expression that we wish to match. Enclosing the text in parentheses allows it to be used as a component in more complex expression. The assertion symbol `\b` means "match a word boundary". A word boundary is any non-word character such as a space, newline, or the beginning/end of a string.

**Example:**

To write an expression that whole-word matches one of the words 'dog' **or** 'cat' **or** 'bird':

The expression `dog` will match any occurrence of "dog" in a string.

The expression `dog|cat|bird` will match any occurrence of "dog" **or** "cat" **or** "bird" in a string.

The expression `\b(dog|cat|bird)` will match any words that start with "dog" **or** "cat" **or** "bird". The expression `\b(dog|cat|bird)\b` will match any whole word that is exactly "dog" **or** "cat" **or** "bird".

## Special Text Characters

Characters and Abbreviations for Sets of Characters	
Element	Description
<code>c</code>	A character represents itself unless it has a special meaning. E.g. <code>c</code> matches the character <i>c</i> .
<code>\c</code>	A character that follows a backslash matches the character itself, except as specified below. E.g., To match a literal caret at the beginning of a string, write <code>\^</code> .
<code>\a</code>	Matches the ASCII bell (BEL, 0x07).
<code>\f</code>	Matches the ASCII form feed (FF, 0x0C).
<code>\n</code>	Matches the ASCII line feed (LF, 0x0A, Unix newline).
<code>\r</code>	Matches the ASCII carriage return (CR, 0x0D).
<code>\t</code>	Matches the ASCII horizontal tab (HT, 0x09).
<code>\v</code>	Matches the ASCII vertical tab (VT, 0x0B).
<code>\x hhhh</code>	Matches the Unicode character corresponding to the hexadecimal number <i>hhhh</i> (between 0x0000 and 0xFFFF).

<code>\0 ooo</code> (i.e., \zero <i>ooo</i> )	Matches the ASCII/Latin1 character for the octal number <i>ooo</i> (between 0 and 0377).
<code>.</code> (dot)	Matches any character (including newline).
<code>\d</code>	Matches a digit.
<code>\D</code>	Matches a non-digit.
<code>\s</code>	Matches a whitespace character.
<code>\S</code>	Matches a non-whitespace character.
<code>\w</code>	Matches a word character.
<code>\W</code>	Matches a non-word character.
<code>\n</code>	The <i>n</i> -th >backreference, e.g. \1, \2, etc.

## Sets of Characters

Square brackets [ ] mean match any character contained in the square brackets. The character set abbreviations described above can appear in a character set in square brackets. Except for the character set abbreviations and the following two exceptions, characters do not have special meanings in square brackets.

- The caret (^) negates the character set if it occurs as the first character (i.e. immediately after the opening square bracket). `[abc]` matches ‘a’ or ‘b’ or ‘c’, but `[^abc]` matches anything *but* ‘a’ or ‘b’ or ‘c’.
- The dash (-) indicates a range of characters. `[W-Z]` matches ‘W’ or ‘X’ or ‘Y’ or ‘Z’.

## Special Quantifiers

By default, text is automatically quantified by `{1,1}`, i.e. it should occur exactly once. Text is a character, or an abbreviation for a set of characters, or a set of characters in square brackets, or a set of text in parentheses. In the following list, **T** stands for an arbitrary set of text.

<code>T?</code>	Matches zero or one occurrences of <i>T</i> . This quantifier means <i>The previous text is optional</i> , because it will match whether or not the text is found. <code>T?</code> is the same as <code>T{0,1}</code> (e.g., <code>dents?</code> Matches ‘dent’ or ‘dents’).
<code>T+</code>	Matches one or more occurrences of <i>T</i> . <code>T+</code> is the same as <code>T{1,}</code> (e.g., <code>0+</code> matches ‘0’, ‘00’, ‘000’, etc.).

$T^*$	Matches zero or more occurrences of $T$ . It is the same as $T\{0,\}$ . The * quantifier is often used in error where + should be used.  For example, if $\backslash s^*\$$ is used in an expression to match strings that end in whitespace, it will match every string because $\backslash s^*\$$ means <i>Match zero or more whitespaces followed by end of string</i> . The correct expression to match strings that have at least one trailing whitespace character is $\backslash s+\$$ .
$T\{n\}$	Matches exactly $n$ occurrences of $T$ . $T\{n\}$ is the same as repeating $T$ $n$ times.  For example, $x\{5\}$ is the same as $xxxxx$ . It is also the same as $T\{n,n\}$ (e.g. $x\{5,5\}$ ).
$T\{n,\}$	Matches at least $n$ occurrences of $T$ .
$T\{,m\}$	Matches at most $m$ occurrences of $T$ . $T\{,m\}$ is the same as $T\{0,m\}$ .
$T\{n,m\}$	Matches at least $n$ and at most $m$ occurrences of $T$ .

To apply a quantifier to more than just the preceding character, use parentheses to group characters together in an expression. For example, **tag+** matches a ‘t’ followed by an ‘a’ followed by at least one ‘g’, whereas **(tag)+** matches at least one occurrence of ‘tag’.

**NOTE:** Quantifiers are normally “greedy”. They always match as much text as they can. For example, 0+ matches the first zero it finds and all the consecutive zeros after the first zero. Applied to ‘20005’, it matches ‘20005’.

## Assertions

Assertions make some statement about the text at the point where it occurs in the expression but they do not match any characters. In the following list  $T$  stands for any arbitrary set of text.

$\wedge$	The caret signifies the beginning of the string. If you wish to match a literal $\wedge$ you must escape it by writing $\backslash \wedge$ .  For example, <b><math>\wedge\#include</math></b> will only match strings which <i>begin</i> with the characters '#include'. When the caret is the first character of a character set it has a special meaning, see Sets of Characters above.
----------	--

	The dollar signifies the end of the string.
\$	For example <code>\d\s*\\$</code> will match strings which end with a digit optionally followed by whitespace. If you wish to match a literal \$ you must escape it by writing <code>\\$</code> .
\b	A word boundary.  For example the expression <code>\bOK\b</code> means match immediately after a word boundary (e.g. start of string or whitespace) the letter 'O' then the letter 'K' immediately before another word boundary (e.g. end of string or whitespace). Note that the assertion does not actually match any whitespace so if we write <code>(\bOK\b)</code> and we have a match it will only contain 'OK' even if the string is "It's <u>OK</u> now".
\B	A non-word boundary. This assertion is true wherever \b is false.  For example if we searched for <code>\Bon\B</code> in "Left on" the match would fail (space and end of string aren't non-word boundaries), but it would match in "tonne".
(?=T)	Positive lookahead. This assertion is true if the expression matches at this point in the expression.  For example, <code>const(?=\s+char)</code> matches 'const' whenever it is followed by 'char', as in 'static <u>const</u> char *'. Compare with <code>const\s+char</code> , which matches 'static <u>const</u> char *'.
(?!T)	Negative lookahead. This assertion is true if the expression does not match at this point in the expression.  For example, <code>const(?!\s+char)</code> matches 'const' except when it is followed by 'char'.

## Wildcard Matching

Wildcard matching can be performed using the special characters period (.) and asterisk (\*).

### Example:

The expression `.*` will match any text string that is not empty.

**NOTE:** When using the `.*` expression, make sure to limit the scope of your search as much as possible by picking specific objects to search on and specific locations (i.e. decks, rectangles, animation names, etc.). Otherwise the find results will be too broad.

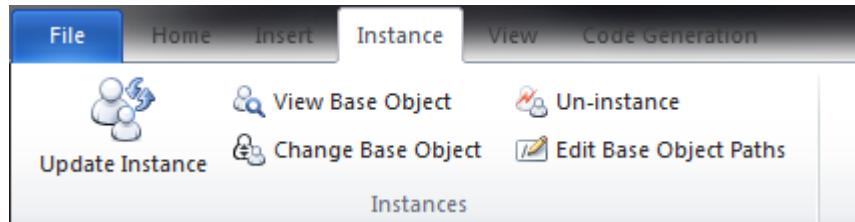
# Chapter 9: Classing

Altia's classing feature allows you to create designs efficiently by:

1. Permitting the editing of one master instead of repetitively editing copies of the same object, and
2. Allowing for collaborative objects to be placed into the design and updated at a later time.

The Classing feature is simply the ability to copy a regular object in one design file as an "Instance" in other design files. The regular object is now the "Base Object" for these Instances. The Instances have links back to the design file containing the Base Object and when the Base Object changes, you can update the Instances of the Base Object in other design files.

In other words, your objects in one design file can be "linked" to objects in other design files. Changes in a Base Object are automatically incorporated into Instances in a different design file by the use of the **Update Instance** command. For example, using Instances, you can change the background color of a button in a Base Object, and all Instances of the Base Object in another design file will automatically change to reflect the new background color by way of the **Update Instance** option from the Altia Design **Instance** ribbon.

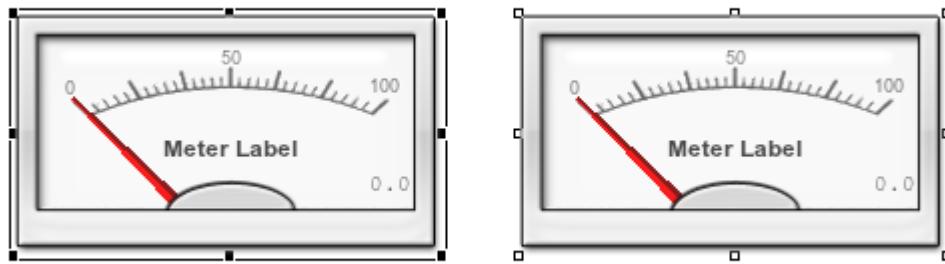


# 9.1 Classing

## 9.1.1 Instances

Instances are similar to normal graphic objects but with some important differences. The visual difference is the appearance of the handles when the object is selected.

The illustration below shows an Instance of a meter object on the left and a regular meter object on the right. You'll notice that the Instance's handles are solid and are connected by a line.



An Instance's description in the Graphics Editor's info pane is also different from a regular graphic object's description. The Instance in the previous illustration might have the following description "**Inst of id# 1038 in meters.dsn, id# 1**". This tells us what design file the Base Object is from (**Meters.dsn**) and the Base Object's ID number (1038) as well the Instance's object ID number (#1) within the current design.

As a special note, you cannot:

- Focus into an Instance.
- Add, delete, or redefine animations on an Instance from the Animation pane. These operations are disabled in the Animation pane when an Instance is selected.
- Add, delete, or redefine stimulus on an Instance from the Stimulus pane. These operations are disabled in the Stimulus pane when an Instance is selected.
- Add, delete, or redefine control logic on an Instance from the Control pane. These operations are disabled in the Control pane when an Instance is selected.
- Add, delete, or redefine properties on an Instance. The existing properties can be set from the Property pane and these property settings are preserved when a new version of a Base Object is imported. But, editing an existing property's definition, deleting an existing property, or adding a new property is disabled from the Property pane when an Instance is selected.
- Add, delete, or redefine connections on an Instance from the Connections pane for the Instance. Existing connections can be linked to other connections for other objects and these

links are preserved when a new version of a Base Object is imported. But, adding, deleting, or redefining the existing connections is disabled for an Instance.

- Rename animation/stimulus/control names from the Rename Animation dialog. You can still open the Rename Animation dialog from the **Home** ribbon's **Rename** button, or an object's right-click context menu. When the Rename Animation dialog opens, only the **Replace Prefix** and **Prepend** options are enabled if the selected object is an Instance. Adding a new prefix to existing animations with the **Prepend** option is available or replacing an existing prefix with the **Replace Prefix** option is also available. If a prefix is added or replaced, this change is preserved when the Instance updates to a new version of its Base Object assuming the Base Object's prefix has not been drastically manipulated from its previous version. If this happens, Altia may be unable to establish the relationship between the old and new Base Object prefix to preserve the added or replaced prefix on the Instance.

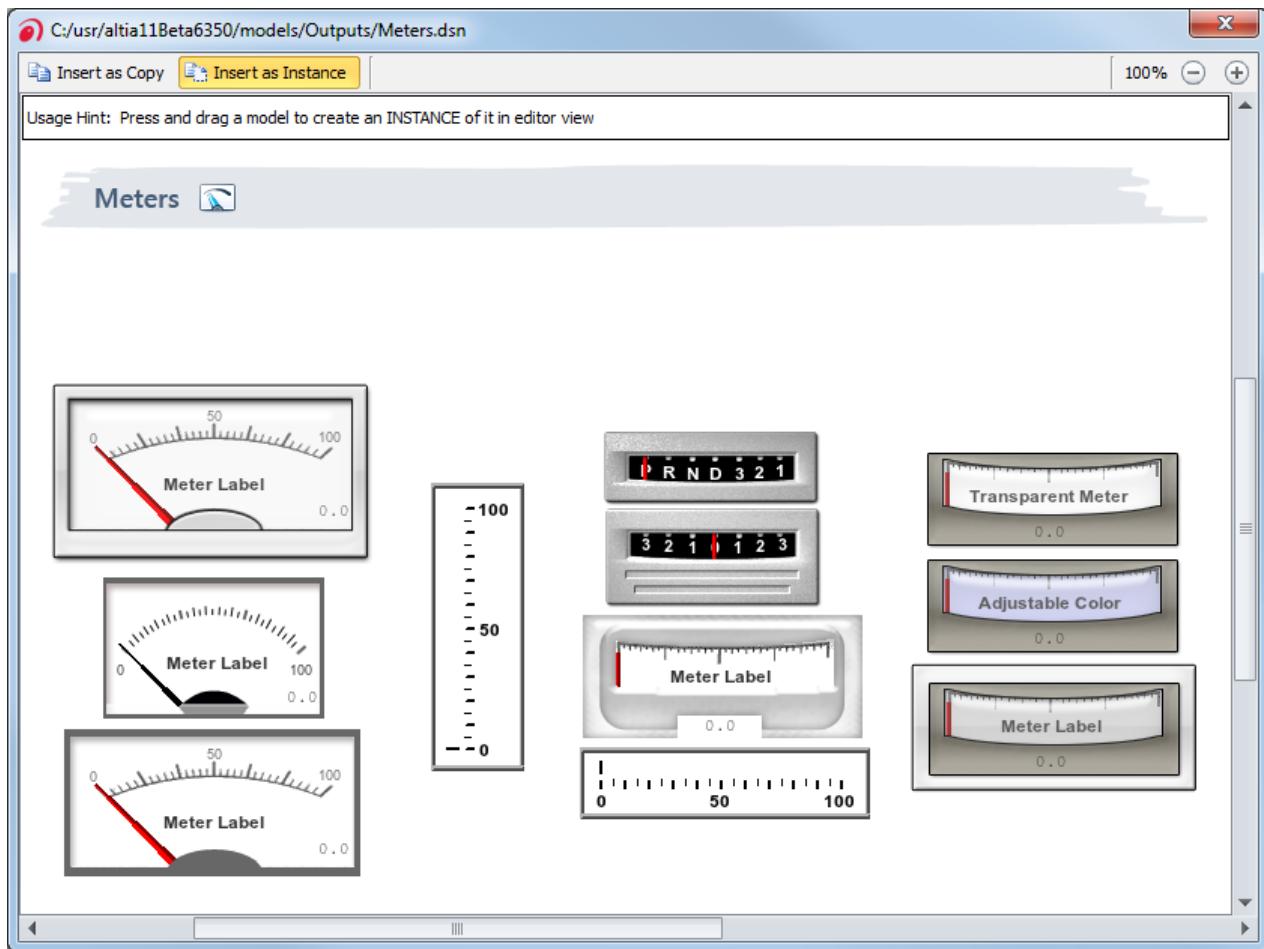
**NOTE:** A particular Base Object must maintain the same object ID number and reside in the same design file name for its entire existence as a Base Object because Instances refer to the Base Object by the name of the design file it resides in and its object ID number within that file. A Base Object should never be deleted from the design file containing it nor should it ever be cut and pasted. Instead, if changes are necessary, they must be made to the existing object. This typically requires focusing into the top-level object of the Base Object and making changes at this focus level or a deeper focus level, but never deleting the top-level object.

Instances can only be created from objects that are containers (such as a Group or Deck). You cannot make an Instance from a single line or a single rectangle, but you can make an Instance from a Group containing a single line or single rectangle. A Group is more flexible than a Deck (and can of course contain a Deck) and is therefore an excellent choice for the top-level object of a Base Object.

## 9.1.2 Create an Instance from a Base Object

Creating a copy of a Library object is no different than in previous versions of Altia Design: change to the **Insert** ribbon and select to open one of the standard libraries from the Model Library gallery or select the **Browse More Model Libraries...** menu option at the bottom of the Models Library gallery dropdown. This opens the chosen design file into an Altia Models Library window. As always, you can immediately drag the desired object into the Editor's work area to create a regular copy of it. This is the **Insert as Copy** operation. Now, however, an additional option is provided: **Insert as Instance**. Clicking on that button tells Altia to drag the object into the Editor's work area as an Instance instead of a regular copy. Altia automatically creates a link between that Instance and the Base Object of the design file in the Models Library window. If you want multiple Instances of that Base Object, simply drag and drop the Base Object multiple times with the **Insert as Instance** button selected.

Later, any changes done to the Base Object can be imported into your design (if you so choose) to update the Instances of the Base Object.



Let's say a design is opened in Altia Design that contains an Instance that refers to an older version of a Base Object. Then a Models Library window is opened that contains a newer version of the Base Object. If an **Insert as Instance** of the newer Base Object is performed, a dialog will appear asking if it is OK to update the existing Base Object and all of its Instances. This is because different Base Object versions cannot exist at the same time in the same design file. Choose the **Yes** button on the dialog to simultaneously complete the Instance copy and update the existing Instances to the newer Base Object. Choose the **Cancel** button to cancel the Instance copy and not update the existing Instances to the newer Base Object.

Instances actually reference a Base Object Local Copy that is a version of the Base Object from when it was last copied as an Instance from a Models Library window or from the most recent **Update Instance** command, whichever is newer. This copy is part of the design file that contains the Instances, but it is not visible in the design. In other words, it is part of the design data, but not in the drawable “universe” of the design data. Each Instance refers back to its Base Object Local Copy. If that Base Object Local Copy changes, all Instances of that Base Object change in that design.

Here are the typical steps for changing the Base Object Local Copy, and therefore all Instances of the Base Object, in a design file:

**Step 1:** The design file containing the Base Object is opened in Altia Design. Changes are made to the Base Object (remember, the Base Object should never be deleted or cut, only changed) and the design file is saved to the same design file name.

**Step 2:** A design file containing Instances of that Base Object is opened in Altia Design. At least one of the Instances of the Base Object is selected and the **Update Instance** option from the Altia Design **Instance** ribbon is chosen to update all Instances of the Base Object to its latest version.

In Step 2, the design file containing the Base Object is actually opened by Altia Design (it does not draw it, it just reads the file), the definition of the changed Base Object replaces the current Base Object Local Copy, and the Instances are updated using the new version of the Base Object Local Copy.

### 9.1.3 Instance Ribbon

The **Instance** ribbon provides the special operations that are allowed on Instances. These include:

- **Update Instance**
- **View Base Object**
- **Change Base Object**
- **Un-Instance**
- **Edit Base Object Paths**

### 9.1.4 Import Changes from Base Object

After a Base Object has been edited in its own design file and is ready for importing into a design file with Instances, open the design file containing Instances into Altia Design. Select at least one of the Instances of the Base Object or select an object containing an Instance or Instances. Click on the **Instance** ribbon, and choose **Update Instance**. This will update the Base Object Local Copy and all Instances to the most recent version of the Base Object saved to its design file.

If an internal change is required for an Instance, Altia recommends changing the Base Object and then import the changes so Instances update to the new Base Object. If the change should only apply to the exact Instance and not any other Instances and not the Base Object, an un-instance operation could be the solution. This operation “unlinks” the Instance from its Base Object and the Instance is just like any regular object. An un-instance operation is performed from the **Un-Instance** option in the Altia Design **Instance** ribbon.

If an Instance is transformed via a scale, rotate and/or distort operation, that transform on the Instance overrides any transform changes done to the Base Object. If the Instance has only been moved, then any transform (scale, rotate, and/or distort) changes done to the Base Class will be reflected in the Instance the next time Base Class changes are imported into the design file containing Instances

Any connections that are linked to/from an Instance from/to another object remain connected when a new version of its Base Object is imported. If connections are removed or added for the Base Object, the changes are reflected in all Instances when the new Base Object is imported. Even if an Instance’s connection has a link, the connection is removed if that connection has been removed for the Base

Object being imported. Similarly, if a new version of a Base Object has a connection added, the Instances will have this new connection when the new version of the Base Object is imported.

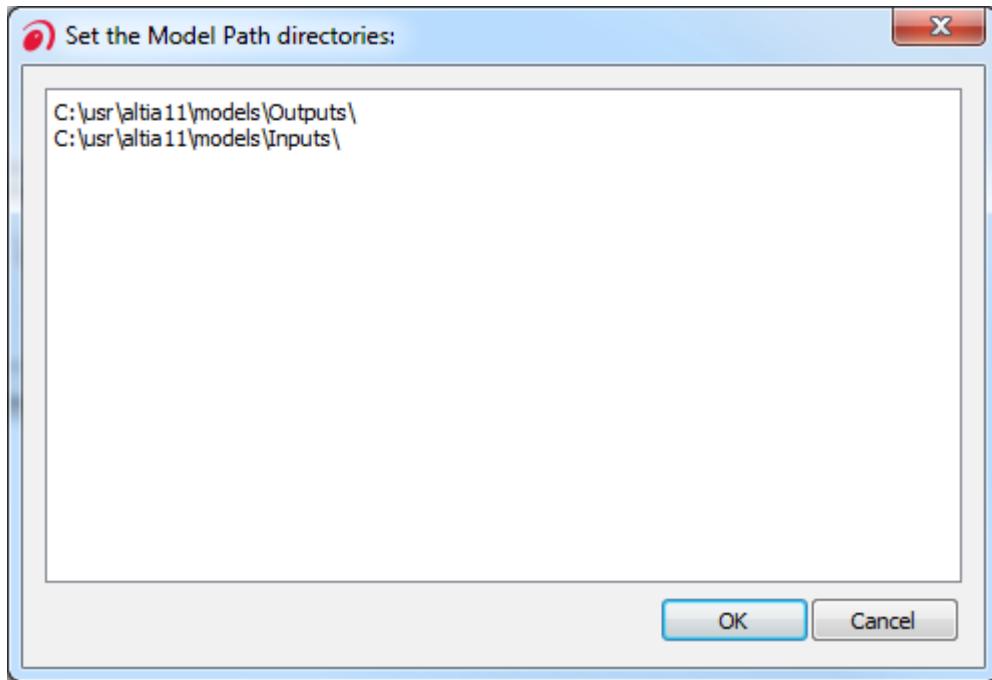
Any properties that have been set by the user on an Instance remain in effect even after a new version of the Instance's Base Object is imported. In other words, properties set by the user override Base Object property settings and properties never set by the user take their settings from the Base Object even after an import of a new version of the Base Object. If a new version of a Base Object has a property removed, that property is removed from Instances when the new version of the Base Object is imported even if the property was set by the user. Similarly, if a new version of a Base Object has a property added, the Instances will have this new property when the new version of the Base Object is imported.

Since focusing into an Instance is not enabled, be sure that any properties or connections for the Base Object are at its top level.

### **9.1.5 Edit Base Object Paths**

This command opens a dialog displaying a list of directories in an edit field, each on their own line. This type of list is referred to as a "path" in desktop computing environments. For this usage, it is called the "Model Path." The user can then edit the Model Path or just examine it. The Model Path is a list of directories where Base Objects can be found for a given Altia Design software installation. It is saved in a separate location from the design file and applies to all design files opened from that installation of Altia Design.

When an Instance is created, it refers to a Base Object Local Copy in the design file. This Base Object Local Copy holds the file name from which the original Base Object came as well as the object ID number of the Base Object in the file. Since design files can be on different machines and the location of these Base Object design files can vary on these different machines, fully qualified path names for finding these files are not used. This is indicated by the object identification shown in Altia Design when an Instance is the selected object. It reads something like "Inst of id# 190 in buttons.dsn, id# 11" which says it is an Instance of the object ID 190 in buttons.dsn (and the Instance itself has object ID 11 in the current design file). Notice there is no path given for the design file containing the Base Object, just the name of the design file. The design file name is used in conjunction with the Model Path to locate a Base Object's design file such as on a Base Object import operation. Whenever an Instance is dragged from a Models Library window into the Editor's work area, the directory of the Models Library window is added to the Model Path if that directory is not already in the Model Path.



The Model Path is the list of directories to look for design files containing Base Objects with each directory separated by a semi-colon (;). Order of the directories is important. Altia Design will search the directory list in order (left to right) looking for the first occurrence of a design file with the correct file name that contains the correct object ID. This allows the user to “overload” a particular Base Object design file with a different version by putting that directory containing the different version earlier in the Model Path. For example, if the user created an Instance of a Base Object from the standard Altia software installation Meters library and would like to change the Base Object without changing the original Altia Meters.dsn file, the user could make a copy of Meters.dsn, change the Base Object meter, and save the modified Meters.dsn file using the same name, but in a new directory. The user could then add that new directory to the Model Path and place it before the Altia software installation models directory. When an **Update Instance** is performed in the future, it would find the new Base Object first and update Instances with that new Base Object. Notice that the user should work from a copy of the original Meters.dsn and modify the exact meter with the required object ID number to create the new Base Object because the object ID number must remain the same. Otherwise, the new Base Class Object is not found during an import of changes.

## 9.1.6 View Base Object

This command will open a Models Library window for the design file containing the Base Object and selects that object in the Models window. If the Base Object cannot be found because the object was deleted, or the design file containing the Base Object is not in the Model Path, an error dialog is shown.

This feature is helpful if multiple users are editing and/or using the same Base Object and you are not sure if you wish to import changes from the Base Object. Altia recommends viewing the changed Base Object prior to updating to it.

## 9.1.7 Un-Instance

Selecting this option will take the currently selected Instance and turn it back into a regular object no longer tied to a Base Object.

## 9.1.8 Collaboration Using Classing

Classing permits parallel or collaborative development of a superset model by following these steps:

1. Create a placeholder object (hereafter referred to as “Placeholder”) in the Editor and save it within your Altia “Library” file.
2. Start a design within the Editor in normal fashion (hereafter referred to as the “Collaborative Design”)
3. Open the Library file in a Models Library window and drag the Placeholder into the Collaborative Design as an Instance.
4. Send the Library File to your collaborative partner and have him/her edit the Placeholder object as required.
  - a. If networked, make your Library folder/files “Shared” within Windows, and send the collaborative partner the location of the Library File.
5. Upon completion of the edited Placeholder, make sure the revised file overwrites the old file in the original file location.
6. Within the Collaborative Design, perform an **Update Instance** from the Instance ribbon and the work completed in parallel may be easily inserted into the overall design.

## 9.1.9 Key Terms

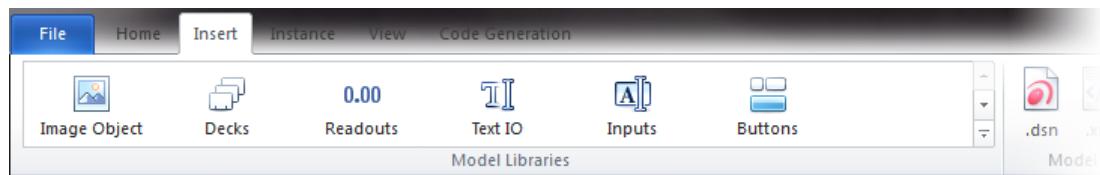
- Base Object - The Altia object that is referenced by an Instance.
- Base Object Local Copy - The version of the Base Object that is stored in the design (.dsn) file with Instances. This copy allows designs to remain complete and functional even when the .dsn file containing the Base Object cannot be found.
- Group - An Altia container object that can be focused into (an Altia object that contains other objects) such as a Group or Deck.
- Instance - An Altia object that is a copy of a Base Object.
- Object - A generic and very general term for a graphical construct in Altia. In this document, the term “object” does not refer to the programming- or class-related usage of the term. So, Groups are objects (and contain objects) - which also means that Base Objects and Instances are objects.

# Chapter 10: Models Libraries

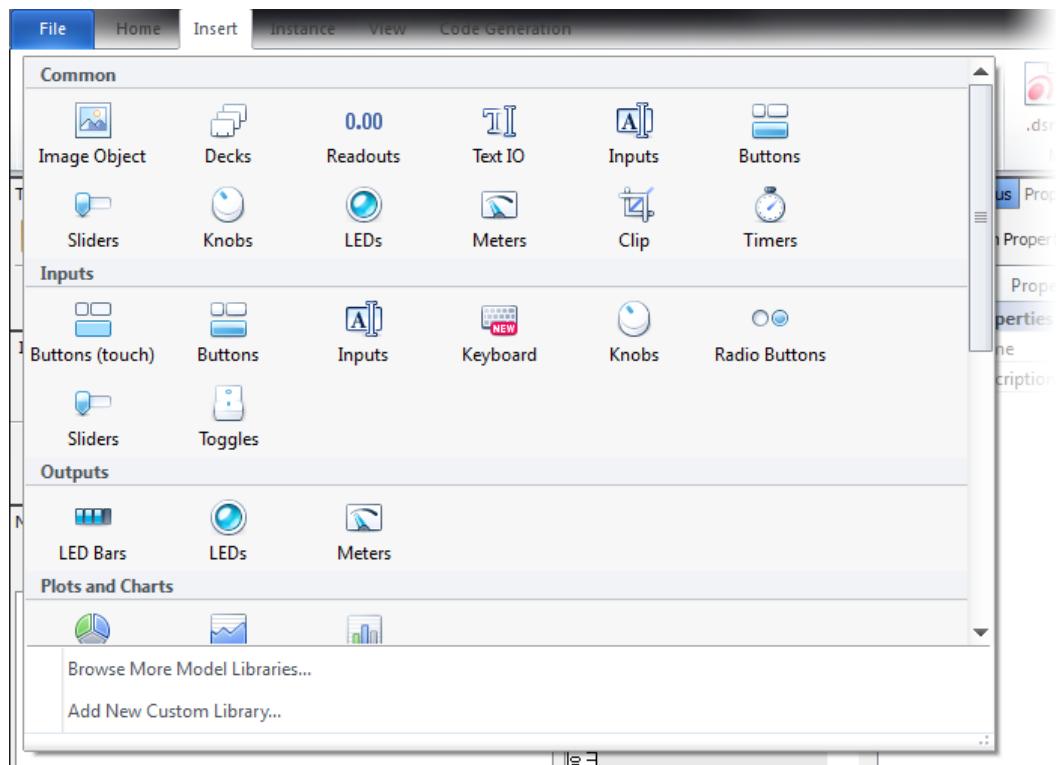
Altia design files are often used to maintain libraries of component models. These libraries allow efficient standardization on and reuse of certain models for developers of Altia designs.

To save you time in creating designs, Altia Design provides numerous Models Libraries containing commonly used components (buttons, on/off switches, meters, etc.) and special objects that expand Altia Design's capabilities (sound objects, a deck object, etc.).

When a component model is added to your design, any animation, stimulus, or control attached to that model will be added as well. You can also add your own custom Altia objects to a Models Library.



To access a Models Library, click the **Insert** ribbon. The Model Libraries gallery allows selection of a library to open. If the library you wish to open is not visible within the Insert Ribbon. Click the Models Library gallery dropdown arrow to view all of the available Model Libraries.



For more information on using the Model Library gallery, see [Chapter 1, Section 1.3.1](#).

## 10.1 Opening a Models Library

To open a Model Library, click on the desired library in the Insert ribbon's Model Library gallery, or from the Model Library Gallery dropdown. You may also open a library by selecting **Browse More Model Libraries...** from the Model Library gallery dropdown to use a standard Windows file open dialog to select a library saved anywhere on your computer.

**NOTE:** When using **Browse More Model Libraries...** from the Model Library gallery dropdown, be sure to choose a file with a .dsn extension, as files with other extensions are not library (design) files.

The following models appear in the Model Library gallery dropdown's **Inputs** category. These models reside in the standard models / Inputs directory, \$ALTIHOME/models/Inputs.

**Table 10-1 Model Libraries in the Inputs category**

NAME	DESCRIPTION
<b>Buttons (touch)</b>	Button objects of varying behaviors using easily configurable Image Objects for button up and down state graphics.
<b>Buttons</b>	Push buttons of various shapes and sizes.
<b>Inputs</b>	Assorted text and numerical input fields.
<b>Keyboard</b>	Virtual keyboard object
<b>Knobs</b>	Rotating input knobs of various types and sizes.
<b>Radio Buttons</b>	Push buttons of various shapes and sizes.
<b>Sliders</b>	Many styles of slider bars.
<b>Toggles</b>	A multitude of 2- and 3-position toggle switches.

The following models appear in the Model Library gallery dropdown's **Outputs** category. These models reside in the standard models /Outputs directory, \$ALTIAHOME/models/Outputs.

**Table 10-2 Model Libraries in the Outputs category**

NAME	DESCRIPTION
<b>LED Bars</b>	Several sizes of 10-LED bars.
<b>LEDs</b>	Different sized single indicator lights.
<b>Meters</b>	Numerous analog display meters.

The following models appear in the Model Library gallery dropdown's **Plots and Charts** category. These models reside in the standard models /Plots and Charts directory, \$ALTIAHOME/models/Plots and Charts.

**Table 10-3 Model Libraries in the Plots and Charts category**

NAME	DESCRIPTION
<b>Multi Plot</b>	Advanced object supporting plots of many kinds including X-Y, Stripcharts, and Multi-Line.
<b>Pie</b>	Several instances of the pie chart object in different configurations.
<b>Plot</b>	Instances of simple line plots, filled plots, and strip charts which can dynamically plot data. Data is typically provided by an application program.
<b>Stripchart</b>	Flexible strip chart that supports 1 to 8 simultaneous plots.

The following models appear in the Model Library gallery dropdown's **Purchased** category. These models reside in the standard models / Purchased directory, \$ALTIHOME/models/Purchased.

**Table 10-4 Model Libraries in the Purchased category**

NAME	DESCRIPTION
<b>3D Scene Object</b>	Displays 3D assets and animates them.
<b>Drawing Area Object</b>	A canvas that lets the developer draws on using a programmable API.

The following models appear in the Model Library gallery dropdown's **Special Functions** category. These models reside in the standard models / Special Functions directory, \$ALTIHOME/models/Special Functions.

**Table 10-5 Model Libraries in the Special Functions category**

NAME	DESCRIPTION
<b>Blur Filter</b>	Container that can apply a Gaussian blur algorithm to its raster contents.
<b>Clip</b>	A clip object is a group object which shows its contents (another object) "clipped", or trimmed, to the clip object's extent, like the viewfinder of a camera. The view displayed by the clip object can be "panned" around its contents like a movie camera.
<b>Coverflow Horiz</b>	Horizontal scrollable image based menu.
<b>Coverflow Vert</b>	Vertical scrollable image based menu.
<b>Decks</b>	A deck object is a group object which permits stacking objects on top of one another, like a deck of cards. Any card can be "flipped" to the top, making it visible and hiding all others.
<b>Display Object</b>	A container used to manage hardware display layers.
<b>Image Object</b>	An object used to display external image files.

<b>Language</b>	Allows quick multi-object language switching to support localization.
<b>Layer Manager</b>	A Layer Manager Object is used to create a multi-layer HMI for use on an embedded target with one or more multi-layer display controllers. Layers are created in the Layer Manager and can be assigned to Layer Groups in order to compose the presentation of the HMI (similar in concept to creating screens).
<b>Mask Object</b>	Create image and object-based masking effects.
<b>Shapes</b>	Various geometric shapes.
<b>Skin</b>	Re-skin your GUI graphics without affecting code.
<b>Snapshot</b>	Capture an image of other objects or parts of the design.
<b>Sounds</b>	Sound object configured to play various .au files on UNIX or .wav files on the PC.
<b>Tickmarks</b>	Draw tickmarks in a number of different configurations.
<b>Timers</b>	Timer and counter components.
<b>Views</b>	A container that displays its contents in a separate window.

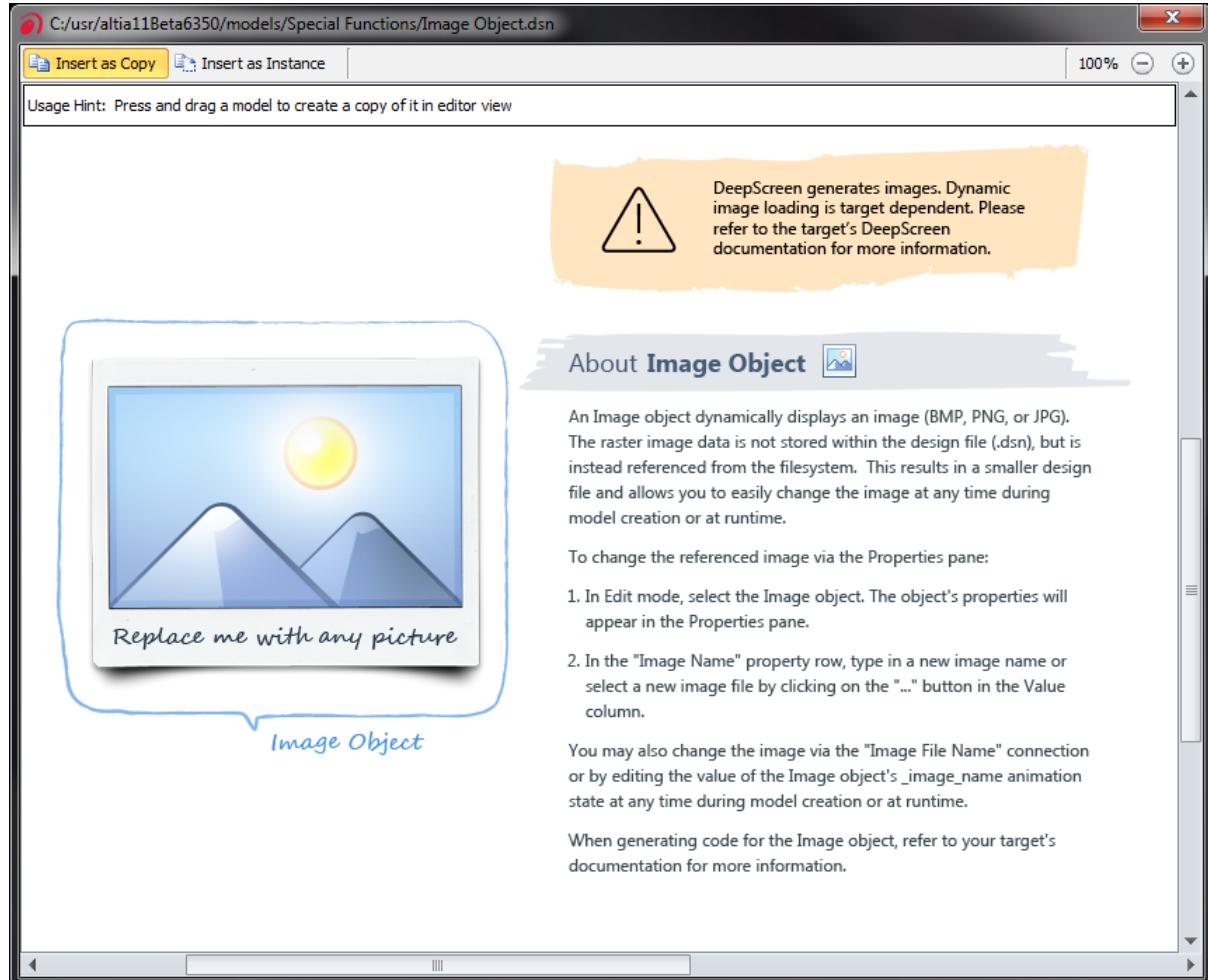
The following models appear in the Model Library gallery dropdown's **Text** category. These models reside in the standard models /Text directory, \$ALTIAHOME/models/Text.

**Table 10-6 Model Libraries in the Text category**

NAME	DESCRIPTION
<b>Multi-Line</b>	Text objects that can display multiple lines of text.
<b>Readouts</b>	Simple number and text readout components.
<b>Text IO</b>	Text I/O objects are variants of standard text objects; however, the text they display can be updated dynamically by control or an application program. Text I/O objects are also used to create text input areas that accept keyboard events. The text that is entered can be accessed by control or application code.

## 10.2 Models View

A Models View gives you a limited access view into any Altia design file of your choosing. A design open in the Graphics Editor will not be affected by a Models View and more than one Models View can be open. An object can be copied from a Models View into the Altia Editor.



**NOTE:** You may open any Altia Design file into a Models view. In other words, any design you create can serve as a Models Library.

## 10.2.1 Insert As Copy



When the **Insert As Copy** option is selected (default), copies of objects will be created when dragging from the Models View to the graphic editor. Simply press the left mouse button down on an object in the Models View and, while still holding the button down, drag the object into any Graphics Editor View, position it, and release the mouse button.

You may also drag in multiple components at once. Select the desired objects by clicking in empty space in the Models View and, while holding down the mouse button, dragging the selection box over the desired components. Click on one of the selected objects and drag into the Graphics Editor to copy over all the selected objects.

If one or more components at a time are copied from a Models View, any linked connections between the objects are automatically preserved. That is to say, linked connections are preserved between objects in the copy set.

If the chosen model has any animation, stimulus, or control assigned to it, Altia Design will automatically add a numerical prefix to these names to make them unique for your design. If the default Auto-Rename feature is turned off (via the application defaults file), a dialog will appear letting you manually rename the animation, stimulus, and control. For more information on renaming, please refer to

[Chapter 4: Stimulus Editor](#).

## 10.2.2 Insert As Instance



When the **Insert As Instance** option is selected, objects dragged from the Models View to the graphic editor will be "instances" of the object. The Instances have links back to the design file containing the Base Object and when the Base Object changes, you can update the Instances of the Base Object in other design files.

For more information on Classing and Instances, see [Chapter 9: Classing](#).

## 10.3 Using Properties

The components in the Standard Model Libraries have properties that allow you to easily change their appearance or behavior.

### 10.3.1 Opening the Properties Pane

If the Properties pane (Figure 10.1) is not currently open, it can be turned on from the **View** ribbon's **Windows** dropdown (Figure 10.2).

Figure 10.1: Properties pane

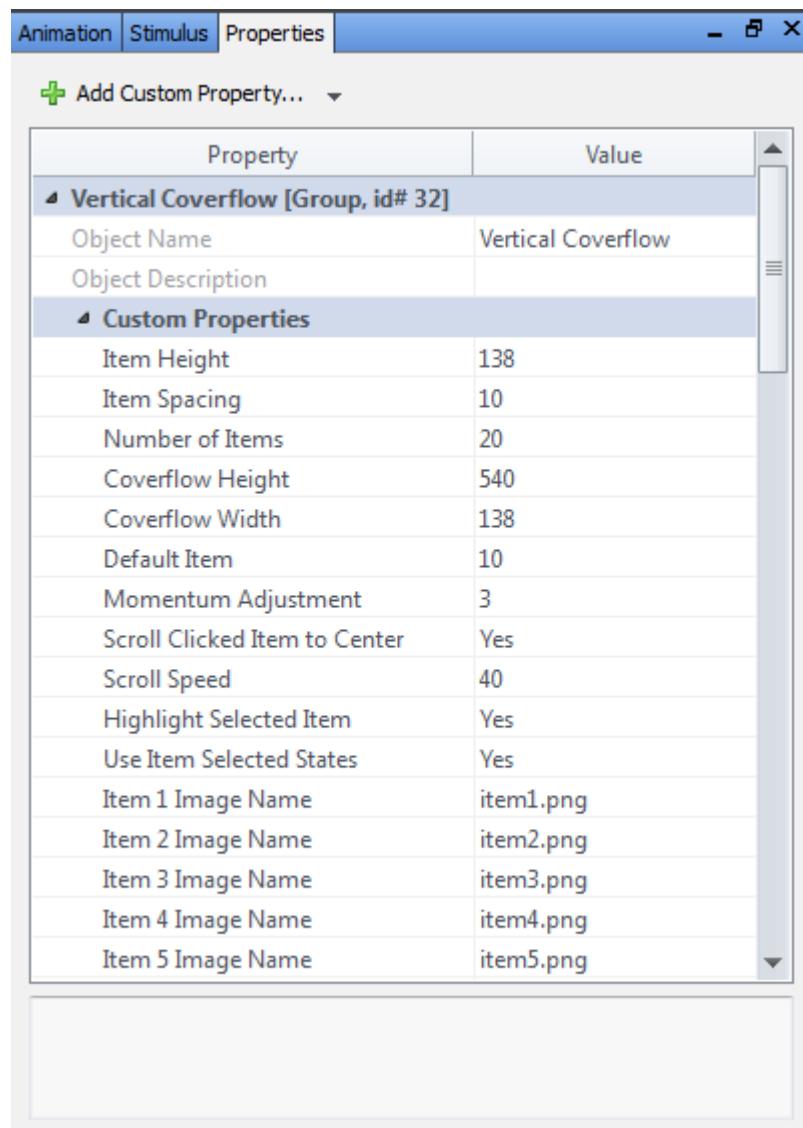
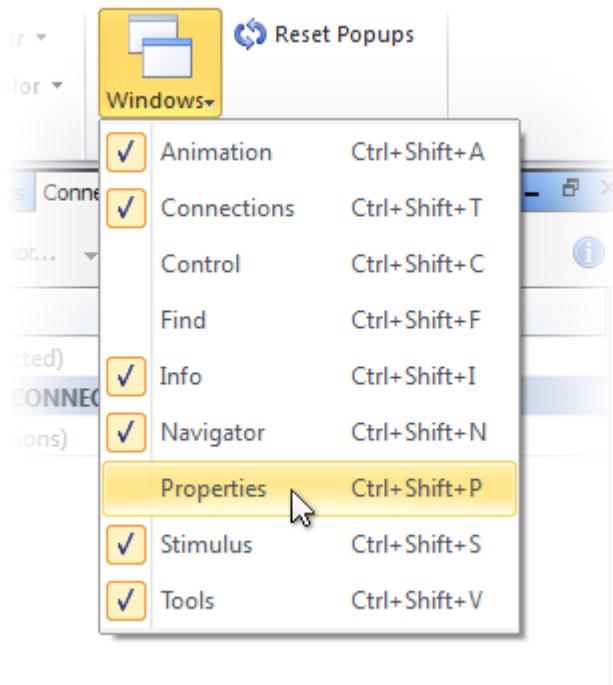


Figure 10.2: View ribbon's Windows dropdown



### 10.3.2 Setting Property Values

Object properties are set by entering values in each of the fields listed for the object. Each field on the property dialog can have data entered in it either as a value, a string, or an option from a drop-down list.

You may use the Tab key to quickly move between the fields in the dialog. As each field is highlighted, a descriptive message appears in the top of the dialog.

#### The “Object Name” Property

Objects with a property titled **Object Name** have an identifying text string name associated with them. This text is used to distinguish objects in the Property Dialog or in the Connections Dialog. It is also used during HTML export. See [Chapter 1, Section 1.1.3, XML/Documentation](#) for more information.

To name an object, simply go to the **Object Name** field in the object’s Property pane and type in a new text string. For example, perhaps changing the **Object Name** property from **Slider** to **Gain Feed** would be more meaningful in your design. If you have a number of similarly appearing objects in your model, naming them can be indispensable.

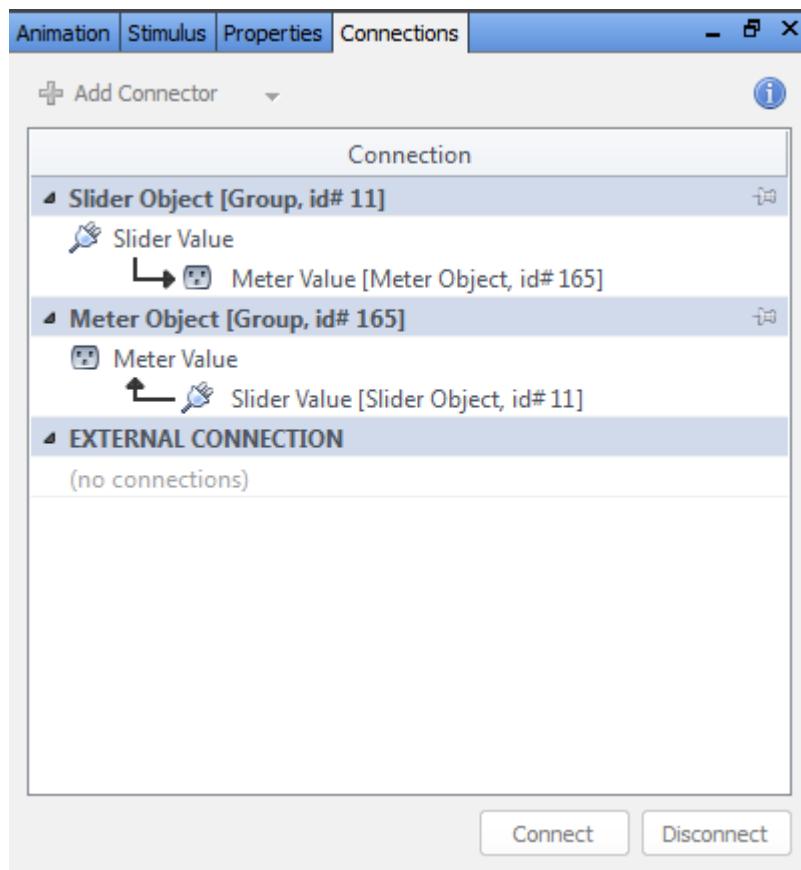
### 10.3.3 Adding and Modifying Properties

For more information on adding and modifying properties using the Properties Editor, please see [Chapter 5: Properties Editor](#).

## 10.4 Using Connections

Connections are a quick and easy way for Altia object's to communicate with each other or with the outside world (simulation models and external programs). The Connections dialog is used to view the defined input and output connections for an object and change the link(s) associated with connections.

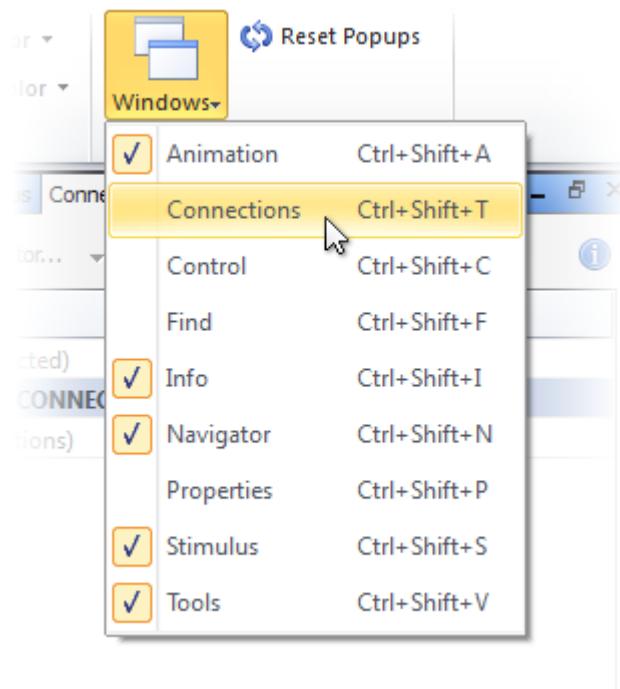
Figure 10.3: The Connections pane



### 10.4.1 Opening the Connections Pane

If the Connections pane (Figure 10.3) is not currently open, it can be turned on from the **View** ribbon's **Windows** dropdown (Figure 10.4). For more information on creating new connections and making connections between objects, see [Chapter 7: Connections](#).

Figure 10.4: View ribbon's Windows dropdown



## 10.4.2 Linking Two Objects Using Connections

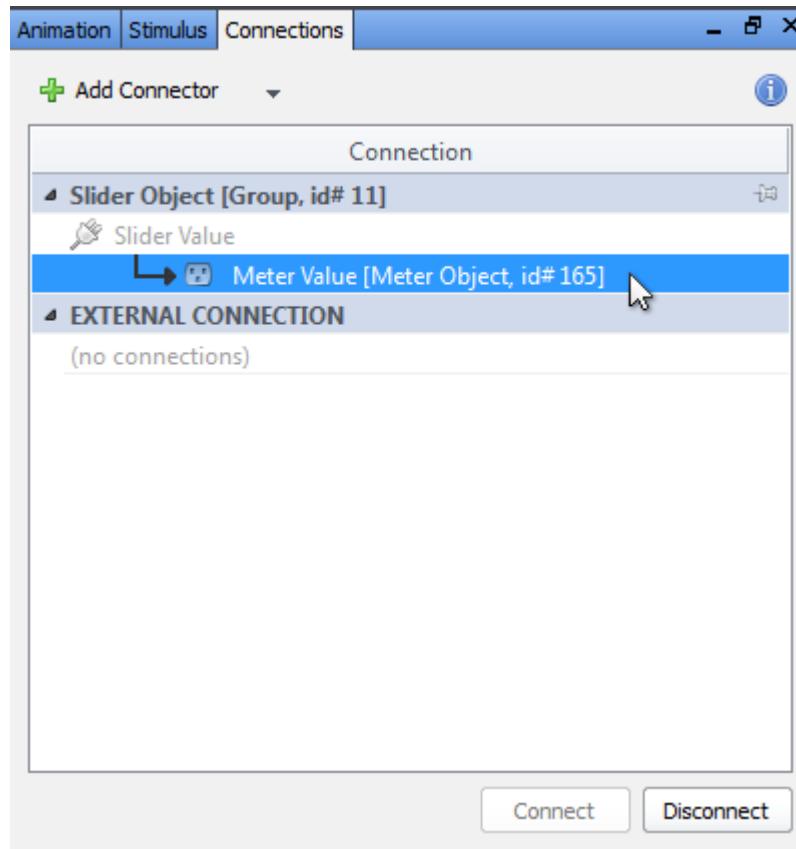
To connect two objects together, perform the following steps:

1. Select the first object you want to connect.
2. Shift-select the second object to which you want to connect. Both objects should now be selected.
3. The two selected objects' connections are shown in the **Connections** pane.
4. Select an input or output from each object that you want to connect. Note that an input connector can only be connected to an output connector and vice versa.
5. Press the **Connect** button in the Connections pane and the objects will be connected.

### 10.4.3 Unlinking a Connection Between Two Objects

To disconnect two objects, do the following:

1. Select either of the objects you want to disconnect.
2. In the Connections pane, select the "link" that you want to break (links are denoted by <- or ->). Be sure to select the link and not the connector (as shown below).



3. Press the **Disconnect** button. This will disconnect the two objects and remove the links.

### 10.4.4 Showing the Objects Associated With a Link

To see the object to which a link is referring, just select the link and right-click to display the context menu. Choose **Show Connected Object** from the context menu. This will select the object that is connected by that link.

## 10.4.5 Adding and Modifying Connections

For more information on adding and modifying connections using the Connections Editor, please see [Chapter 7: Connections](#).

## 10.5 Advanced Object Models

In addition to standard Models Libraries containing buttons, knobs, dials, etcetera, Altia Design includes Models Libraries that contain advanced objects. This section details how such libraries are used. The following libraries are discussed:

**Blur** (Special Functions/Blur.dsn)

**Clip** (Special Functions/Clip.dsn)

**Decks** (Special Functions/Decks.dsn)

**Image Object** (Special Functions/Image Object.dsn)

**Language** (Special Functions/Language.dsn)

**Layer Manager** (Special Functions/Layer Manager.dsn)

**Mask Object** (Special Functions/Mask Object.dsn)

**Multi-Line** (Text/Multi-Line.dsn)

**Multi Plot** (Plots and Charts/Multi Plot.dsn)

**Plot** (Plots and Charts/Plot.dsn)

**Pie** (Plots and Charts/Pie.dsn)

**Skin** (Special Functions/Skin.dsn)

**Snapshot** (Special Functions/Snapshot.dsn)

**Sounds** (Special Functions/Sounds.dsn)

### Text Input/Output Objects

- **Inputs** [Inputs/Inputs.dsn]
- **Readouts** [Text/Readouts.dsn]
- **Text IO** [Text/Text IO.dsn]

**3D Scene Object** (Purchased/3D Scene.dsn)

## 10.5.1 Blur Filter

Special Functions/Blur.dsn

The blur filter is a container that can apply a Gaussian blur algorithm to its raster contents. This widely used graphic effect reduces the detail of an image resulting in an effect similar to that produced by an out-of-focus lens.

To add the Blur Filter to a design, simply open the BlurFilter.dsn file and drag the Blur Filter container into the design's work area.

### Built-in Animations for the Blur Filter

Property	Animation	Description
<b>Standard Deviation for X-axis</b>	<b>stdDevX</b>	A floating point value specifying the standard deviation of the Gaussian distribution in the X direction. Valid values are from 0.01 to 10.0. The bigger the number, the more blurring occurs. Invalid values will be ignored.
<b>Standard Deviation for Y-axis</b>	<b>stdDevY</b>	A floating point value specifying the standard deviation of the Gaussian distribution in the Y direction. Valid values are from 0.01 to 10.0. The bigger the number, the more blurring occurs. Invalid values will be ignored.
<b>Link Standard Deviation Value Changes</b>	<b>linkStdDev</b>	A toggle that links the values of both standard deviation values on the X and Y axes. If 0, the stdDevX and stdDevY values are independent. If not 0, a change in one value will also change the other value.

In addition, the following should be considered:

- Code generation is limited to OpenVG targets.
- The PowerVG Windows emulation is known to have performance issues.

### Adjusting the Blur

With the Blur Filter, it is possible to set the standard deviation of the X and Y direction using a value of 0.01 to 10.0. You can also add a toggle to link the two values, which allows you to blur the contents in both directions simultaneously or unlink them and blur the image in either direction independently. Supported contents include imported Raster objects, externally referenced Image objects, and Snapshot objects.

## 10.5.2 Clip Object Model

Special Functions/Clip.dsn

The file Clip.dsn contains a model of a clip object, which is a special group object. Like a camera view finder, the clip object allows you to see a select portion of the larger picture. Any objects outside of the viewing area are “clipped away”—invisible to the viewer, but still present. The clip object is useful for building one-dimensional or two-dimensional scroll lists or view areas for applications such as map displays, aerospace components, and video camera simulations.

To add a clip to your design, click on the model and drag it into your design’s work area. The object’s animation names will automatically be uniquely renamed so that you may have various clip objects that do not conflict with one another.

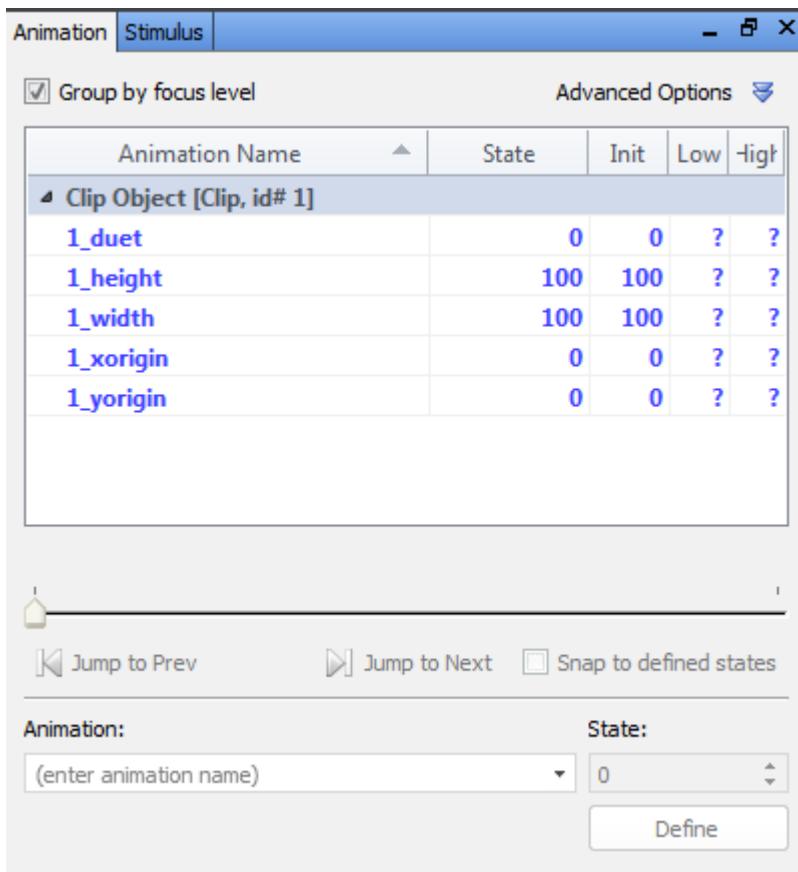
Add objects to a clip, delete or modify them just as you would in a regular group. With the Graphics Editor in **Edit** mode, select the clip object, then click on **Focus In**. Focus into the clip object group to add, delete, or modify the objects it contains. When you are done editing, click on **Focus Out** to step out of the group. You can also copy and paste existing objects or drag models from a Models View into the clip object group while you are focused in on the clip.

Because the clip is derived from a standard group object, animation, stimulus, and control that affects the clip will also affect the objects in the clip. For example, if the clip is stretched, objects in the clip will be stretched. In the same manner, any color, fill, pattern, outline or font set for a clip object will be forced onto all of the objects within it. To clear any of these attributes, select the clip and choose **Clear Group Style** from the Graphics Editor’s **Home** pane.

**NOTE:** If a clip is rotated, the objects it contains will rotate, and the behavior of width, height, xorigin, and yorigin will change. The clip itself, however, is forced to maintain a rectangular clip region, and its sides will remain exactly aligned with the vertical and horizontal axes.

### Adjusting the Size of the Clip Area

To adjust the size of the clip area without changing the size of any objects contained within it, you must adjust the clip object’s **height** and **width** animations. With the clip object selected, you will see the following animations in the Animation pane:



To change the width or height of the clip area, choose **width** or **height** from the Animation List Area and change its state value. You can also use a client program, control block, or stimulus to dynamically change the state values of **width** or **height**.

The anchor point for the size is the lower left corner of the object so an increase in **width** will grow the clip area to the right and it will grow taller with an increase in the state of **height**.

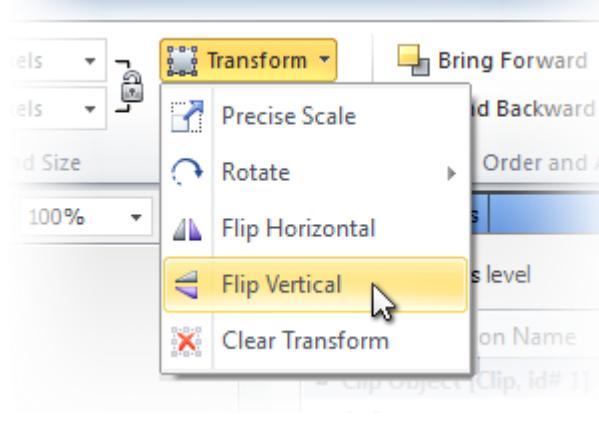
## Duet Mode

When the animation named **duet** is set to state 0 (the default mode), **width** can be set independently of **height**, and **xorigin** can be set independently of **yorigin**. When **duet** is set to state 1, each **width** state change must be accompanied by a **height** state change before the clip object is visually modified. Similarly, each **xorigin** change must be accompanied by a **yorigin** change. Duet mode allows for smooth two-dimensional size adjustment and panning.

## Adjusting the Origin of the Clip's View

By default, the origin of the clip's view is the lowest and left-most point of any objects within the clip. You can change this origin by changing the animation state values of **xorigin** and **yorigin**. For example, if the state values for **xorigin** and **yorigin** are both changed to 10, the objects in the clip would appear to move to the left and down. That is, the clip's origin moves 10 pixels up and 10 pixels to the right.

Since the clip's origin is anchored in the lower left corner, increasing **yorigin** state values causes objects in the clip to scroll downward. You can easily change this behavior by flipping the clip object vertically. Select **Flip Vertical** from the Home ribbon's Transform dropdown menu. By doing this, you can make objects in the clip scroll upward when the state value of **yorigin** is increased, which is useful in creating a text scroll list.



**NOTE:** Before performing the **Flip Vertical**, you should make copies of all objects in the clip so that they can be pasted back into the clip after scaling. Otherwise, the objects in the clip will be flipped over.

Due to windowing system and performance considerations, a clip is forced to maintain a rectangular clip region. If a clip object is rotated, the objects it contains will rotate and the behavior of the **width**, **height**, **xorigin**, and **yorigin** animations will change as a result of the transformation. However, the clip region will remain rectangular with its sides exactly aligned with the vertical and horizontal axes.

## The Clip's Effect on Animation and Stimulus

When an object in a clip is hidden, any animations it may have are updated as states arrive. If the object comes into view at a later time, it will be properly re-drawn. Any portion of a mouse/keyboard input stimulus area is disabled when hidden. If a portion of a **Leave** stimulus is hidden, the clip object's border, where it intersects the area, becomes the new stimulus area border. This is also true for an **Enter** stimulus.

Timer stimulus or control on hidden objects will behave normally. However, if mouse or keyboard input stimulus triggers the timer stimulus or control and the input stimulus is hidden, the timer or control block will not be triggered.

### 10.5.3 Deck Object Model

Special Functions/Decks.dsn

The file deck.dsn contains a deck, which is a special group object that behaves much like a deck of cards. Using a deck, you can control the display of many subgroups of objects (referred to as “cards”) one set at a time. Simply by changing the state value of the animation named **card**, you can hide the current card (subgroup of objects) and display another card. A deck is ideally suited for controlling multiple screen applications, hierarchical soft-key style menu systems, or dynamic presentations. The number of cards you can have in a deck is limitless, as is the number of decks you can have in a design. Individual cards in a deck can also contain their own decks.

To add a deck to your design, click on the deck model and drag it into your design’s work area.

A deck is derived from a standard group object and, therefore, inherits all capabilities of a group. For example, if you stretch or rotate a deck, the objects in it will be stretched or rotated. The same is true for the color, brush, pattern, and font attributes. To clear any such attribute setting, select **Clear Group Attributes** from the Graphics Editor’s **Object** menu.

#### Adding Objects to the Cards of a Deck

1. With the Graphics Editor in **Edit** mode, select a deck object.
2. Open the Animation Editor and highlight **card** in the Animation List Area. This will place **card** in the **Animation** field of the Animation Editor. Choose a card by choosing a state value. The first card of the deck has a state value of **0**, the second a state value of **1**, etc. Keep in mind that the card with state value **0** is the default card and will appear when the design file is loaded.

**NOTE:** **To make a card other than 0 the default card, select the deck object, set the card animation to the desired initial state (the number of your default card), and choose Set Selected Init State to Current State from the Animation Editor’s right-click context menu.**

3. Add objects to the card you have chosen by selecting **Focus In** in the Graphics Editor. If you wish to add an existing object, select the object and press **Ctrl+C** (copy) before focusing in on the card. Paste the object into the card after you have focused in on it. To edit existing objects in a card, **Focus In** on the card and make appropriate changes to the objects within it.
4. When you are done adding objects to a card or editing existing objects, select **Focus Out** and choose your next card via the **card** animation’s state value.

Only the card currently in view (at the top of the deck) can receive input stimulus. Cards that are hidden, however, continue to receive animation state changes if they have any assigned animation. If a card receives state changes while hidden, it will be correctly updated when it is displayed again. In addition, timer stimulus or control can be activated in a hidden card. Timer stimulus will continue to be active until it receives a stop event.

There is no limit to the number of cards you can have in a deck or the number of decks you can have in a design. Also, cards of a deck can contain additional decks themselves which makes it possible to design very interesting hierarchical display scenarios.

## The Details of a Card Animation State Change

When the state of a deck's card animation is changed, all objects in the current card are removed from the graphics "universe." They are still part of the design, but they cannot redraw and they ignore all input stimulus. They will, however, continue to receive animation state change events if they have animation. When they are displayed again, they will be correctly updated based on any state changes which occurred while they were "hidden." If any objects in a hidden card have timer stimulus, it *can* be activated while the card is hidden. If it is active at the time the card is hidden, it will remain active until it receives a stop event.

## Characteristics of a Deck

A deck is derived from a standard group object so it inherits all capabilities of a group. For example, if the deck is stretched or rotated, the objects in the deck will also stretch or rotate. The deck can be given user-defined animation in addition to its existing card animation. For example, an animation called **move** could be assigned to the deck to move it around the screen.

If the foreground color, background color, pattern, outline, or font is set for a deck object, the attribute will be forced onto all of the objects in the deck just as would happen with a normal group object. To clear such an attribute setting, choose **Clear Group Style** from the **Home** ribbon while the deck is selected.

## Setting the Initial State of a Deck

By default, the card animation for a deck is set to a 0 value when a design is initially loaded. To set the initial state to something other than 0, select the deck object, set the card animation to the desired initial state using the Animation Editor, and then choose **Set Selected Init State to Current** from the Animation Editor's right-click context menu.

## Typical Deck Uses

In run mode, it is customary to control a deck's card animation from input stimulus, timer stimulus, or an application program. A deck is ideally suited for controlling multiple screen applications, bitmap animation sequences, hierarchical soft-key style menuing systems, or dynamic presentations.

## 10.5.4 Image Object

### Special Functions/Image Object.dsn

The image object allows a design to dynamically display images from files (instead of importing a **Picture** from the **Insert** ribbon which stores the raster image data directly in the .dsn). There are several advantages to loading an image dynamically. First, it reduces the size of the design which improves start-up time. In addition, the image object allows you to change the displayed image dynamically (using control statements or a client application) by changing the name of the image file associated with the image object or by changing the image file contents and requesting a new read of the file by the object. To add an image object to a design, select the Image Object library from the Insert ribbon's Model Library gallery dropdown, to open a Models View, then drag the image object into the editor's drawing area. Users developing for multiple platforms should note that the image object reads PNG (.png), JPEG (.jpg), and BMP (.bmp) on Windows or X Window Dump (.xwd) files on UNIX.

Sending a string to the image object's **image\_name** animation sets the current image file. If you leave off an extension for the name, Altia assumes .bmp on a PC and .xwd on UNIX. If a file is not found and the environment variable **ALTIAIMAGE** is set, Altia checks in the directory given by **ALTIAIMAGE**. If that fails, Altia searches /usr/altia/image.

## 10.5.5 Language Object

### Special Functions/Language.dsn

The file language.dsn contains a model of a Language object, which is a special object that allows users to change text strings for portions of a design at runtime using a language configuration file.

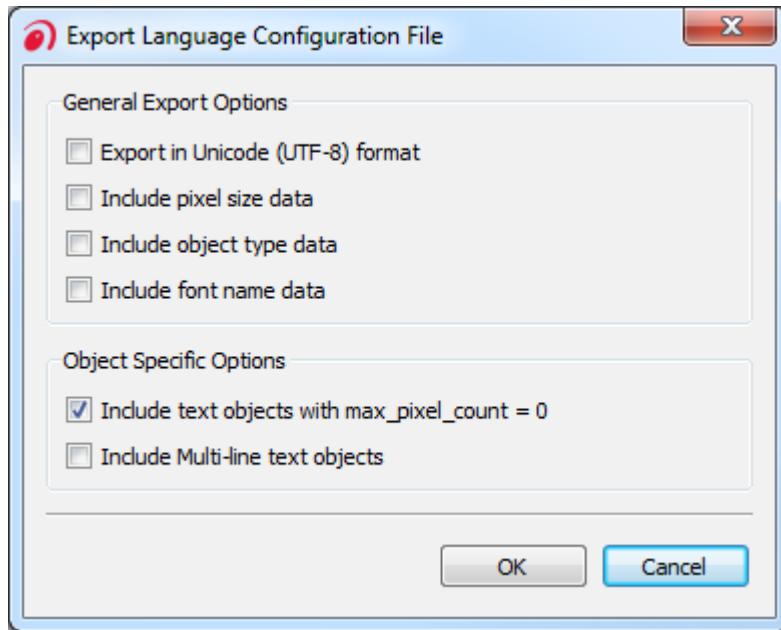
Text objects (Text I/O or Multi-Line Text objects) are identified using the **text** animation name. All text objects with the matching **text** animation name will be affected by the string change.

One or more Language objects can be used in a design. Each Language object uses animations to store important information about the language including the language configuration file name. A separate **load** animation is used to trigger the execution of the language configuration which will change the text strings.

Altia Design provides an export feature which allows quick creation of Language configuration files using an existing design file as a template.

## Exporting a Language Configuration File

1. From within Altia Design, select one or more objects that will be included in the language. If no objects are selected, the entire design (all objects) will be used in the language configuration file.
2. Start the language export using the **File** ribbon's **Export/Configuration Files/Language File** option. The following dialog will appear:



3. Select the desired export options. The options are described in the following table:

Export in Unicode (UTF-8) format	By default, the language configuration file will be exported in UTF16. Selecting this option will force the language configuration file to be UTF8 format.
Include pixel size data	Includes the <code>_WIDTH</code> and <code>_HEIGHT</code> tokens in the language configuration file.
Include object type data	Includes the <code>_TYPE</code> token in the language configuration file.
Include font name data	Includes the <code>_FONT</code> token in the language configuration file.
Include text objects with max_pixel_count = 0	Includes all Text I/O objects that have a <code>max_pixel_count</code> animation state of 0.
Include multi-line text objects	Includes all Multi-Line Text objects (MLTOs). These objects will be included in the language configuration file.

4. Press the **OK** button and select a name and location for the language configuration file.
5. Open the language configuration file in any text editor and make desired modifications to the language data as necessary. Even with no modifications, the language configuration file is directly useable by the Language object.

**NOTE:** If multiple text objects (Text I/O or Multi-Line Text objects) have the same text animation name, only the first text object will be used during the export operation.

## Loading a New Language Using a Language Object

1. With the Graphics Editor in Edit mode, select a Language object. If your languages contain special characters, make sure you are running a Unicode version of Altia Design.
2. Open the Animation Editor and highlight the `language_file_load` animation in the Animation List Area. This will place `language_file_load` in the **Animation** field of the Animation Editor. Set the Value field to any value. As soon as the value is changed, or “tickled”, this starts the loading of the Language object configuration file that was last given to the `language_file_name` animation. Alternately, you can force a load of the language file by selecting **Yes** from the dropdown for the **Load Image File Now** property in the **Properties** pane.

## Language Configuration File Syntax

The language configuration file is a token-based command file. Each line in the file represents one command to be executed by the Language object. A command can affect a single **text** animation. This **text** animation may exist for one or more Text I/O or Multi-Line Text objects.

Each command is comprised of tokens and their associated values (in double-quotes). The following tokens are allowed in the configuration file:

<b>_NAME</b>	The text animation name affected by this command.
<b>_TEXT</b>	The new text string to load into the text animation.
<b>_TYPE</b>	The type of text object. The possible values are TEXTIO and MLINETEXT which represent Text I/O objects and Multi-Line Text objects, respectively.
<b>_WIDTH</b>	The value of the <code>max_pixel_count</code> animation for the associated Text I/O object. For Multi-Line Text objects, this is the value of the <code>ml_automrap_pixels</code> animation.

<u>_HEIGHT</u>	The height of the font in pixels. For Multi-Line Text objects, this is the pixel height of the object (i.e. number of rows multiplied by the height of the font).
<u>_FONT</u>	The current UNIX font specification for the associated object.

The \_WIDTH, \_HEIGHT, and \_FONT tokens are provided as extra information when a language configuration file is exported from Altia Design. They are provided to allow post-processing of the language configuration file using 3rd party language tools. These tokens are not processed when the language configuration file is loaded by the Language object.

## An Example Language Object Configuration File

```
# Altia Language Data File
_NAME="setting_menu_title_readout_text" _TEXT="Settings
Menu"
_NAME="setting_menu_item_1_readout_text" _TEXT="Language"
_NAME="setting_menu_item_2_readout_text" _TEXT="Time"
_NAME="setting_menu_item_3_readout_text" _TEXT="Date"
_NAME="setting_menu_item_4_readout_text" _TEXT="Format"
```

## Special Notes About Language File Construction

The “#” character can be used to start a comment line in a skin configuration file. Comment lines will be ignored by the Language object at runtime.

Token values must be enclosed within double-quotes.

No spaces can exist between a token, the “=” sign, and the first double-quote symbol.

## Language Object Animations

If a design does not already have a Language object, add one from the Language models library. To do this, go to the **Insert** ribbon, then select Language from the Models Library gallery dropdown to open it into an Altia Models View. Drag the Language object from the Altia Models View into the design. The Language object has the following animations (and it has properties and connections for these animations):

<code>language_file_name</code>	The name of the language configuration file. This name may include a relative or absolute file path.
<code>language_file_load</code>	Triggers the loading and execution of the language configuration file. Write any value to this animation to start the loading of the chosen language configuration file.

## Using the Language Object with DeepScreen

Two options are available for the Language object when generating code using Altia DeepScreen. First, you can choose to generate code for the language configuration files (this is the default action). Secondly, you can choose to load language configuration files from the file system at runtime.

By default, DeepScreen will convert the language configuration file for each Language object into a data structure in `languageData.c`. This is the most efficient approach for runtime performance and resource utilization.

When DeepScreen auto-generates language data into `languageData.c`, it uses the currently referenced language configuration file for each Language object in the design. A separate Language object is required for each language used by the design. This also requires that all possible languages be known at code generation time.

Optionally, the same language configuration files used in Altia Design can be used at runtime. To use the language configuration files directly, select the **Language Object use files** check-box from the **Code Generation Options** dialog window. For this usage, the language configuration file can be changed at runtime by sending the new file and path to the `language_file_name` animation. This allows for a single Language object to be used by the design which can load any number of languages at runtime

## Typical Language Object Uses

In run mode, it is customary to control a Language object's animation from input stimulus, or an application program. A Language object is ideally suited for quickly changing the text values of multiple Text I/O objects automatically within your design. This not only makes it much easier than editing each value individually, but it allows you to manage your translation data outside of Altia Design using simple .txt files to control which strings should be used for which languages.

## 10.5.6 Layer Manager Object Model

Special Functions/Layer Manager.dsn

The file `Layer Manager.dsn` contains Layer Manager objects preconfigured for different display resolutions. The Layer Manager Object is a special container object used for creating multi-layer HMIs for embedded targets with multi-layer display controllers. Each Layer Manager object will be treated as a separate "display".

Using a Layer Manager Object, you can control the presentation of many subgroups of objects (referred to as "layers"). The layers are similar to cards in a Deck Object. However, unlike a deck, more than one layer can be visible at a time. There is no limit to the number of layers you can have in a Layer Manager. When using a Layer Manager to create an HMI for an embedded device, there will be a limit to the number of layers that can be visible at one time (refer to the DeepScreen documentation for your specific hardware device).

A design can have more than one Layer Manager. In general, each Layer Manager will be associated with a physical display (or window if using a windowing operating system). A Layer Manager Object can only be placed at the top level (focus level 0) in your Altia Design project. In addition, the Layer Manager should never be transformed (scaled, rotated, distorted, etc.) Instead apply transformations to objects inside the individual layers of the Layer Manager.

To add a Layer Manager to your design, click on a Layer Manager Object in the Layer Manager model library and drag it into your design's work area.

### Adding Objects to a Layer in the Layer Manager

1. With the Graphics Editor in **Edit** mode, select a Layer Manager object.
2. Open the Animation Editor and highlight the `layer_id` animation in the Animation List Area. This will place `layer_id` in the **Animation** field of the Animation Editor. Choose a layer by choosing the desired state value. The first layer of the Layer Manager has a state value of 0, the second a state value of 1, etc.

**NOTE:** Keep in mind that layers have a Z-Order when drawn in the Layer Manager. Layer 0 will be on top of Layer 1 which will be on top of Layer 2, etc. Layer numbers can be reassigned using the Navigator or the Layer Manager Configuration GUI. Negative layer numbers can also be used. In this case Layer -2 would be on top of Layer -1 which will be on top of Layer 0, etc. The layers will appear in the proper Z-Order in both the Navigator and Layer Manager Configuration GUI.

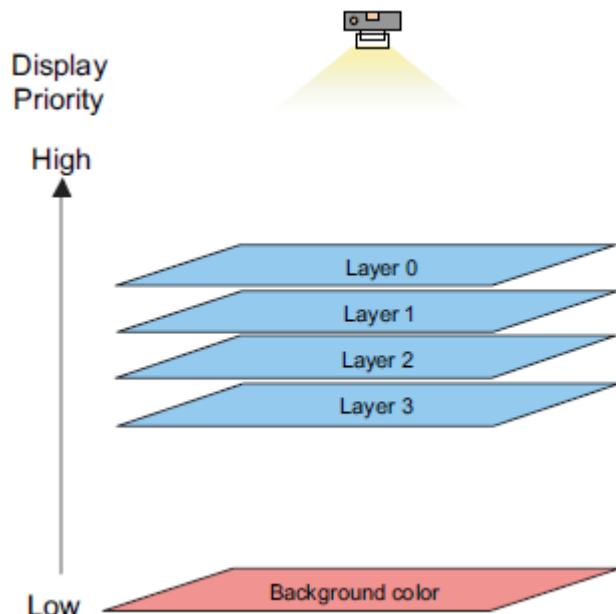
3. Add objects to the layer you have chosen by clicking the **Focus In** control on the Home Ribbon in the Graphics Editor. If you wish to add an existing object, select the object and press **Ctrl+C** (copy) before focusing in on the layer. Paste the object into the layer after you have focused in on it. To edit existing objects in a layer, **Focus In** on the layer and make appropriate changes to the objects within it.

- When you are done adding objects to a layer or editing existing objects, select **Focus Out** and choose your next layer via the `layer_id` animation's state value.

Only layers that are visible can receive input stimulus. Layers that are not visible continue to receive animation state changes if they have any assigned animation. If an object receives state changes while hidden, it will be correctly updated when the layer is made visible again. In addition, timer stimulus or control can be activated for objects in a hidden layer. Timer stimulus will continue to be active until it receives a stop event.

## Layer Order

The layers appear in a top-down order (Z-order) with layer 0 on top of layer 1 and so on. In addition, the Layer Manager has a background color which appears behind all layers. The following diagram illustrates how the layers are combined in the Layer Manager:



**NOTE:** Different DeepScreen Targets will have restrictions on how many layers can overlap at one time. In addition, there will be restrictions on the maximum number of layers that can be visible at one time. Refer to the DeepScreen Target documentation for further details.

## Layer Manager Animations

The Layer Manager has many animations used to configure and manipulate the appearance of the object as well as the layers within the object. In addition, layers can be assigned to Layer Groups to quickly configure the appearance of many layers at the same time. This section describes the animations of the Layer Manager.

<b>display_id</b>	A unique numerical identifier for the Layer Manager. When using multiple Layer Managers this identifier will be used to associate each Layer Manager with a specific embedded display. <b>This animation s only used in Altia DeepScreen.</b>
<b>display_enable</b>	This animation turns the Layer Manager "on" or "off". It's similar to turning an embedded display on and off. A value of zero will disable the Layer Manager and it will appear as a black rectangle in Altia Design. A non-zero value will enable the Layer Manager and it will operate normally, showing all visible layers.
<b>display_color</b>	This animation sets the background color for the Layer Manager. This is the color shown behind all the visible layers. If no layers are visible, the Layer Manager will appear as a solid rectangle drawn in this color.
<b>display_format</b>	This animation is used to simulate different color formats for embedded display hardware. <b>This animation is not yet implemented and nothing will happen when assigning new values to it.</b>
<b>display_width</b>	This animation sets the pixel width of the Layer Manager. All layers in the Layer Manager will be clipped to this region. The background color will be drawn to fill this region. When using Altia DeepScreen, this value is the pixel width of the embedded display.

<b>display_height</b>	This animation sets the pixel height of the Layer Manager. All layers in the Layer Manager will be clipped to this region. The background color will be drawn to fill this region. When using Altia DeepScreen, this value is the pixel height of the embedded display.
<b>layer_id</b>	This animation is similar to the "card" animation for a Deck Object. Use this animation to set the <b>active</b> layer for the Layer Manager. All the subsequent layer animations will manipulate the layer assigned to <b>layer_id</b> .
<b>layer_visible</b>	This animation sets the visibility for the <b>layer_id</b> layer. A zero value makes the layer hidden (not shown). A non-zero value makes the layer visible (shown).
<b>layer_alpha</b>	This animation sets the opacity for the <b>layer_id</b> layer. A zero value makes the layer completely transparent. A value of 255 makes the layer completely opaque (i.e. the layer is not faded from its original state).
<b>NOTE: A layer with zero opacity is still visible! That means it will still receive input stimulus.</b>	
<b>layer_x</b>	This animation sets the horizontal position for the <b>layer_id</b> layer. This value is relative to the bottom left corner of the Layer Manager. Negative values will shift the layer off the left side of the display. Positive values will shift the layer towards the right side of the display.
<b>layer_y</b>	This animation sets the vertical position for the <b>layer_id</b> layer. This value is relative to the bottom left corner of the Layer Manager. Negative values will shift the layer off the bottom side of the display. Positive values will shift the layer towards the top side of the display.

<b>layer_width</b>	This animation sets the width for the <b>layer_id</b> layer. The layer width is a window into the layer buffer. Only the layer buffer contains pixels that can be shown in the Layer Manager. The layer width can be a different size from the layer buffer. Only the portion of the layer buffer that intersects with the layer width will be visible in a layer.
<b>layer_height</b>	This animation sets the height for the <b>layer_id</b> layer. The layer height is a window into the layer buffer. Only the layer buffer contains pixels that can be shown in the Layer Manager. The layer height can be a different size from the layer buffer. Only the portion of the layer buffer that intersects with the layer height will be visible in a layer.
<b>layer_offset_x</b>	This animation sets the horizontal buffer offset for the <b>layer_id</b> layer. This sets the buffer position that will appear at the left edge of the layer window. A value of zero means the left edge of the buffer will appear at the <b>layer_x</b> position in the Layer Manager. Negative values are allowed (which will shift the buffer towards the right of the layer window).
<b>layer_offset_y</b>	This animation sets the vertical buffer offset for the <b>layer_id</b> layer. This sets the buffer position that will appear at the bottom edge of the layer window. A value of zero means the bottom edge of the buffer will appear at the <b>layer_y</b> position in the Layer Manager. Negative values are allowed (which will shift the buffer towards the top of the layer window).
<b>layer_buffer_width</b>	This animation sets the width of the buffer for the <b>layer_id</b> layer. The buffer size sets the drawable region of the layer. Objects must reside within the layer buffer dimensions to be drawn. In addition, the buffer is clipped by the layer width/height when it is shown in the Layer Manager. Only positive values are allowed for the buffer size.

<b>layer_buffer_height</b>	This animation sets the height of the buffer for the <b>layer_id</b> layer. The buffer size sets the drawable region of the layer. Objects must reside within the layer buffer dimensions to be drawn. In addition, the buffer is clipped by the layer width/height when it is shown in the Layer Manager. Only positive values are allowed for the buffer size.
<b>layer_buffer_flags</b>	<p>This animation sets draw flags for the <b>layer_id</b> layer.</p> <p><b>This animation is only used in Altia DeepScreen.</b></p>
<b>layer_blend_mode</b>	<p>This animation sets the blend mode for the <b>layer_id</b> layer. Currently the modes 0 (NORMAL), 1 (VIDEO), and 2 (CHILD) are supported.</p> <p>NORMAL layers will be drawn showing all the objects contained in the layer (just like a normal group of objects).</p> <p>VIDEO layers will not be drawn, but will instead show a video stream. In Altia Design the video stream will be from the enumerated list of video sources provided by the Video For Windows (VFW) library. The exact stream can be specified using the <b>layer_blend_target</b> animation.</p> <p>CHILD layers are special layers that are manipulated by a parent layer. All CHILD layers must have a parent NORMAL layer. See the section on CHILD layers.</p>
<b>layer_blend_target</b>	<p>This animation sets the blend target for the <b>layer_id</b> layer.</p> <p>For VIDEO layers, this will set the video source. For NORMAL and CHILD layers, this animation is non-functional.</p>

<b>layer_color</b>	This animation sets the color for the <b>layer_id</b> layer. For ALPHA layers, this is the color used to blend with the alpha pixels in the layer buffer. For all other layer formats, this is the background color of the layer. Layers formats with an alpha component will have transparent backgrounds regardless of what color is assigned. The color is a text value and can be a color like "black", "dark blue", etc. The text can also be a hex color format like "#FF0080" where the bytes are red, green, and blue (i.e. #RRGGBB) in the range 00 to FF.
<b>layer_chroma</b>	This animation sets the chroma color for the <b>layer_id</b> layer. All pixels in the layer buffer that match this color will appear transparent. The color is a text value and can be a color like "black", "dark blue", etc. The text can also be a hex color format like "#FF0080" where the bytes are red, green, and blue (i.e. #RRGGBB) in the range 00 to FF.
	<b>The value "none" is a special text string which means no chroma color is used for the layer.</b>
<b>layer_format</b>	This animation sets the pixel color format for the <b>layer_id</b> layer. The buffer for this layer will use the specified color format.  <b>Refer to the color format table for allowed format values.</b>
<b>layer_palette</b>	This animation sets the palette for the <b>layer_id</b> layer. This animation is a text value which specifies a color palette file (.pal).  <b>This animation is only used in Altia DeepScreen.</b>

**group\_id** This animation activates the specified Layer Group. All layers in the Layer Manager will be configured to match the composition of the specified Layer Group. An invalid (undefined) Layer Group will not change any layers.

**Refer to the Layer Manager Configuration GUI section in this User Guide for details on how to create Layer Groups.**

## Layer Color Formats

Each layer has a color format which describes how colors are stored for each pixel in the layer buffer. The color format is specified in the **layer\_format** animation as a numerical identifier. The following table describes the numerical IDs:

- 0** This is the 24-bit RGB format
- 1** This is the 32-bit ARGB format
- 2** This is the 16-bit RGB format
- 3** This is the 16-bit ARGB format
- 4** This is the 8-bit ALPHA format
- 5** This is the indirect color (palette) format. The bit-depth is determined by the number of unique colors used in the layer. A maximum of 255 colors are allowed (one color is reserved for chroma). Each color is 24-bit RGB.
- 6** This is an indirect color (palette) format with separate alpha channel. This is the same as format '5', where the colors are stored as 24-bit RGB, however a separate 8-bit alpha mask is used for alpha capabilities. The bit-depth is always 16-bits (8 bits for alpha value, 8-bits for palette index).

**NOTE:** Different DeepScreen Targets will have restrictions on which color formats can be used. In addition, there will be restrictions what objects can be put into a layer of a certain format (example: no lines in palette layers). Refer to the DeepScreen Target documentation for further details.

## Layer Size vs. Buffer Size

### Layer Buffer Size

The `layer_buffer_width` and `layer_buffer_height` animations will set the dimensions of the frame buffer for the active layer. When focusing into a Layer Manager, the buffer dimensions for the active layer will be shown on the Altia Design canvas as a solid outlined rectangle. The rectangle will be filled with the layer color (specified in the `layer_color` animation). Only objects within the buffer region will be drawn for a layer. **The buffer is the portion of the layer that resides in memory and contains all the pixels for the layer.**

### Layer Size

The `layer_width` and `layer_height` animations will set the layer "window". This is "window" is the portion of the layer that will appear in the Layer Manager. When focusing into a Layer Manager, the window dimensions for the active layer will be shown on the Altia Design canvas as a dashed outlined rectangle. **The window is the portion of the layer buffer that is visible in the Layer Manager.**

## Layer Position vs. Buffer Position

### Layer Buffer Position

The `layer_offset_x` and `layer_offset_y` animations will set the offset of the layer buffer relative to the layer window. In short, the offset is the position of the buffer in the window. An offset of (0,0) means the bottom left corner of the layer buffer will be aligned to the bottom left corner of the layer window. **The offset animations can be used to create a "pan" effect by scrolling a larger layer buffer behind a smaller layer window.**

### Layer Position

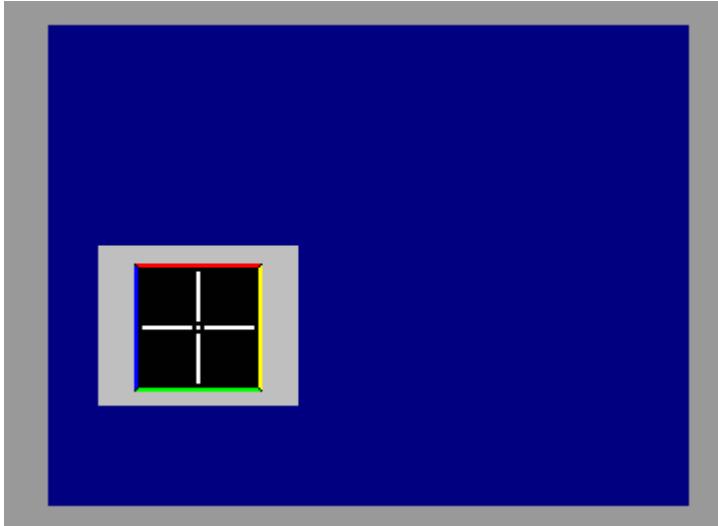
The `layer_x` and `layer_y` animations will set the layer "window" position in the Layer Manager. The window position is relative to the bottom left corner of the Layer Manager. A position of (0,0) means the layer window will appear at the bottom left corner of the Layer Manager.

Because the buffer is positioned within the layer window using the "offset" animations, changing the layer position has the effect of moving the buffer around in the Layer Manager.

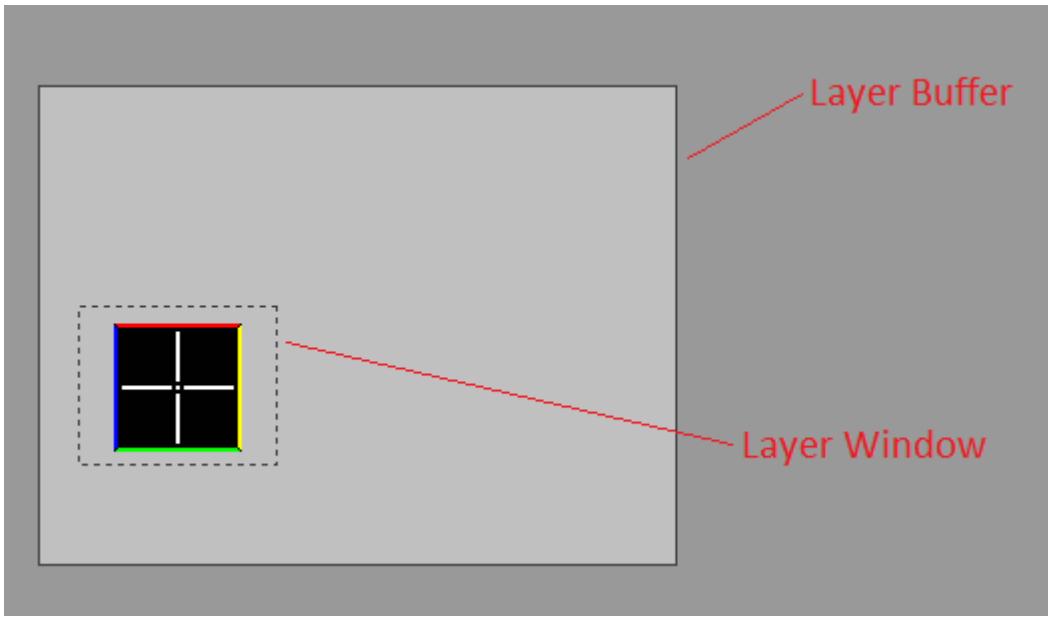
## Altia Design Helpers

When focused out of a Layer Manager, the object will show all visible layers using the current configuration for each layer. The following picture shows an example of a simple QVGA sized Layer Manager. The background color for the Layer Manager is set to "dark blue". There is a single visible layer

in the Layer Manager with a width of 100 and a height of 80 (layer width and height animations). The layer position is set to (25, 50) using the x/y animations. The layer color is set to "gray" and contains a simple square image with basic colors.



When focused into a Layer Manager, the objects for the active layer (set by the `layer_id` animation) will be shown. The layer buffer will be shown with a solid outline and filled with the layer color. In addition the layer window will be outlined with dashes. The following picture shows the same layer from before. The layer has a buffer with a width of 320 and a height of 240 (buffer width and height animations). The buffer is offset from the layer window by (20, 50) using the offset animations.



The following picture shows the animation values for the Layer Manager used to configure the above example:

Animation

Group by focus level      Advanced Options ▾

Animation Name	State	Init	Low	High
↳ QVGA Display [Layer Manager, id# 1]				
1_qvga_display_color	"dark blue"	0	?	?
1_qvga_display_enable	1	1	?	?
1_qvga_display_format	0	0	?	?
1_qvga_display_height	240	240	?	?
1_qvga_display_id	0	0	?	?
1_qvga_display_width	320	320	?	?
1_qvga_layer_alpha	255	255	?	?
1_qvga_layer_blend_mode	0	0	?	?
1_qvga_layer_blend_target	0	0	?	?
1_qvga_layer_buffer_flags	0	0	?	?
1_qvga_layer_buffer_height	240	240	?	?
1_qvga_layer_buffer_width	320	320	?	?
1_qvga_layer_chroma	"none"	0	?	?
1_qvga_layer_color	"gray"	0	?	?
1_qvga_layer_format	0	0	?	?
1_qvga_layer_height	80	80	?	?
1_qvga_layer_id	1	0	?	?
1_qvga_layer_offset_x	20	20	?	?
1_qvga_layer_offset_y	50	50	?	?
1_qvga_layer_palette	"auto"	0	?	?
1_qvga_layer_visible	1	1	?	?
1_qvga_layer_width	100	100	?	?
1_qvga_layer_x	25	25	?	?
1_qvga_layer_y	50	50	?	?
1_qvga_group_id	0	0	?	?

Jump to Prev    Jump to Next     Snap to defined states

Animation:  State:

## Child Layers

A CHILD layer is created by setting the `layer_blend_mode` animation to a value of 2. All child layers must have a parent. The parent is the first NORMAL layer (blend mode 0) that appears above the child in the Layer Manager UI (or in the Navigator). Child layers will be indented under their associated parent layer.

Child layers are modified by their parent layer. The parent layer will override or modify child layer data. The following table lists the data that is modified and how it is modified:

<b>Visibility</b>	The child layer will be hidden whenever the parent layer is not visible.
<b>Format</b>	The pixel format of the child is set to match the parent.  <b>IMPORTANT:</b> Only transparent formats will be useful (ARGB32, ARGB16, ALPHA, and APAL). Opaque color formats will obscure the child layers.
<b>Color</b>	The background color of the child is set to match the parent.
<b>Chroma</b>	The chroma color of the child is set to match the parent.
<b>Opacity</b>	The opacity of the child layer will be modulated by the opacity of the parent layer.
<b>Width/Height</b>	The child layer will be clipped by the dimension of the parent.
<b>Position</b>	The child layer's position becomes a relative position off of the parent's position. If the parent layer is moved, all child layers will also move.
<b>Buffer Offsets</b>	The child layer's position will be modified by the parent's buffer offsets (similar to how the parent's buffer would move when the offsets are changed).
<b>Palette</b>	The palette for the child is set to match the parent.

## 10.5.7 Mask Object

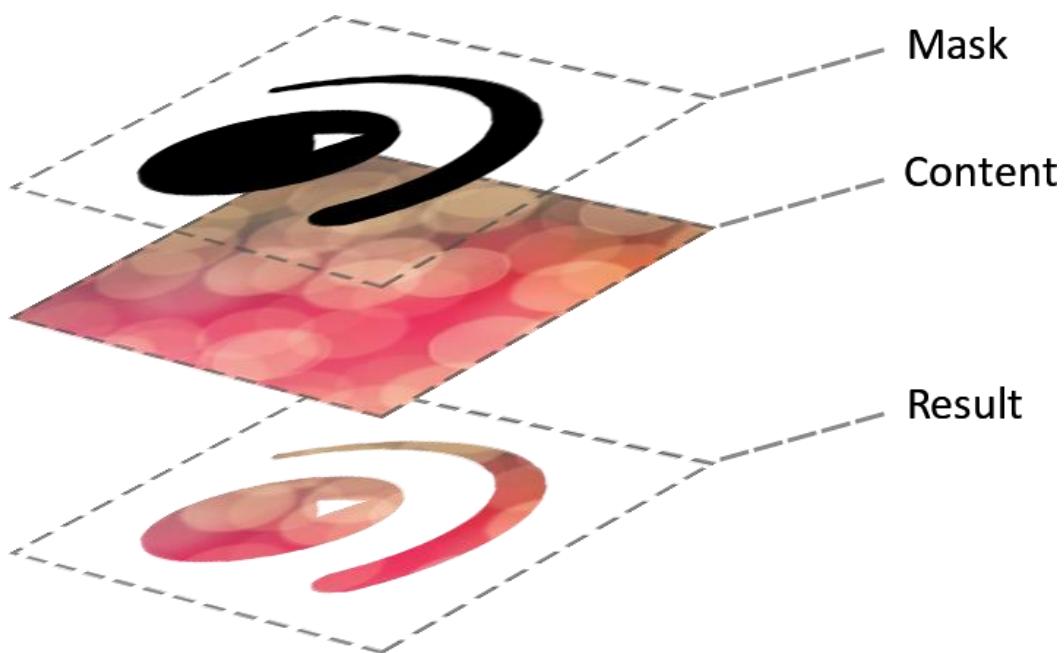
Special Functions/Mask Object.dsn

The Mask Object is a special container object that allows you to apply masking to its child objects to create sophisticated visual effects. Masking is the combining of multiple images or objects to create a composite image with selective areas of transparency.

The Mask object has two containers, similar to a Deck object's "cards".

- The '**Mask**' container may contain any number of Altia objects. Any transparent area within the Mask container will be applied to the contents of the Content container. If an object in the Mask container is an Image Object or imported raster that contains an alpha channel (typically .png file format), the alpha channel's transparency will also be applied to the contents of the Content container.
- The '**Content**' container may contain any number of Altia objects.

The contents of the Mask and Content containers are combined into a resulting composite image.



**NOTE:** Objects in the Mask and Content containers may contain animation, stimulus, and control logic – even other Mask objects! Very sophisticated visual effects and behaviors can be achieved this way.

To add a Mask object to a design, simply open the Mask object library and drag the Mask container into the design's work area.

## Mask Object Animations

The Mask object has several animations used to set its clipping boundaries and to allow insertion of Altia objects into the Mask and Content containers.

**width** This animation sets the pixel width of the Mask object. All objects inside the Mask object's containers will be clipped to this region.

**height** This animation sets the pixel height of the Mask object. All objects inside the Mask object's containers will be clipped to this region.

**container** This animation is similar to the "card" animation for a Deck Object. Use this animation to set the active container.

Valid **container** values are:

- 0 - Content container
- 1 - Mask container

Values below 0 or above 1 are invalid. If a value other than 0 or 1 is entered and a **Focus In** is performed, the **Focus In** operation will enter the Content container.

**mode** The mode animation affects the way that the mask and content are combined.

Valid **mode** values are:

- 0 - Normal
- 1 - Inverted

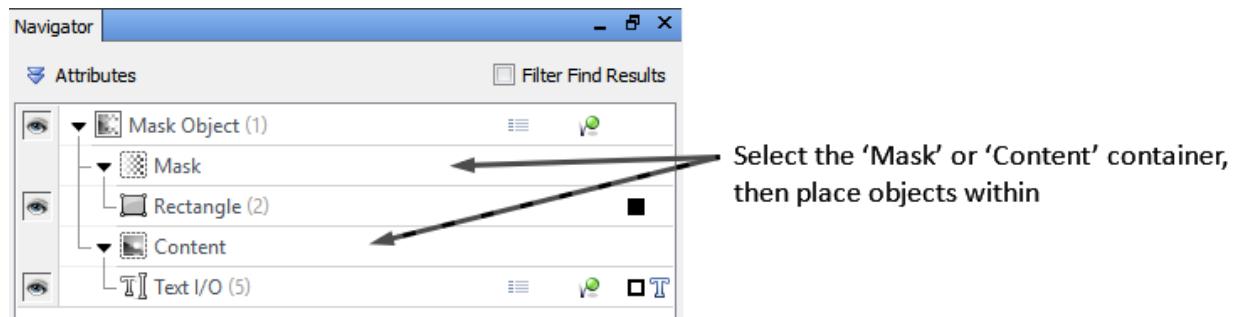
When **mode** is set to inverted, the transparency of the content within the Mask container is inverted before combining with the Content container's contents.

**NOTE:** The size of the width and height directly affect the amount of memory required by the Mask object.

Try to keep these values as small as possible to minimize memory consumption in DeepScreen generated code.

## Placing Objects within Mask Object Containers

Placing objects into the containers within a Mask object can be done a few different ways. The simplest is to use the **Navigator** to select the Content or Mask containers and simply drag-and-drop or copy/paste objects into them.



See [Chapter 2: Navigator](#) for more information on using the Navigator.

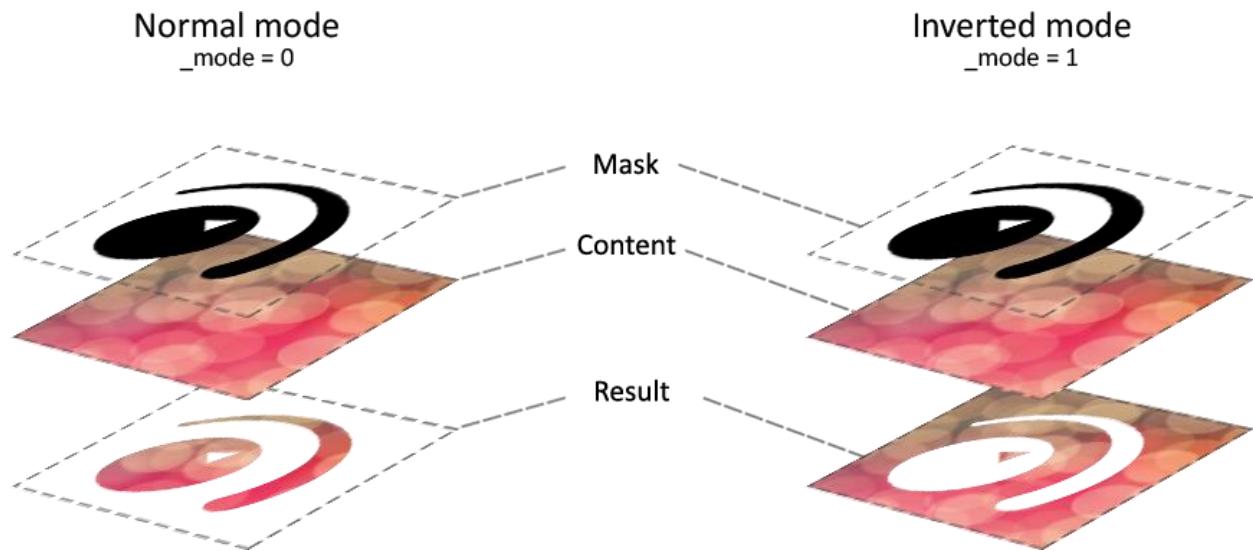
The second method is to manipulate the Mask Object's **container** animation directly, as described in the following steps:

1. With the Graphics Editor in **Edit** mode, select a Mask object.
2. Open the Animation Editor pane and highlight the **container** animation in the Animation list. This will place **container** in the **Animation** field of the Animation Editor. Choose a container by specifying the desired state value. The Content container has a state value of **0**. The Mask container has a state value of **1**.
3. Add objects to the container you have chosen by clicking the **Focus In** button on the Home Ribbon in the Graphics Editor. If you wish to add an existing object, select the object and press **Ctrl+C** (copy) before focusing in to the container. Paste the object into the container after you have focused in to it. To edit existing objects in a container, **Focus In** to the container and make appropriate changes to the objects within it.
4. When you are done adding objects to or editing objects within a container, select **Focus Out** and choose your next container via the **container** animation's state value.

## Mask Object Special Considerations

### Inverted Mask Mode

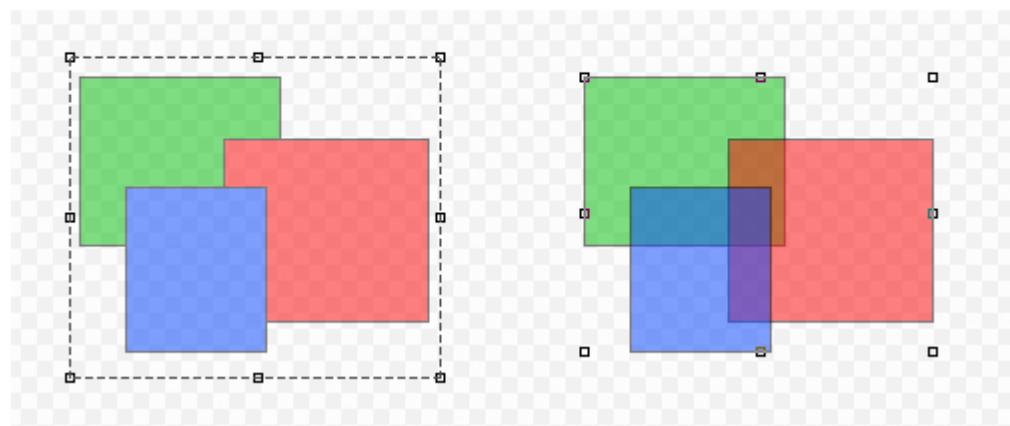
When the Mask Object's `mode` animation is set to state 1, the Mask object enters Inverted mode. When in inverted mode, the transparency of the contents of the Mask container is inverted before combining with the Content container's objects to create the composite image.



### Mask Opacity

Objects within the Mask object are composited together in a manner similar to the Snapshot object. This provides some visual behaviors unique to the Mask object. If the opacity of the Mask object is set to a value less than 100%, the opacity is applied to the composited image as a whole instead of setting each child object's transparency individually as with other container objects such as groups or Clip objects.

**Mask object**  
(Mask set to 50% opacity)      **Group of objects**  
(Group set to 50% opacity)



## **Empty Mask or Content Containers**

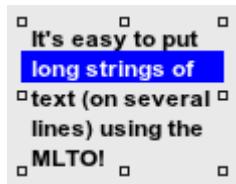
The Mask object combines the objects within the Content and Mask containers to form a composite image. If no objects exist in the Content container, the Mask object will appear invisible in the model universe. If no objects exist in the Mask container then no mask can be applied to the objects within the Content container, therefore the objects within the Content container will be fully visible.

## **Transforming Mask Objects**

Be sure not to transform the Mask object (scale, rotate, distort, etc.) before focusing in and adding or editing objects in the Mask or Content containers. This will apply an undesired transform to the child objects and may cause erroneous results when the Mask object is drawn afterwards.

## 10.5.8 Multi-Line Text Object Models

Text/Multi-Line.dsn



Multi-Line Text Objects (MLTO) are special Altia built-in objects that manage multiple lines of text. MLTOs help a user enter multiple lines of text into an Altia design while it is running. They also allow a control statement or client program to display strings of text dynamically.

To add an MLTO to your design, click on it and drag it into the design's work area. Changing animation names for these objects is automatically done which gives you the ability to have multiple MLTOs working independently in a single design. To manually change names for existing objects, choose **Rename Animations...** from the Animation Editor's right-click context menu, or by clicking **Rename** on the **Home** ribbon.

You can change the font, color, or pattern of any MLTO. You can also move it, stretch it, etc. To perform these operations, put the Graphics Editor into **Edit** mode and click on the object to select it. Choose the tool or command you wish to execute and proceed as you would with a normal object.

## What Can the Multi-Line Text Object (MLTO) Do?

In the MLTO, it is possible to:

- Set the current line number to change (`set_line`)
- Send a text string to the current line (`text`)
- Send a single text character to the current line (`character`)
- Choose whether new text replaces existing text or is added to it (`insert_mode`)
- Activate a highlight bar behind a specific line (`hilight`)
- Highlight the line that is currently under the mouse cursor (`click`)
- Delete a line (`clear`)
- Set the length of empty space to the left of text (`left_pad`)
- Set the length of empty space to the right of text (`right_pad`)
- Set the height of empty space between lines of text (`vert_pad`)
- Set the style of the empty space between lines of text (`vert_style`)
- Set the minimum width of the highlight bar (`hilight_min_width`)
- Set the character count at which lines should wrap when setting the text (`autowrap_chars`)
- Set the pixel count at which lines should wrap when setting the text (`autowrap_pixels`)
- Set the color of the highlight bar (`hilight_color`)
- Set the justify mode of the lines of text (`justify`)

The MLTO also has several built-in animations that generate their own events. The MLTO can report:

- The text contained in a single line or all lines (`text`)
- Each change in the total width of the object (in pixels) due to a change of the object's contents (`width`)
- Each change in the total height of the object (in pixels) due to a change of the object's contents (`height`)

## Adjusting the Colors of an MLTO

There are following parts of the MLTO that can be affected by color: unselected text, selected text, the highlight bar, and individual characters or words.

- The base color of the MLTO text is determined by the object's *outline* color setting.
- The color of the text when it is highlighted (that is, when the highlight bar is activated for that particular line with the **hilight** animation), is determined by the object's *fill* color setting.
- The color of the highlight bar is set by the **hilight\_color** animation. This animation is described in detail later in this section.
- Individual words or letter colors can be specified as part of the MLTO text using its **text** or **character** animation. The color is specified in an html-like style tag as such:

```
<font color="#0000FF">will make this text blue.</font>
```

The color value must be a hexadecimal six character RGB value. Any number of tags can be set and there is no limit on the length of a font color tagged section of text. Nested tags are not supported.

## Descriptions of MLTO Built-in Animations

### • The MLTO **text** Animation

Send a string to this animation to display it in the Multi-Line Text Object (MLTO).

Send a string from application code using the **altiaSendText()** or **AtSendText()** Altia API functions.

How the string is displayed depends on the states of several animations (**set\_line**, **insert\_mode**, **autowrap\_chars** and **autowrap\_pixels**).

If **insert\_mode** is non-zero, then the MLTO is in insert mode. In insert mode, if **set\_line** is 0, then a new line will be added at the bottom of the MLTO and the text will be displayed in that new line. If **set\_line** is greater than zero, then the text is inserted at line **set\_line** - all lines (including whatever was at line **set\_line**) are shifted down.

If **insert\_mode** is zero and **set\_line** is 0, then the entire text of the MLTO is replaced by text. If **insert\_mode** is zero and **set\_line** is non-zero, then line **set\_line** is replaced by the string sent to text.

The following table summarizes the above two paragraphs:

Table 6-5 The behavior of <code>set_line</code> and <code>insert_mode</code> for the text animation.		
<code>insert_mode</code>	<code>set_line</code>	DESCRIPTION
<code>non-zero</code>	0	A new line is added at the end of the MLTO and text is put in that line.
<code>non-zero</code>	<code>n (&gt; 0)</code>	A new line is inserted before line <code>n</code> and <code>text</code> is put in that line. That is, <code>text</code> becomes the new line <code>n</code> and all other lines are shifted down.
0	0	The text replaces all existing text in the MLTO. How the new text is displayed is still affected by <code>autowrap_chars</code> and <code>autowrap_pixels</code> .
0	<code>n (&gt; 0)</code>	The text at line <code>n</code> is replaced by the new text.

## autowrap\_chars and autowrap\_pixels

If either of these animations is set to a value greater than 0 and `set_line` is 0, then a string sent to text is broken at `autowrap_chars` characters or `autowrap_pixels` (whichever comes first). The wrap is done at the last space, if one exists on this line - or if there is no space (that is, the current word is longer than `autowrap_chars` or `autowrap_pixels`), then the word is separated at `autowrap_chars` or `autowrap_pixels`.

If `set_line` is not zero, then no auto-wrapping is done, no matter how long the line is. The MLTO assumes that if `set_line` is not zero, any changes should be made to *that* line.

**NOTE:** `autowrap_chars` and `autowrap_pixels` do not affect text typed in directly from the editor (that is, text entered by double-clicking directly on an MLTO in the drawing area).

The `text` animation can also report the current text in an MLTO. If -1 is sent to the `text` animation, it outputs the characters of all lines of the MLTO if `set_line` is 0 or it will output just the current line of the MLTO if `set_line` is set to a valid line number. The characters are generated in sequence as `text` animation event values with a final 0 value to indicate the end of the transmission. If these values, including the final 0 value, are written in the same sequence into an array of characters (for example, `char string[50]` in C/C++), the array will hold a valid string. When using the Altia API to interface

application code to a design, use the `altiaGetText()` and `AtGetText()` API functions to easily get the string from the `text` animation.

- **The MLTO character Animation**

This animation is used to send a single character at a time to the MLTO.

Send a character from application code using the `altiaSendEvent()` or `AtSendEvent()` Altia API functions. For example:

```
AtSendEvent(id, "character", (AltiaEventType) 'a');
```

A backspace character ('\b' in C code or literally the value 8) removes the previous character.

The way a character is added to the MLTO depends on `insert_mode` and `set_line` as described by the following table.

Table 6-6 The behavior of `set_line` and `insert_mode` for the character animation.

<code>insert_mode</code>	<code>set_line</code>	DESCRIPTION
0	0	<code>character</code> is added to MLTO just like text I/O. MLTO cleared if last item sent was a string, otherwise, <code>character</code> is added to string of characters.
0	<code>n (&gt; 0)</code>	<code>character</code> replaces line <code>n</code> .
<code>non-zero</code>	0	<code>character</code> is added to the end of last line.
<code>non-zero</code>	<code>n (&gt; 0)</code>	<code>character</code> is added to the end of line <code>n</code> .

## **autowrap\_chars and autowrap\_pixels**

If either of these animations is set to a value greater than 0 and **set\_line** is 0, then characters sent to **character** will wrap at **autowrap\_chars** characters or **autowrap\_pixels** (whichever comes first). The wrap is done at the last space, if one existed on this line - or if there is no space (that is, the current word is longer than **autowrap\_chars** or **autowrap\_pixels**), the word is separated at **autowrap\_chars** or **autowrap\_pixels**.

If **set\_line** is not zero, then no auto-wrapping is done, no matter how many characters are sent to **character**. The MLTO assumes that if **set\_line** is a real line number, any changes should be made to *that* line.

- **The MLTO insert\_mode Animation**

This animation determines whether the MLTO will add new text/characters to what is already in the MLTO or overwrite the current text. If **insert\_mode** is 0, then text is overwritten. If **insert\_mode** is not zero, then text/characters sent to the **text** or **character** animations are added to the MLTO. What is overwritten or where the insertion is done depends on the value of **set\_line**.

**insert\_mode** acts like a state - the MLTO is either in insert mode or it is not. No action is taken when the **insert\_mode** animation is changed.

- **The MLTO hilight Animation**

For an MLTO that has n lines, send a number greater than zero and less than or equal to n to this animation to highlight that line. The line will have a box drawn behind it in the color last sent to the **hilight\_color** animation.

The text of the line changes to the background color for the MLTO. Text in regular un-highlighted lines is drawn in the foreground color for the MLTO.

The pattern of the highlight box can be changed using the pattern drop-down. The size of the highlight box is adjusted using the **hilight\_min\_width**, **left\_pad**, **right\_pad**, and **vert\_pad** animations described later in this document.

- **The MLTO click Animation**

If a value of 1 is sent to this animation, the MLTO detects whether the mouse is over the MLTO and, if it is, the line that it is over is highlighted and the **hilight** animation receives this value. Subsequent clicks over the same line do not generate new **hilight** events and the line stays highlighted.

If a value of 2 is sent to the **click** animation, then the click toggles the highlighted state of the line.

If any value is sent to **click** and the mouse is not over the MLTO, then **hilight** is set to 0.

If a value other than 1 or 2 is sent to **click**, it is treated just like a 1.

- **The MLTO set\_line Animation**

Nothing is done when the value of **set\_line** changes. However, **set\_line** is used in conjunction with many other animations. **set\_line** is sent a value prior to sending **text**, **character**, or **clear** values.

Table 6-7 The behavior of <b>set_line</b>	
Animation	<b>set_line</b> Purpose
<b>text</b>	Text is sent to line # <b>set_line</b> .
<b>character</b>	Character is sent to line # <b>set_line</b> .
<b>clear</b>	Line # <b>set_line</b> is deleted.

When **set\_line** is 0, actions affect the entire MLTO. **BE VERY CAREFUL CHANGING ANIMATIONS WHEN set\_line IS 0!** If **set\_line** is set to n > 0, only line n is affected.

- **The MLTO clear Animation**

Sending any value to this animation causes the MLTO to delete all of its lines if **set\_line** is 0 or less than 0.

If **set\_line** is greater than zero, sending any value to this animation causes the line specified by the **set\_line** animation to be deleted. If other lines follow the deleted line, they all shift up one line.

- **The MLTO left\_pad Animation**

Sending a value greater than or equal to zero adds that many blank pixels to the left of the MLTO text. **left\_pad** is used to shift the text to the right.

**left\_pad** works regardless of **justify\_mode** (alignment) and the upper left corner of the MLTO stays in the same place as the padding is changed.

- **The MLTO `right_pad` Animation**

Sending a value greater than or equal to zero adds that many blank pixels to the right of the longest line of text. `right_pad` is used to guarantee some amount of empty space to the right of every line of text.

`right_pad` works regardless of `justify_mode` (alignment) and the upper left corner of the MLTO stays in the same place as the padding is changed.

- **The MLTO `vert_pad` Animation**

Sending a value greater than or equal to zero adds that many blank pixels to the top and bottom of each text line in the MLTO. `vert_pad` is used to increase the spacing between the lines of text in the MLTO. It also increases the height of the highlight bar.

Sending a value less than 0 reduces the spacing between the lines. Lines can even overlap if the negative value is large enough until the entire MLTO collapses into a single line for very large negative `vert_pad` values. When the `vert_pad` is negative, the padding at the top and bottom of the MLTO remains constant at 0 pixels from the top of the top line and 0 pixels from the bottom of the bottom line.

The upper left corner of the MLTO stays in the same place as the padding is changed.

- **The MLTO `vert_style` Animation**

Controls the style of the vertical padding. MLTOs in designs created in versions of Altia Design prior to version 10.2 will automatically get this new built-in animation when the design (.dsn) file is opened into Altia Design 10.2 or newer and its value will default to 0. The new `vert_style` animation takes these values:

- 0 This is the default value and it represents the only style previously available. It provides 1x the `vert_pad` value as padding (in pixels) at the top and bottom of the MLTO and 2x the `vert_pad` value as padding (in pixels) between lines.
- 1 A new style that provides .5x the `vert_pad` value (rounded to a whole number) as padding (in pixels) at the top and bottom of the MLTO and 1x the `vert_pad` value as padding (in pixels) between lines.
- 2 A new style that provides no padding at the top and bottom of the MLTO and 1x the `vert_pad` value as padding (in pixels) between lines.

- **The MLTO `hilight_min_width` Animation**

Normally, the highlight bar is as wide as the widest line in the MLTO plus any left padding if `left_pad` is greater than zero and any right padding if `right_pad` is greater than zero. Use this animation to set the minimum width, in pixels, of the highlight bar.

- **The MLTO `autowrap_chars` Animation**

If this animation is set to a value greater than 0, then text that is input to the MLTO via the `text` or `character` animations will wrap at `autowrap_chars` characters.

Auto-wrapping is not done if `set_line` is set to a non-zero line.

- **The MLTO `autowrap_pixels` Animation**

If this animation is set to a value greater than 0, text that is input to the MLTO via the `text` or `character` animations will wrap at `autowrap_pixels` pixels.

Auto-wrapping is not done if `set_line` is set to a non-zero line.

- **The MLTO `hilight_color` Animation**

This animation takes a string like the `text` animation.

For Altia Design, Altia FacePlate, or Altia Runtime, the string is a color name (for example, “red”) or an RGB value (for example, “255 0 0”).

For DeepScreen generated code, the string must be an RGB value (for example, “255 0 0”). A color name string does nothing in DeepScreen because generated code does not contain a color name lookup table.

The color given is applied to the highlight bar.

Set the `hilight_color` from application code using the `altiaSendText()` or `AtSendText()` Altia API functions.

- **The MLTO justify Animation**

By default, text is left justified in the display. For a different justification, choose **justify** in the Animation Editor's List Area and assign it one of the following values:

- 0 Left justified (default).
- 1 Right justified.
- 2 Left justified.
- 3 Center justified.
- 4 Left justify relative to a sibling in the same group as the MLTO.
- 5 Right justify relative to a sibling in the same group as the MLTO.
- 6 Center justify relative to a sibling in the same group as the MLTO.

**NOTE:** For values of 4, 5 or 6 to work, the MLTO must be in a group with at least one other object in the same group.

- **The MLTO width Animation**

Nothing happens when this animation is changed. However, when the extent of the MLTO changes (either from the **text**, **character**, **clear**, **right\_pad**, or **left\_pad** animations), the horizontal extent of the MLTO is sent to the **width** animation. At the same time, the initial value of the **width** animation is automatically set.

The **width** and **height** animations can be used to draw a dynamically sized background behind the MLTO.

- **The MLTO height Animation**

Nothing happens when this animation is changed. However, when the extent of the MLTO changes (either from the **text**, **character**, **clear**, or **vert\_pad** animations), the vertical extent of the MLTO is sent to the **height** animation. At the same time, the initial value of the **height** animation is automatically set.

The **width** and **height** animations can be used to draw a dynamically sized background behind the MLTO.

## The MLTO Examples in mlinetext.dsn

The mlinetext.dsn library contains several examples of the MLTO:

- **A Streamlined Example**

This is just an MLTO in a group with properties and connections for the MLTO's built-in animations. If the MLTO is to be controlled from application code using the Altia API, this example is a good option. If the MLTO is just going to be used to display multiple lines of text, this is also a good option.

This example is lightweight (that is, almost no control and no stimulus). If DeepScreen is being used to generate code, this example will generate the smallest amount of data and code (as compared to the other MLTO examples in mlinetext.dsn).

Set the font for the text of the MLTO from the Altia Design font selector in the Text Tool toolbar.

**WARNING:** Set the text and text highlight colors from the properties and NOT from the Altia color picker. Setting the colors from the color picker will disable the ability to set the colors from the properties (because setting colors from the color picker sets the group's colors which override the MLTO's colors). If this happens, select the group object, open the Altia Design Object menu, and choose Clear Group Attributes.

Note that the streamlined example has a useful feature not available with the other examples in mlinetext.dsn. If you want to create a paragraph of text without regard for line breaks (that is, let the MLTO pick the line breaks for you), do the following:

1. Set the **Autowrap (pixels)** or **Autowrap (characters)** property to the desired maximum pixels or characters for a line. To avoid confusion, do not set both of these properties to non-zero. If both properties are non-zero, the property which results in the shortest line length will prevail (and this can be confusing).
2. Set the **Current Line** property to 0.
3. Use the **Current Line Text** property field to enter in the text for the paragraph. When you press Enter in the field or click the mouse outside of the field, the entire text is sent to the MLTO and it will create the line breaks passed on the setting for the **Autowrap (pixels)** or **Autowrap (characters)** property.

- **A More Complex Example**

This is also an MLTO in a group. It has slightly different properties from the streamlined example to make it easier to add lines and delete lines. It also has mouse stimulus to allow for the highlighting and selection of a line. A property named **Mouse Input** is used to enable or disable the handling of mouse input.

This example uses control to implement the **Insert New Line** and **Delete Line** properties. It uses stimulus and control to implement highlighting from the mouse.

The use of control and stimulus results in a less efficient example (that is, it uses more memory and may be slightly slower than the streamlined example). If DeepScreen is being used to generate code, this example will generate more data and code compared to the streamlined example.

This example has nearly the same connections as the streamlined example.

Set the font for the text of the MLTO from the Altia Design font selector in the Text Tool toolbar.

**WARNING:** Set the text and text highlight colors from the properties and NOT from the Altia color picker. Setting the colors from the color picker will disable the ability to set the colors from the properties (because setting colors from the color picker sets the group's colors which override the MLTO's colors). If this happens, select the group object, open the Altia Design Object menu, and choose Clear Group Attributes.

- **A Very Sophisticated Example**

This is an MLTO in a Clip object and the Clip is in a group. Putting the MLTO in a Clip makes it possible to scroll the lines of the MLTO up/down and left/right. The Clip's width and height determine the visible width and height of the MLTO. The MLTO can of course contain many more lines and these are made visible by changing the **xorigin** and **yorigin** animations of the Clip.

This example has the properties of the more complex example with the addition of a **Visible Width**, **Visible Height**, **Current X Offset**, and **Current Y Offset**. It also has the connections of the more complex example with the addition of connections to scroll and set the visible width and height.

The scrolling features of this example require the addition of the Clip object as well as control. As a result, this example is the least efficient of the examples from `mlinetext.dsn`.

Set the font for the text of the MLTO from the Altia Design font selector in the Text Tool toolbar.

**WARNING:** Set the text and text highlight colors from the properties and NOT from the Altia color picker. Setting the colors from the color picker will disable the ability to set the colors from the properties (because setting colors from the color picker sets the group's colors which override the MLTO's colors). If this happens, select the group object, open the Altia Design Object menu, and choose Clear Group Attributes.

Example scroll bars accompany this example MLTO. Each scroll bar's **Scroll Bar Output** connection is easily linked to the **Scroll Vertical** or **Scroll Horizontal** connection for the example MLTO. These connections are already linked in mlinetext.dsn. Copy them together from a Models View and the connections will be preserved. To preserve connections when copying objects from a Models View, press the left mouse button in empty space of the Models View and drag it to select multiple objects. Press on one of the selected objects to drag all selected objects from the Models View to the drawing area.

## 10.5.9 Multi Plot Object Models

Plots and Charts/Multi Plot.dsn

The file Multi Plot.dsn contains an object to facilitate the dynamic plotting and charting of data in line or filled line form. The Multi Plot Object can plot data in X/Y as well as stripchart mode. Multiple plot lines can be plotted at the same time. When data is plotted, the Multi Plot Object allows for zooming and panning to different locations. It is a highly configurable object allowing configuration of plot lines, grid lines, data markers, etc. In addition, through advanced interfaces to the Multi Plot Object, external data buffers can be directly connected and annotations, such as basic shapes and text, can be added to the plot axes.

To add a Multi Plot Object to your design, from the Models View window click on it and drag it into the design's work area.

The Multi Plot Object is basically a "window" into which data is plotted. Therefore, the only visible component of the plot when it is first inserted into a design is its outline. This outline color can be changed using the foreground color and line style menus for the object. In addition, if the object outline is hidden, the object will no longer display an outline.

**NOTE:** It may be best to leave an outline on the plot object while working on a design as an indicator of the location and size of the plot object.

The size of the Multi Plot Object can only be set by using the **width** and **height** animations. It cannot be stretched or scaled using the handles. This allows the plot to readjust the axis and plot line values if the plot changes size.

### Getting Started with the Multi Plot

The basic steps for bringing a Multi Plot Object into a .dsn file and using it are described below. The complete description of the interfaces and more detailed use instructions are in the subsequent sections.

1. Drag and drop a Multi Plot Object into your design.
2. Set the height and width of the Multi Plot Object.
3. Set the x and y axis limits as desired.

**NOTE:** Adjusting the height and width of a plot does not affect the representation of data in the plot. Similarly, adjusting the axes limits does not affect the size of the plot.

4. Use the **num\_lines** animation to set the number of plot lines that should be plotted in the axes.
5. Use the **num\_lines**, **next\_x**, **next\_y**, and **latch\_pt** animations to begin adding data to the plot lines.

## Multi Plot Object Common Line Styles

In any of the line style animations, there are several common styles used. If the line style is entered using the Properties dialog, the choices are limited to the following. However, if the line style is directly modified through the Animation Editor or through external code, any 16-bit value/pattern is acceptable. Below is a list of the common styles:

- **NO\_LINE** = 0x0000
- **DOTTED\_LINE** = 0xAAAA
- **LARGE\_DOTTED\_LINE** = 0xCCCC
- **SMALL\_DASHED\_LINE** = 0xFOFO
- **DASHED\_LINE** = 0xFF00
- **DASHDOT\_LINE** = 0xFF18
- **SOLID\_LINE** = 0xFFFF

## Multi Plot Data Buffers

By default, a Multi Plot Object has a circular buffer of data points that is the same size as the width of the Multi Plot Object in pixels. It is also possible, using the Plot Ex interface, to register an external buffer of any size with the Multi Plot Object. The buffer can either be user declared or declared by the Multi Plot Object. See the documentation for the **altiaExCreatePlotBuffer** function for more details.

When an external buffer is registered with a plot, the Multi Plot Object will plot data from the external buffer based on commands given by other Plot Ex function calls. The default buffer is always used to store the data in current view to allow for scrolling through large external buffers.

## Multi Plot Modes

The Multi Plot Object operates in two primary modes, X/Y Plot Mode and Stripchart Mode. The **mplot\_sample\_period** animation determines which mode the Multi Plot will operate in.

If **mplot\_sample\_period** = 0, the Multi Plot is in X/Y plot mode. In this mode, the plot assumes each **mplot\_next\_x** and **mplot\_next\_y** pair that is latched represents data values along the respective axis. If a point is beyond the range of the axis, the plot will not scroll as new points are added. If the panning animations are used to move the plot, the data will move along the desired axis. There is no limit to how far a plot may pan. As new points are added, if the circular buffer becomes full, the oldest point will be obliterated by the newest point.

If **mplot\_sample\_period** is a positive value, the Multi Plot will enter Stripchart mode. In stripchart mode, the **mplot\_sample\_period** defines the amount of time (in x-axis units) that passes between each new data point. The y-axis value in each **mplot\_next\_x** and **mplot\_next\_y** pair is assumed to be the data at a given sample time. The x-axis value in the pair is assumed to be the multiple of sample periods that occurred between the previously latched **mplot\_next\_x** and **mplot\_next\_y** pair and this one.

In Stripchart mode, as new points are added, the x-axis value is incremented by `mplot_sample_period * mplot_next_x` for each sample that is added. When the data reaches the end of the chart, the plot line will scroll as new points are added. If the panning animations are used to move the plot, the data will scroll along the x-axis bounded by the beginning and end of the registered data buffer. Once scrolled, the data will be locked in place and will not change as new points are added unless the circular buffer tail reaches the currently viewed data. In this case, the plot will once again shift as each new point is added.

An example of the Stripchart mode x-axis configuration is described below:

1. Set the axis min and max to define the total amount of time visible on the screen. In this example, set the min to 0 and the max to 100. Thus, our axis spans 100 time units, say milliseconds.
2. Set the sample period. In this example we'll use 10 which implies that each sample is 10 time units (milliseconds) away from the previous.
3. Add a data point. Set `mplot_next_x` to 1 and `mplot_next_y` to 2.5. This will add the first point to the stripchart at a value of 2.5 on the y-axis.
4. Add a second data point. Set `mplot_next_x` to 1 and `mplot_next_y` to 2.5. This will adds a new point 10 time units (ms) away from the previous value with a y-axis value of 3.

`(mplot_next_x * mplot_sample_period = 1 * 10 = 10 time units (ms))`

NOTE: An entered x value of 1 or less is considered to be 1 time sample spacing. Fractional time units are not supported. The sample period should be set to the minimum time increment that will be used.

5. Add a third point. Set `mplot_next_x` to 3 and `mplot_next_y` to 5. This will adds a new point 30 time units (ms) away from the previous value with a y-axis value of 5.

`(mplot_next_x * mplot_sample_period = 3 * 10 = 30 time units (ms))`

6. The process can continue in this manner indefinitely. If the circular buffer size is exceeded, the oldest values will be obliterated as new values are added.

## Grid Lines and Axis Labeling

The Multi Plot Object provides facilities for having grid lines visible in the plot area as well as reporting axis label values.

When the `mplot_grid_major_interval` is set to a non-zero value, the Multi Plot Object will add grid lines along the axis at the specified interval starting at axis value 0 if the grids are not locked or the axis min value if the grids are locked and the Multi Plot is in X/Y Plot Mode or the axis max value if the grids are locked and the Multi Plot is in stripchart mode. In addition, the Multi Plot will calculate the value and pixel position (relative to the bottom of the Multi Plot for y-axis values and relative to the left of the Multi Plot for x-axis values) of each axis major grid line. This label calculation occurs whether or not the grid lines are visible. The label values are accessible in the `mplot_axis_label_XXX` animations.

If both `mplot_grid_major_interval` and `mplot_grid_minor_interval` animations are set to non-zero values, only the major grid lines will calculate axis labels. The minor grid lines are just used for display purposes.

The visibility of the grid lines can be controlled using the `mplot_axis_show_grid` animation. This will show or hide both the major and minor grid lines. If it is desirable to control the visibility individually, the lines can be hidden using either the `mplot_axis_XXX_alpha` animations or by setting the given interval to 0.

## Multi Plot Object Interfaces

The Multi Plot Object has two different interface methods. Most functionality can be controlled through the standard animation interface. In addition, an external C-code interface is provided allowing for sophisticated features such as external data buffers and commands to directly draw "annotations" onto the plot axes.

### Animation Interface

The following animations control the behavior of the Multi Plot Object:

#### Whole-plot Animations

<code>mplot_width</code>	Sets the pixel width of the Multi Plot.
<code>mplot_height</code>	Sets the pixel height of the Multi Plot.
<code>mplot_next_x</code>	Sets the x-axis value for the next data point. The value will not be stored or drawn until the <code>mplot_latch_pt</code> is written.
<code>mplot_next_y</code>	Sets the y-axis value for the next data point. The value will not be stored or drawn until the <code>mplot_latch_pt</code> is written.
<code>mplot_latch_pt</code>	Latches the current values of <code>mplot_next_x</code> and <code>mplot_next_y</code> as a data point and stores the point to the plot.
<code>mplot_clear</code>	Clears all plot data from the plot. It does not clear grid lines or annotations.

<code>mplot_new_values_on_right</code>	Sets whether new stripchart values come in on the right or left side of the chart.
<code>mplot_sample_period</code>	Defines a unit of time between each point in a stripchart. Setting this value to a non-zero value places the Multi Plot Object in stripchart mode. Leaving this value as 0 places the Multi Plot Object in X/Y plot mode.
<code>mplot_nan_value</code>	Sets a value that is considered to be Not a Number (NaN). Any points with this value will be blank on a plot. A value of 0 in this animation indicates that there is no active NaN functionality.

## Zoom and Pan Animations

<code>mplot_zoom_x</code>	Zoom in or out on the x-axis. Positive values indicate a zoom in and negative values indicate a zoom out.
<code>mplot_zoom_y</code>	Zoom in or out on the y-axis. Positive values indicate a zoom in and negative values indicate a zoom out.
<code>mplot_zoom_reset</code>	Resets the zoom on both axes and returns the plot to the original axis settings.
<code>mplot_pan_x</code>	Pans or scrolls the plot in the x-axis direction. Numbers passed in represent the number of pixels to move.
<code>mplot_pan_y</code>	Pans or scrolls the plot in the y-axis direction. Numbers passed in represent the number of pixels to move.
<code>mplot_pan_reset</code>	Clears any active pan or scroll and returns the plot to the original axis settings.

## Axis Settings Animations

### `mplot_axis_idx`

This animation selects which axis the remaining axis animations (those with names beginning with `mplot_axis`) operate on.

Valid values are:

- `X_AXIS` = 0
- `Y_AXIS` = 1

### `mplot_axis_min`

Sets the minimum value on an axis.

### `mplot_axis_max`

Sets the maximum value on an axis.

### `mplot_axis_autoscale`

Enables automatic scaling of an axis. The axis will be sized to contain the maximum and minimum value found in the data.

Valid values are:

- 0 - Auto-scaling Disabled
- 1 - Auto-scaling Enabled

### `mplot_axis_show_grid`

Shows or hides the grid lines (both major and minor) for the selected axis.

TIP: Use the line alpha (transparency) to hide only the major or minor grid lines.

Valid values are:

- 0 - Grid lines hidden.
- 1 - Grid lines visible.

<code>mplot_lock_grid_to_axis</code>	When this animation is set, grid lines are locked to the axis even as the plot receives pan commands and the data moves.
	Valid values are:
	<ul style="list-style-type: none"> <li>• 0 - Grid lines are free to move.</li> <li>• 1 - Grid lines are locked to the axis.</li> </ul>
<code>mplot_grid_major_interval</code>	Sets the numeric data interval between major grid lines.
<code>mplot_grid_major_alpha</code>	Sets the alpha (or transparency) of the major grid lines. Valid values are (0-255).
<code>mplot_grid_major_color</code>	Sets the color of the major grid lines. Color is specified in hex strings.  String Format: "#RRGGBB
	Where RR is a hex representation 0 - 0xFF (0-255) for the red.
	Where GG is a hex representation 0 - 0xFF (0-255) for the green.
	Where BB is a hex representation 0 - 0xFF (0-255) for the blue.
	The "#" character must be the first character of the string.
<code>mplot_axis_grid_major_line_style</code>	Sets the brush pattern for the major grid lines.  The properties dialog will limit the settings to certain values, but any valid 16-bit pattern is acceptable.
	Commonly used values are defined in the section titled <b>Common Line Styles</b> .
<code>mplot_axis_grid_major_line_width</code>	Sets the line width in pixels of the major grid lines.
<code>mplot_axis_grid_minor_interval</code>	Sets the numeric data interval between

	minor grid lines.
<code>mplot_axis_grid_minor_alpha</code>	Sets the alpha (or transparency) of the minor grid lines. Valid values are (0-255).
<code>mplot_axis_grid_minor_color</code>	Sets the color of the minor grid lines. Color is specified in hex strings.  String Format: "#RRGGBB (see <code>mplot_grid_major_color</code> )
<code>mplot_axis_grid_minor_line_style</code>	Sets the brush pattern for the minor grid lines.  The properties dialog will limit the settings to certain values, but any valid 16-bit pattern is acceptable.
<code>mplot_axis_grid_minor_line_width</code>	Commonly used values are defined in the section titled <b>Common Line Styles</b> .
<code>mplot_axis_label_count</code>	Sets the line width in pixels of the minor grid lines.  <b>READ ONLY</b> Provides the number of major grid labels (and visible lines) on the axis.
<code>mplot_axis_label_idx</code>	Determines which axis label is displayed in <code>axis_label_location</code> and <code>axis_label_value</code> .  <b>READ ONLY</b> Displays the location in pixels of the grid line that the current <code>axis_label_value</code> corresponds to.
<code>mplot_axis_label_value</code>	  <b>READ ONLY</b> Displays the axis value associated with the major grid line selected by <code>axis_label_idx</code> .

## Plot Line Animations

<code>mplot_num_lines</code>	Sets the number of active lines on a plot.
	<b>NOTE:</b> Adding lines causes declaration of increased RAM for buffers.
<code>mplot_line_idx</code>	Sets which line the <code>line_XXX</code> animations and the <code>next_x</code> , <code>next_y</code> , and <code>latch_pt</code> animations operate on. Valid values = 0 - ( <code>num_lines</code> -1)
<code>mplot_line_alpha</code>	Sets the alpha value (or opacity) of a plot data line. Valid values (0 - 255).
<code>mplot_line_color</code>	Sets the color of the selected plot line. Color is specified in hex strings. String Format: "#RRGGBB (see <code>mplot_grid_major_color</code> )
<code>mplot_line_datamarker</code>	Selects which type of data marker is used on a line. The valid values are listed below: <ul style="list-style-type: none"><li>• No Markers = 0</li><li>• Circles = 1</li><li>• Squares = 2</li><li>• Triangles = 3</li><li>• Diamonds = 4</li><li>• X's = 5</li><li>• Pluses = 6</li><li>• Stars = 7</li></ul>
<code>mplot_line_fill_mode</code>	Sets whether or not a plot line is filled. Valid values are: <ul style="list-style-type: none"><li>• Line Only, No Fill = 0</li></ul>

- Line and Fill = 1
- Fill Only, No Line = 2

<code>mplot_line_fill_origin</code>	Sets the value where the fill begins. The fill will then fill the area between this value and the plot line.
<code>mplot_line_fill_color</code>	Sets the color of the fill for the selected plot line. Color is specified in hex strings.  String Format: "#RRGGBB (see <code>mplot_grid_major_color</code> )
<code>mplot_line_style</code>	Sets the brush pattern for the selected plot line.  The properties dialog will limit the settings to certain values, but any valid 16-bit pattern is acceptable.
	Commonly used values are defined in the section titled <b>Common Line Styles</b> .
<code>mplot_line_width</code>	Sets the width of the plot line in pixels.

## Plot Calculation/Information Animations

<code>mplot_calc_x_pixel</code>	Sets an x-pixel value to calculate the x-axis value of or displays the x-pixel location of a corresponding <code>mplot_calc_x_value</code> entry.
<code>mplot_calc_x_value</code>	Sets an x-axis value to calculate the x-pixel value of or displays the x-axis value of a corresponding <code>mplot_calc_x_pixel</code> entry.
<code>mplot_calc_y_val_at_x</code>	<b>READ ONLY</b> Displays the y-axis value of the currently selected plot line based on the input <code>mplot_calc_x_value</code> or <code>mplot_calc_x_pixel</code> (whichever was the last input).
<code>mplot_calc_y_pixel</code>	Sets an y-pixel value to calculate the y-axis value of or displays the y-pixel location of a corresponding <code>mplot_calc_y_value</code> entry.
<code>mplot_calc_y_value</code>	Sets an y-axis value to calculate the y-pixel value of

or displays the y-axis value of a corresponding `mplot_calc_y_pixel` entry.

## Plot Ex Function Interface

In addition to the animation interface, the Multi Plot Object can also be interfaced directly from user's C code using the Plot Ex interface.

The Plot Ex interface is a group of functions that are defined in the **altiaEx** interface that allow advanced functionality to be performed on the plot. For additional details about the **altiaEx** interface, see the [Drawing Area Object](#) documentation (located in `C:\usr\altiaXXX\help\AltiaDAO.pdf` where XXX is your version number).

In the Altia installation folder (usually `C:\usr\altiaXXX` where XXX is your version number), there is a directory called `drawingarea`. The AltiaEx and PlotEx interface are defined by the header and source files in the `lib` folder. These files are compiled into a library file called **altiaEx.lib**. To use the Plot Ex functions in your custom code, include `\drawingarea\lib\altia\include\altiaEx.h` in your source, and link **altiaEx.lib** with your project.

The custom code is accessed from the `.dsn` file using **Control Code** and, specifically, by using the **EXTERN** function.

**NOTE:** Important information about the **EXTERN** function is available in the chapter that discusses Control Code. Please refer to that chapter for details about the use of the **EXTERN** function.

When a Multi Plot Object is placed into a design, there is control code associated with it. This control code provides a method to retrieve the object id of the Multi Plot object and pass it into an initialization function in the external C code. In addition, an example **EXTERN** call is included in the comments to illustrate how the object id may be passed into the code. This object id will be used with any of the Plot Ex functions that require a **PlotId** argument. An example of this control code is shown below:

```
// This WHEN command is called when the design is loaded
// It calls a function in the user's dll (if dll functionality is used in a design).
// It allows the object's id to be passed into the dll.
//
WHEN altiaInitDesign == {}
    //First use altiaSetObj to get the object id
    SET altiaSetObj 199
    //
    // Then, call a function in the dll and pass the value in
    //
    // NOTE: If you want to use the dll functionality, the call
    // below should be uncommented, and the function and
    // dll names changed to match those of the dll to be used.
    //
    // EXTERN "setObjId" {altiaSetObj} "test.dll" NONE {}
    //
```

The functions in the Plot Ex interface are described in detail in the following section.

## Plot Ex Function Description

```
PlotBuffId altiaExCreatePlotBuffer ( unsigned int bufferSize,
                                         PlotPoint * bufferStorageLoc,
                                         unsigned int bufferStorageSize
                                       )
```

Creates a circular buffer for use with a plot object.

If a storage location is passed in, the location provided will be used for the buffer storage and is assumed to be the size defined by the parameter. If the storage location is NULL or the storage size is 0, the storage will be dynamically allocated to the bufferSize assuming dynamic allocation is allowed. If it is not, the buffer will remain 0 sized and the id returned will be negative. If the declared storage size is smaller than the desired buffer size, the buffer size will be limited to the storage size.

### Parameters

**bufferSize** Desired circular buffer size.

**bufferStorageLoc** Pointer to memory declared for the buffer contents

**bufferStorageSize** Size of the memory pointed to by bufferStorageLoc

### Returns

id of the buffer for use in other API's if successful, -1 otherwise.

```
ALTIA_BOOLEAN altiaExDestroyPlotBuffer ( PlotBuffId buffId )
```

Destroys a plot buffer and deallocates the associated memory.

If dynamic memory is not used, the memory is not deallocated, but the statically declared buffer is released so that it may be reused.

### Parameters

**buffId** Id of the buffer to destroy.

### Returns

id of the buffer for use in other API's if successful, -1 otherwise.

```
ALTIA_BOOLEAN altiaExDrawPlotEllipse ( PlotId          plotId,  
                                      PlotEllipse *      ellip,  
                                      PlotAnnotationId * id  
                                    )
```

Draws ellipse annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **ellip** Pointer to structure defining the ellipse.
- [out] **id** Pointer to annotation to save the id of the ellipse annotation.

#### Returns

true if successful, false otherwise.

```
ALTIA_BOOLEAN altiaExDrawPlotLine ( PlotId          plotId,  
                                     PlotLineSegment * line,  
                                     PlotAnnotationId * id  
                                   )
```

Draws line annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **line** Pointer to structure defining the line.
- [out] **id** Pointer to annotation to save the id of the line annotation.

#### Returns

true if successful, false otherwise.

```
ALTIA_BOOLEAN altiaExDrawPlotPoint ( PlotId          plotId,  
                                      PlotPixel *        point,  
                                      PlotAnnotationId * id  
                                    )
```

Draws one-pixel annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **point** Pointer to structure defining the point.
- [out] **id** Pointer to annotation to save the id of the point annotation.

#### Returns

true if successful, false otherwise.

```
ALTIA_BOOLEAN altiaExDrawPlotPolygon ( PlotId          plotId,  
                                      PlotPolygon *      poly,  
                                      PlotAnnotationId * id  
                                    )
```

Draws polygon annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **poly** Pointer to structure defining the polygon.
- [out] **id** Pointer to annotation to save the id of the polygon annotation.

#### Returns

true if successful, false otherwise.

```
ALTIA_BOOLEAN altiaExDrawPlotRect ( PlotId          plotId,  
                                    PlotRectangle *    rect,  
                                    PlotAnnotationId * id  
                                  )
```

Draws rectangle annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **rect** Pointer to structure defining the rectangle.
- [out] **id** Pointer to annotation to save the id of the rectangle annotation.

#### Returns

true if successful, false otherwise.

**ALTIA\_BOOLEAN altiaExDrawPlotText**

```
( PlotId          plotId,
    PlotText *      txt,
    PlotAnnotationId * id
)
```

Draws text annotation on a plot.

#### Parameters

- [in] **plotId** Id# for the plot to draw on.
- [in] **txt** Pointer to structure defining the text.
- [out] **id** Pointer to annotation to save the id of the text annotation.

#### Returns

true if successful, false otherwise.

**ALTIA\_BOOLEAN altiaExGetDataAtOffsetFromHead** ( PlotBuffId *buffId*,

```
        unsigned int offset,
        PlotPoint * dataOut
        unsigned int dataCount
)
```

Reads data at an offset from the head position.

Gets one or multiple points. If multiple points are requested, the read will start with the value at the desired location. Subsequent points will be read from locations farther from the head. If the tail is reached, the read will stop.

#### Parameters

- buffId** Id of the buffer to read from.
- dataOut** Pointer to location to store read data.
- dataCount** Number of PlotPoints to read.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExGetDataAtOffsetFromTail ( PlotBuffId buffId,  
                                              unsigned int offset,  
                                              PlotPoint * dataOut  
                                              unsigned int dataCount  
)
```

Reads data at an offset from the tail position.

Gets one or multiple points. If multiple points are requested, the read will start with the value at the desired location. Subsequent points will be read from locations farther from the tail. If the head is reached, the read will stop.

#### Parameters

- buffId** Id of the buffer to read from.
- dataOut** Pointer to location to store read data.
- dataCount** Number of PlotPoints to read.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExInsertPlotDataAtHead ( PlotBuffId buffId,  
                                              PlotPoint * dataIn  
                                              unsigned int dataCount  
)
```

Puts data into a plot buffer at the head position.

Can insert one or multiple points. If multiple points are inserted, the insertion will start with the value pointed to by *dataIn*. Subsequent points beyond *dataIn*, will be inserted at the new head. Thus, the last point pointed to by *dataIn*[*dataCount*-1] will finally be the buffer's head.

#### Parameters

- buffId** Id of the buffer to insert data into.
- dataIn** Pointer to data to insert in the buffer.
- dataCount** Number of PlotPoints to insert.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExInsertPlotDataAtTail ( PlotBuffId buffId,  
                                         PlotPoint * dataIn  
                                         unsigned int dataCount  
                                         )
```

Puts data into a plot buffer at the tail position.

Can insert one or multiple points. If multiple points are inserted, the insertion will start with the value pointed to by *dataIn*. Subsequent points beyond *dataIn*, will be inserted at the new tail. Thus, the last point pointed to by *dataIn*[*dataCount*-1] will finally be the buffer's tail.

#### Parameters

**buffId** Id of the buffer to insert data into.

**dataIn** Pointer to data to insert in the buffer.

**dataCount** Number of PlotPoints to insert.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExPlotRegisterDataBuffer ( PlotBuffId buffId,  
                                         PlotId plotId,  
                                         PlotLineId line  
                                         )
```

Registers a circular buffer with a plot.

Registering a buffer with a plot allows the plot to scroll back through larger data-sets.

#### Parameters

**buffId** Id of the plot circular buffer to register.

**plotId** Id of the plot object to register the buffer with.

**line** id of the plot line on the plot object the buffer should correspond to.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExPlotUnregisterDataBuffer ( PlotBuffId buffId,
                          PlotId      plotId,
                          PlotLineId  line
)
```

Registers a circular buffer with a plot.

#### Parameters

**buffId** Id of the plot circular buffer to unregister.

**plotId** Id of the plot object to unregister the buffer with.

**line** id of the plot line on the plot object the buffer should correspond to.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExPlotUpdate                          ( PlotId plotId )
```

Informs plot object that it has been updated and it should redraw.

This should be called when the user wants to tell the plot object that a plot buffer has new data in it and the plot object should update its buffer pointers. Also any addition of annotations will not be drawn until this function is called.

#### Parameters

**plotId** Id for the plot to update

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExPreloadPlotWithBuffData ( PlotId plotId,
                                                          buffId
)
```

Preloads a plot's internal buffer with data from the indicated plot buffer.

#### Parameters

**plotId** Id of the plot to update the internal buffer on.

**buffId** Id of the buffer to preload.

#### Returns

TRUE if successful, FALSE otherwise.

```
ALTIA_BOOLEAN altiaExRemovePlotAnnotation ( PlotId          plotId,  
                                         PlotAnnotationId id  
                                       )
```

Removes an annotation from a plot.

#### Parameters

- [in] **plotId** Identifies the plot to remove the annotation from.  
[in] **id** Identifies the annotation to remove (was provided when the annotation was created).

#### Returns

TRUE if successful, FALSE otherwise.

## Using Fixed Point Numbers with Multi Plot

In the Altia Design installation folder, there is a Multi Plot demonstration .dsn and dll that demonstrates the use of both the standard animation interface as well as the PlotEx interface.

**IMPORTANT:** **Fixed point is only used with Altia DeepScreen Targets that are configured for fixed point operation. All other implementations use floating point (including interfacing with the Altia Design editor). By using the PLOT\_DOUBLE() and PLOT\_INTEGER() macros, your code will be portable across both fixed and floating point configurations, as well as for the Altia Design Editor, Altia Runtime Engine, and Altia DeepScreen Targets.**

In order to implement fixed point in your Design with Altia DeepScreen, you must:

1. Generate code with one of Altia's fixed point DeepScreen Targets.
2. Use the Multiplot Object external APIs
3. Employ the **PLOT\_DOUBLE()** and/or **PLOT\_INTEGER()** macros for every case in your code where you send double and/or integer plot data, respectively, to the multiplot object.

The fixed point targets use pure integers for the Altia APIs (i.e. altiaSendEvent). It is not possible to send fixed point values into the Multiplot object through the animation APIs. In addition, due to the nature of fixed point, the integer values have a range of -262144 to +262143. Values outside this range will overflow the internal fixed point math resulting in bad plot results.

The MultiPlot Demo included with Altia Design ([AltiaDesign Install Dir]\drawingarea\demos\MultiPlot\MultiPlotDemo\_DLL\src) is an example of how to implement fixed point. You can study its source code, especially plotExTest.c, to understand how this is accomplished.

Be aware that some Altia Library objects will not function properly with fixed point numbers. Others will only function properly if their mode is set to integer rather than decimal. Additionally, if your Control Code in your Design employs floating point expressions and you are attempting to implement fixed point, then a conflict will result when the DeepScreen version of your Design is loaded for execution, because floating point assets will load along with your Design.

## **Multi Plot Examples**

In the Altia Design installation folder, there is a Multi Plot demonstration .dsn and dll that demonstrates the use of both the standard animation interface as well as the PlotEx interface.

This example material can be found at the following path for standard Altia installations:

C:\usr\altiaXXX\drawingarea\demos\MultiPlot\

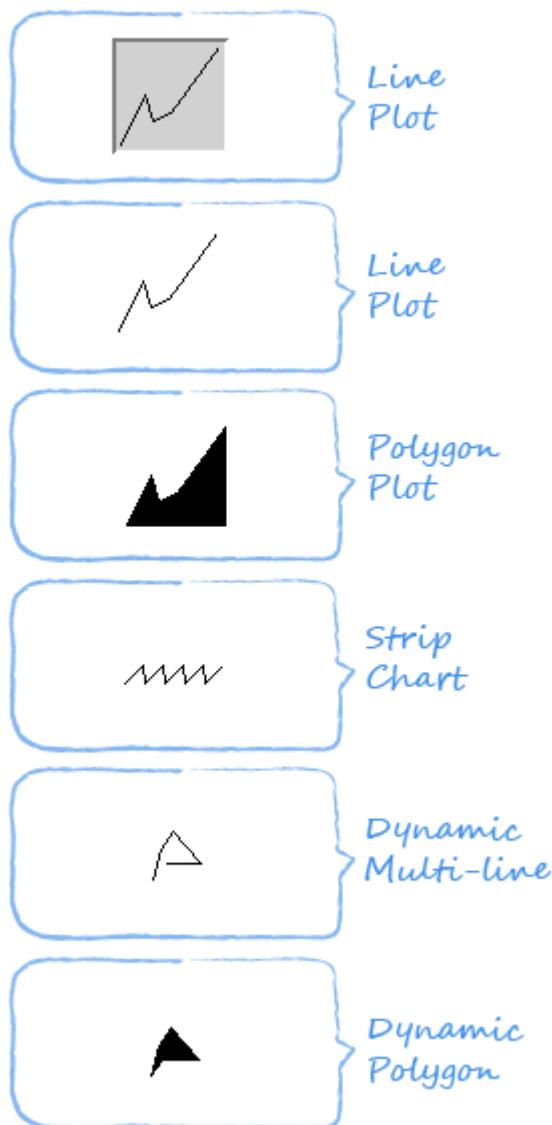
Where XXX is your Altia Design version number (ex. altia111).

## 10.5.10 Plot Object Models

Plots and Charts/Plot.dsn

The file `plot.dsn` contains several primitive line and polygon drawing objects which have built-in animations that facilitate the dynamic plotting and charting of data in line or polygon form. The standard `stripcharts.dsn` library already contains a very flexible strip chart component with properties and connections. Objects in `plot.dsn` are somewhat primitive objects.

Contained in `plot.dsn` are: a **line plot object**; a **filled plot object**; a **strip chart object**; and **dynamic multi-line, polygon, and filled polygon objects**.



To add a Plot Object to your design, from the Models View window click on it and drag it into the design's work area.

You can change the color, pattern, or line style of any plot object. You can also move it, stretch it, etc. To do these operations, put the Graphics Editor into Edit mode and click on the object to select it. Choose the tool or command option you wish to execute, and proceed as you would with any object.

Plot and strip chart objects have explicit widths and heights and can be drawn only in their defined areas. The plot.dsn file provides rows of line plot and strip chart objects in different sizes. The first two models in each row are line plot objects; the next is a filled plot object; and the last is a strip chart. After copying one of these objects to your design, you can change the size and plotting resolution by changing the state values for **setwidth** and **setheight**. Stretching or scaling the plot object will change only its relative size, not its plotting resolution. If the plot object is not stretched or scaled, resolution points are equivalent to screen pixels.

The heights and widths of dynamic multi-line, polygon, and filled polygon objects automatically change to accommodate their data points. Since these objects have no specific width or height settings, plot.dsn provides only one model of each.

## Line and Filled Plot Objects

The following seven animations are associated with line and filled plot objects.

<b>nextx</b>	The state value for nextx corresponds to the horizontal x value (in pixels) of the next point to be plotted.
<b>nexty</b>	The state value for nexty corresponds to the vertical y value (in pixels) of the next point to be plotted.
<b>draw</b>	This animation draws a line from the last point charted to the point designated by the state values of nextx and nexty. When the animation state value of draw is set to any value, the animation will be executed, as long as new values for nextx and nexty have been explicitly set.
<b>clear</b>	This animation erases the current plot.
<b>rewind</b>	Use this animation to move back to a previous point in the list of points. Set the state value of rewind to the number of the point to which you wish to rewind. This list of points is 0-based. For example, setting the state value of rewind to 5 rewinds to the sixth point in the current point list. New x,y pairs will overwrite current x,y points starting at the rewind point.
<b>setwidth</b>	The state value of setwidth corresponds to the width of the plot object as measured in pixels. The state of

this animation is usually adjusted from the Animation Editor during design editing. It can also be changed dynamically by a client program, control block, or stimulus.

<b>setheight</b>	The state value of setheight corresponds to the height of the plot object as measured in pixels. The state of this animation is usually adjusted from the Animation Editor during design editing. It can also be changed dynamically by a client program, control block, or stimulus.
------------------	---

After copying a line or fill plot object to the drawing area of the Graphics Editor, you can change its width, height, or clear the current data by selecting **setWidth**, **setheight**, or **clear** from within the Animation Editor and changing the animation's value. The state of the **clear** animation just needs to be set to do a clear.

A line plot object is derived from the multi-line object that can be created using the Graphics Editor. As a result, you can change a line plot object's color or brush just as if it were a normal multi-line object. You can also manipulate it with move, rotate, etc., and give it animation beyond its built-in capabilities.

A fill plot object is derived from the fill polygon object. It creates a plot with an area filled by a color/pattern combination like the filled polygons created using the Graphics Editor. If the first point sent via the **nextx** and **nexty** animations matches the last point prior to a draw, a closed polygon will be drawn. If not, the object will fill the area between the plot line and the bottom of the plot area (that is, the horizontal axis at  $y=0$ ) to create a closed polygon.

The first column of line plots in `plot.dsn` have a shadowed background grouped with each line plot. The colors of the background and shadows are changed by selecting a new background color from the color picker. The attributes of the line plot object itself are changed by focusing into the group, selecting the plot object, and then making an attribute change (for example, new line style, new foreground color).

## Strip Chart Object

The strip chart object in `plots.dsn` is a rather primitive object in comparison to the strip chart in the standard `stripcharts.dsn` library.

The strip chart object differs from the line and filled plot objects in that the x coordinate is a constant. For each y coordinate entered, the strip chart scrolls horizontally by a set amount. Therefore, the strip chart object has no `nextx` animation. It contains five animations that behave in the same ways as the animations for line plot objects. These animations are: `nexty`, `clear`, `draw`, `setWidth`, and `setheight`. In addition, the strip chart object contains the following two animations:

<b>setxinc</b>	The state value for this animation sets the increment constant for future x coordinates. For example, if setxinc has a <b>state</b> value of 5, the x coordinates 0, 5, 10, 15, ... will be used in sequence for new <b>nexxy</b> values.
<b>setright</b>	This animation sets the strip chart's direction. If its state value is 0, existing points move to the left as new points are added to the right (the default). If its state value is non-zero, existing points move to the right as new points are added to the left.

## Dynamic Multi-Line and Polygon Objects

Dynamic multi-line, polygon, and filled polygon objects are similar to line plot objects, but their widths and heights adjust according to data points received. When the state of the **clear** animation is set, it clears all data points and sets an object's size to 15 x 15 pixels, with the lower left corner of the object representing the origin. All x and y values represent pixel distances from the origin. Negative x and y values represent data points to the left or below the origin. All dynamic multi-line, polygon, and filled polygon objects contain five animations: **nextx**, **nexxy**, **draw**, **clear**, and **rewind**. They behave in the same ways as the animations for line and filled plot objects, except that **nextx** and **nexxy** values are not bound by a specific width or height.

## 10.5.11 Pie Object Model

Plots and Charts/Pie.dsn

This library contains pie object models that have built-in animations, including support for multiple “slices.” Each slice can have a specific span, color, and a text label. Attributes such as the number of slices and the pie object’s radius can be changed dynamically by using control statements or client application code.

To add a pie object to your design, click on the model and drag it into your design’s work area. Altia will automatically rename the animations on the pie object so that multiple pie objects will not interfere with each other.

Animations associated with pie object modes are: **label**, **color**, **percent**, **set**, **draw**, **delete**, and **radius**. The state value of **set** determines which of the slices you wish to change. Use **label**, **color**, **percent** and **delete** to set the attributes of the slice. Then use **draw** to execute the changes. As an example, the following control code example sets the label and percent for slice 2 and deletes slice 3.

```
WHEN altiaInitDesign == {}  
SET set 2  
SET span 30  
SET label "A"  
SET delete 3  
SET draw 0
```

## Pie Object Animations

### set

To change an existing slice’s **label**, **color**, or **percent** (span) or to add a new slice, **set** is used to identify the number of the slice you wish to change. The identifying number for a slice can be any integer value.

**NOTE:** The default state for set is 0. Any label, color, or span changes made without setting the set animation will affect the number 0 slice. A slice with a greater identifying number covers all slices with lower numbers if they happen to overlap.

### **label**

To add a text label to a slice, set the state of the slice you wish to work with, then send a string to **label**. To clear an existing label, send the text string " " (a string containing a single space character). A label change will not show on the display until the **draw** animation state is set (to any value). A pie object's foreground color is used for every label. **Labels** for different slices cannot be shown in different colors.

### **color**

Set a slice's color by sending a string of states for **color** that give in text form the color's name (a literal color name, a decimal RGB color specification, or a hex RGB color specification).

- A literal color name would be "red" or "white". On Windows systems, recognizable names are defined in the file \usr\altia\bin\colors.ali.
- A decimal RGB color is a string of the form "X Y Z" where X, Y, and Z are the intensity values for red, green, and blue. For example, "0 0 255" is the color blue.
- A hex RGB color specification is a string of the form "#XXYYZZ", where XX, YY, and ZZ are 2-digit hex numbers representing the intensities for red, green, and blue respectively. For example, "#0000ff" is the color blue.

A color change will not show on the display until the state of **draw** is set to any value. This is also true for **label** and **percent** changes. If color for a particular slice is never set, then the slice is drawn using the current background color for the pie object.

### **percent**

The **percent** animation is set to define the span for a slice as a percentage from 0 to 360 degrees. For example, a **percent** state value of 25 would yield a span of 90 degrees. The slice with the lowest identifying number starts at the angle 0 origin (3 o'clock) and spreads counter-clockwise. Additional slices start where the previous slice stops and also spread in the counter-clockwise direction.

As an example, the following set of control statements would set the span for slice 2 to 30 and its label to A:

```
SET set 2
SET span 30
SET label "A"
SET draw 0
```

The following calls in a client application program would do the same:

```
AtSendEvent(id, "set", 2);
AtSendEvent(id, "span", 30);
AtSendText(id, "label", "A");
AtSendEvent(id, "draw", 0);
```

If the total percentage for all slices exceeds 100, then the slices with higher identifying numbers will spread over the lowest slices. If an individual slice's **percent** animation is set to a value less than 0 or greater than 100, the slice will have a span of 0 degrees.

To change the origin for the first slice, simply rotate the pie object using the **Rotate** tool from the Tool pane, or select **Transform / Precise Rotate** on the **Home** ribbon. To change the spanning direction, scale the object horizontally or vertically by selecting **Flip Horizontal** or **Flip Vertical** from the Home ribbon's **Transform** dropdown menu.

#### **draw**

Before **label**, **color**, **percent**, or **delete** state changes can affect the appearance of a pie object, the state value of **draw** must be set to any value.

#### **delete**

To remove a specific slice from a pie object, set the state of **delete** to the slice's identifying number and then set the state of **draw** to any value.

#### **radius**

A new state value for **radius** changes the radius for all slices in the pie object, and it defines the new radius in pixel units. The radius changes immediately. It does not require the setting of **draw** as does **label**, **color**, **percent**, and **delete**.

## **Initializing Animations of a Pie Object**

The various animations for a pie object are initialized using the Animation Editor. Select the object so that its animation names appear in the Animation Editor's List Area. Highlight the animation name whose state you wish to change. The name will appear in the **Animation** field of the Animation Editor. In the **State** field, type in a new value or toggle the increment/decrement arrows.

For **label**, **color**, and **percent**, you must first adjust the state of **set** to identify the slice that should be changed. For these same animations, and with **delete**, you must finish by setting the state of **draw**. Otherwise, the changes will not be seen.

For **label** and **color**, it is possible to enter an entire text string. In the **State** field, enter your string delimited by double quotation marks. For instance, to set the **label** animation to the string **XYZ**, you would enter the following:

Animation: **label**      State: "XYZ"

As soon as you press the Enter key while in the **State** field or click the mouse cursor outside of the field, the text in the delimited string will be sent in sequence to **label**.

When you save a design to a file, the current states for the various animations are also saved.

## Pie Examples in more/pie.dsn

- **Primitive pie**

This is a pie object with no additions. It currently has only one slice - slice 0 - which appears in the Background color last selected for the object from the Graphics Editor's color picker. The color can be changed by selecting a new background color for the object after dragging it into the drawing area from a Models View. This pie can be expanded into a multi-slice pie with labels and multiple colors by setting the various animations as summarized earlier.

- **Sweeping seconds pie**

This pie is much like the primitive pie example in that it has a single slice - slice 0 - which appears in the Background color for the object. In addition, however, it has control and timer stimulus that allows it to change the state of its percent animation at 1 second intervals. The buttons next to the object can be used in run mode to start and stop the pie's timer stimulus and execute one of its **WHEN** control blocks to reset the pie slice.

To view the control for this pie, drag it from a Models View into the drawing area. With the object selected, press the **Control** button along the top of the editor to open the Control Editor. To see the object's timer stimulus, press the **Stimulate** button to open the Stimulus Editor.

- **3 slice pie**

This pie object has interesting control added so it is easy to set the span of its 3 slices, enable/disable labels, and enable/disable a background. The various sliders and switches can be copied with the object to exercise its capabilities in run mode. This example demonstrates how control can be used to uniquely manage a pie's behavior.

To view the control for this pie, drag it from a Models View into the drawing area. With the object selected, press the Control button along the top of the editor to open the Control Editor.

## 10.5.12 Skin Object Model

Special Functions/Skin.dsn

The file skin.dsn contains a model of a Skin object, which is a special object that allows users to reconfigure portions of a design at runtime using a skin configuration file. The skin configuration file allows the following runtime changes to a design:

- **Object Color (both foreground and background)**
- **Object position and anchor**
- **Object font**
- **Object line width size (sometimes also referred to as the brush or stroke width)**
- **Object image name (only for Image objects)**

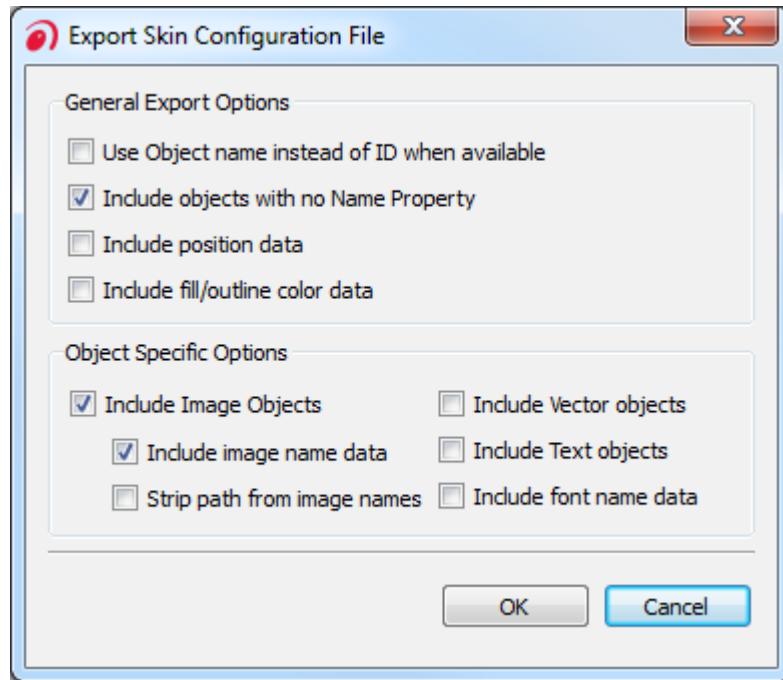
Objects can be identified by **Object ID** or by the object's **Object Name** property. Using the **Object Name** property allows many objects to be changed using a single command in the skin configuration file.

One or more Skin objects can be used in a design. Each Skin object uses animations to store important information about the skin including the skin configuration file name and the skin folder used to store all images used with the skin. A separate animation is used to trigger the execution of the skin configuration which will change the appearance of the objects in the Altia Design.

Altia Design provides an export feature which allows quick creation of skin configuration files using an existing design file as a template.

## Exporting a Skin Configuration File

1. From within Altia Design, select one or more objects that will be included in the skin. If no objects are selected, the entire design (all objects) will be used in the skin configuration file.
2. Start the skin export using the **File** ribbon's **Export/Configuration Files/Skin File** option. The following dialog will appear:



3. Select the desired export options. The options are described in the following table:

Use object Name instead of ID when available	Uses the object's <b>Object Name</b> property to identify objects in the skin configuration file. This is more portable than using an object's ID number, since the ID number will change if the object is copied and pasted, or if the object is imported into a design from a model library. If multiple objects have the same <b>Object Name</b> property value, the first object with that name will be used to create the token information for the named group (ex. position, color, etc).
Include objects with no Name Property	Includes all objects that do not have an <b>Object Name</b> property. Unchecking this option provides an efficient way to filter out unnamed objects not desired in the skin configuration file. This option is checked by default.

Include position data	Includes the <b>_POSX</b> and <b>_POSY</b> tokens in the skin configuration file.
Include fill/outline color data	Include the <b>_FG</b> , <b>_BG</b> , and <b>_BRUSH</b> tokens in the skin configuration file.
Include image objects	Include all Image objects in the skin configuration file. This option is checked by default.
Include image name data	Include the <b>_IMAGE</b> token in the skin configuration file.
Strip paths from image names	For all Image objects exported in the skin file, the path will be stripped, leaving only the image file name in the exported data. This is useful if all images are located in the same folder and the folder is specified in the Skin object <b>Image Path</b> property.
Include vector objects	Include all vector objects such as lines, rectangles, splines, and polygons in the skin configuration file.
Include text objects	Include all static text and Text I/O objects in the skin configuration file.
Include font name data	Include the <b>_FONT</b> token in the skin configuration file.

4. Press the **OK** button and select a name and location for the skin configuration file.
5. Open the skin configuration file in any text editor and make desired modifications to the skin data as necessary. Even with no modifications, the skin configuration file is directly usable by the Skin object.

**NOTE:** Continue reading for guidelines on runtime performance when using the Skin object. It's important to modify the skin configuration file created during the export process. The exported configuration file contains details on every object selected (or all objects in the design if nothing is selected). This is a lot of data and will result in unnecessarily slower runtime performance if only a few objects are actually changed by the skin configuration file.

**NOTE:** By default, the position on container objects is NOT exported. This is because the position of a container object is determined by the position of all its children. Repositioning the container will have mixed results if the children are also repositioned in the same skin file. Sometimes it's desirable to move a group instead of the children. For this case, hand-edit the skin file in order to remove the children and add a single parent.

**NOTE:** Using the skin file to set position on objects which have a position-base animation will cause conflicts at run time. For example, an object that moves as a result of an animation. The animation and the Skin object will attempt to set the object's position. For this case it's important to set the specific animation to a known state before loading a skin. The known state should be the same animation state as when the skin was exported.

## Skin Configuration File Syntax

The skin configuration file is a token based command file. Each line in the file represents one command to be executed by the Skin object. A command can affect a single object (using the Object ID) or multiple objects (using the **Object Name** property).

Each command is comprised of tokens and their associated values (in double-quotes). The following tokens are allowed in the configuration file:

**\_OBJ** The ID of the object affected by this command.

**\_NAME** The **Object Name** of all objects affected by this command.

**\_POSX** The new desired absolute horizontal position for the object(s).

**\_POSY** The new desired absolute vertical position for the object(s).

**\_ANCHOR** Specifies anchor point which will be set to the **\_POSX** and **\_POSY** position on the screen. Allowed values are **BOTTOMLEFT**, **BOTTOMCENTER**, **BOTTOMRIGHT**, **MIDDLELEFT**, **MIDDLECENTER**, **MIDDLERIGHT**, **TOPLEFT**, **TOPCENTER**, and **TOPRIGHT**. If the **\_ANCHOR** property is excluded, the default value of **BOTTOMLEFT** will be used.

<u>FG</u>	The new foreground color for the object(s).
<u>BG</u>	The new background color for the object(s).
<u>BRUSH</u>	The new line size in pixels for the object(s).
<u>IMAGE</u>	The new image file name for the object(s).
<u>FONT</u>	The new font for the object(s).

Non-relevant data for an object is ignored. For example, the IMAGE token does not have any effect on vector or text objects. It only has meaning when assigned to Image objects. Similarly, the FONT token will only have an effect on text objects.

## An Example of a Skin Configuration File

```
# Altia Skin Data File
_OBJ="4" _IMAGE="image_002.png"
_NAME="Menu Text" _FG="#ffffffff" _FONT="*-helvetica-bold-r-
normal-*-*180-*"
_OBJC="2" _POSX="535" _POSY="404" _BRUSH="0"
_NAME="Cursors" _FG="#ff0000" _BG="#000000" _BRUSH="2"
_OBJC="78" _POSX="512" _POSY="332"
_OBJC="74" _POSX="359" _POSY="464" _FG="#ff3300"
_BG="#ffffffff" _BRUSH="1"
_OBJC="75" _POSX="525" _POSY="550" _FG="#00ff99"
_BG="#ffffffff" _BRUSH="1"
_OBJC="76" _POSX="417" _POSY="546" _FG="#99cc66"
_BG="#ffffffff" _BRUSH="1"
```

## Special Notes About Skin File Construction

The “#” character can be used to start a comment line in a skin configuration file. Comment lines will be ignored by the Skin object at runtime.

Token values must be enclosed within double-quotes.

No spaces can exist between a token, the “=” sign, and the first double-quote symbol.

## Skin Object Animations

The skin object has the following animations and it has properties and connections for these animations.

<b>skin_file_name</b>	The name of the skin configuration file. This name may include a relative or absolute file path.
<b>skin_image_path</b>	The absolute or relative path used to find the images referenced in the skin configuration file. All image names will be prefixed with this image path prior to setting the <b>image_name</b> animation for the associated Image objects. A "/" symbol between the path and the file name will be inserted automatically. This animation is initially set as an empty string. It is not possible to reset the animation back to an empty string. In that case, import a new Skin object from the Skin object library or change the path animation to a value of "." (single period symbol). The single period symbol will result in all images being prefixed with the "./" path.
<b>skin_file_load</b>	Triggers the loading and execution of the skin configuration file. Write a non-zero value to this animation to start the loading of the chosen skin configuration file.

## Using the Skin Object with DeepScreen

Two options are available for the Skin object when generating code using Altia DeepScreen. First, you can choose to generate code for the skin configuration files (this is the default action). Secondly, you can choose to load skin configuration files from the file system at runtime.

By default, DeepScreen will convert the skin configuration file for each Skin object into a data structure in **skinData.c**. This is the most efficient approach for runtime performance and resource utilization.

**NOTE:** **Image files referenced by the skin configuration file will always be loaded from the file system at runtime.**

When DeepScreen auto-generates skin data into **skinData.c**, it uses the currently referenced skin configuration file for each Skin object in the design. A separate Skin object is required for each skin used by the design. This also requires that all possible skins be known at code generation time.

Optionally, the same skin configuration files used in Altia Design can be used at runtime. To use the skin configuration file directly, select the **Skin Object use files** check-box from the **Code Generation Options** dialog window. For this usage, the skin configuration file can be changed at runtime by sending the new file and path to the **skin\_file\_name** animation. This allows for a single Skin object to be used by the design which can load any number of skins at runtime.

Regardless of code generation options, there are some important points to consider when using the Skin object with DeepScreen.

Firstly, if the skin configuration file references a font, the font must already exist in the design file. DeepScreen will try and resolve the font names in the skin configuration file at runtime. The font must exist in the DeepScreen generated data files. For this to occur, at least one Text I/O or Multi-Line Text object must exist in the design with the specified font. Otherwise, the affected text will not be drawn with the new font.

Secondly, if the skin configuration file references a brush, the brush must already exist in the design. DeepScreen will try and resolve the brush line-width in the configuration file at runtime. A brush must already exist in the DeepScreen generated data files with the same line-width. Otherwise the affected lines will not be drawn with the new brush. A simple way to accomplish this is to make a “palette” group in your .dsn file. The group can be hidden so it will never be drawn at runtime. The group should contain multiple line and Text I/O objects - each object with a possible brush or font that will be used by the skins at runtime. This will force DeepScreen to add the brushes and fonts to the auto-generated data files.

Finally, the path in the **skin\_image\_path** animation will be prefixed to each image file name when the skin is loaded using the **skin\_file\_load** animation. Altia will insert a “/” symbol between the skin image path and the skin image filename.

## Performance Considerations

DeepScreen will execute all commands in a skin file, even if the command does not result in a visible change to the object. For example, setting a brush for an image or text object does not have an effect – however the command will still be processed. Similarly, setting the font for a vector object will not have a visible affect, however the command will still be processed. Setting position data on objects results in some processing even if the new (x,y) coordinates do not result in a new position for the object. The object will not be redrawn, but processing time was spent comparing new and old positions.

To improve performance, keep skin configuration files as compact as possible. Only put commands in the skin file that will result in an appearance change. Make sure to review the skin configuration file exported by Altia Design using the **File...-> Export...** menu.

## Using Skins with the Image Object

It is possible to change the image referenced by an Image object using the Skin object. This is a desirable feature of the Skin object. There are two modes of operation for the Image object when the **image\_name** animation is changed.

By default, the Image object will load and process the image file as soon as the `image_name` is changed. This means each `_IMAGE` token in the skin file will result in a synchronous file load as the skin configuration is executed.

Optionally, the Image object will load and process the image file when the Image object first becomes visible after the `image_name` is changed. This minimizes the amount of image loading that occurs when a skin configuration file is executed. Images will only be loaded for those Image objects that are visible at the time the skin configuration file is executed.

To enable the optional behavior mentioned above, use the `NO_PRELOAD` compile time definition for Altia DeepScreen generated code. Refer to the *Altia DeepScreen User's Guide* for further details on setting custom target definitions at compile time (using `TARGETDEFS`).

## 10.5.13 Snapshot Object Model

Special Functions/Snapshot.dsn

The file `snapshot.dsn` contains a model of a Snapshot object, which is a special image object that behaves much like camera. Using a Snapshot object, you can capture screen content within your `.dsn` file and use it as an image. With the properties of the Snapshot object, you can specify whether you want to capture a specific region of the screen, whatever is behind the actual Snapshot object at the time of capture, or simply capture the contents of an object by capturing a snapshot using the desired object's ID. Once a Snapshot object has captured its image, it may be animated as with any other Altia Design graphic object. The captured image is not stored permanently along with the rest of your design.

A typical use of the Snapshot object is in the creation of fluid screen transitions. Current objects on a screen may be captured into a Snapshot object. Those screen objects may then be hidden while the captured image is moved off screen or dissolved to become the next set of objects. Off-screen objects may be captured into a Snapshot object that is also positioned off screen then moved into position on screen. To do this with "real" objects may take a substantial amount of resources. With the Snapshot object, sophisticated screen transitions are possible.

### Snapshot Object Properties

The following table shows the properties and their equivalent animations used by the Snapshot object.

Property	Animation	Description
Do Capture	<code>capture</code>	<ul style="list-style-type: none"><li>• Set to <b>Behind</b> (1) to capture the region behind the Snapshot object. This region matches the location, width, and height of the actual Snapshot object.</li><li>• Set to <b>Location</b> (2) to capture a specific region of the screen.</li><li>• Set to <b>Object</b> (3) to capture a specific object via the object ID.</li></ul>
Source X Coordinate	<code>srcX</code>	Set to the X coordinate of the source region when using <code>capture = Location</code>
Source Y Coordinate	<code>srcY</code>	Set to the Y coordinate of the source region when using <code>capture = Location</code>
Source Width	<code>srcW</code>	<ul style="list-style-type: none"><li>• Set to the width of the source region when using <code>capture =</code></li></ul>

		<b>Location</b>
		<ul style="list-style-type: none"> <li>Set to the width of the region behind the Snapshot object when using <b>capture = Behind</b></li> <li>Automatically set by the Snapshot object to the width of the object being captured when using <b>capture = Object</b></li> </ul>
<b>Source Height</b>	<b>srcH</b>	<ul style="list-style-type: none"> <li>Set to the height of the source region when using <b>capture = Location</b></li> <li>Set to the height of the region behind the Snapshot object when using <b>capture = Behind</b></li> <li>Automatically set by the Snapshot object to the height of the object being captured when using <b>capture = Object</b></li> </ul>
<b>Source Obj ID</b>	<b>srcObjId</b>	<ul style="list-style-type: none"> <li>Set to the object's ID when using <b>capture = Object</b></li> <li>Set to -1 to clear the Snapshot object's image data when using <b>capture = Object</b></li> </ul>
<b>Alpha Mode</b>	<b>alphaMode</b>	<ul style="list-style-type: none"> <li>Set to <b>Opaque</b> (0) when capturing objects that will fill the entire Snapshot object or for objects with the same background color as that of the design. Objects with transparency will not be captured correctly in this mode. This only applies to hardware accelerated DeepScreen targets.</li> <li>Set to <b>Transparent</b> (1) when capturing objects with non-rectangular shapes or transparency. This only applies to hardware accelerated DeepScreen targets.</li> </ul>

- Set to **Software** (2) when running in Altia Design, under Windows, or in an altiaGL DeepScreen target.

**Source Layer**

**srcLayer**

- Set to the layer number to capture.
- Set to -1 to capture all layers.

**NOTE:** All of the above animations can easily be set using the properties of the Snapshot object. However, you can also set them dynamically via code for interesting effects and for making animated transitions.

## Capturing an Image Behind the Snapshot Object

1. In Edit Mode, on the **Insert** ribbon, select the **Snapshot** library from the Model Library gallery dropdown.
2. Drag the Snapshot object into your design and position the lower left corner as desired. Notice that while in Edit Mode, a Snapshot object with no content is drawn as a wire frame. In Run Mode or DeepScreen, an empty Snapshot object is not drawn at all.
3. Select the Snapshot object and its properties will appear in the **Properties** pane.
4. Set the **Source Width** and **Source Height** properties as desired.
5. Select **Behind** from the **Do Capture** drop-down menu.

## Capturing an Object Using the Snapshot Object

1. First, select the object you want to capture and take note of its Object ID# in the status bar of the Altia Design window.
2. In Edit Mode, on the **Insert** ribbon, select the **Snapshot** library from the Model Library gallery dropdown.
3. Drag the Snapshot object into your design and position the lower left corner as desired. Notice that while in Edit Mode, a Snapshot object with no content is drawn as a wire frame. In Run Mode or DeepScreen, an empty Snapshot object is not drawn at all.
4. Select the Snapshot object and its properties will appear in the **Properties** pane.
5. Set the **Source Obj ID** property to the Object ID of the object you noted in step 1.
6. Select **Object** from the **Do Capture** drop-down menu.

## Clearing the Snapshot Object

1. Select the Snapshot object and its properties will appear in the **Properties** pane.
2. Set the **Source Obj ID** property to -1.
3. Select **Object** from the **Do Capture** drop-down menu.
4. Notice that while in Edit Mode, a Snapshot object with no content is drawn as a wire frame. In Run Mode or DeepScreen, an empty Snapshot object is not drawn at all.

## Typical Snapshot Object Uses

In run mode, it is customary to control a Snapshot object's animation from input stimulus, timer stimulus, or an application program. A Snapshot object is ideally suited for capturing portions of the screen within your .dsn file, and then animating away or fading out the snapshot object while the interface is changing behind the snapshot, thereby creating a non-destructive and easy-to-edit screen transition. However, other uses vary, and include using the snapshot object to create dynamic reflected text for Text I/O objects by capturing a snapshot of the text each time it changes and showing the new text in a mirrored, slightly translucent snapshot object below the actual Text I/O object.

## 10.5.14 Sound

Special Functions/Sounds.dsn

Altia Design is capable of playing sound files from stimulus, control, or program events. On PC's, these are formatted as .wav files and on UNIX systems, they are formatted as .au files.

The standard Sounds.dsn library contains a sound object with connections and properties already included. As of Altia Design 2.0, the sound object has been enhanced with built-in animations and capabilities for Windows systems using a Sound Blaster compatible sound card.

To play a particular sound, select the sound object in the Graphics Editor. Open the Animation pane and highlight **sound\_name** animation in the List Area (which we will refer to as the <sound> animation). This item should match the text displayed by the object. Affect the state for the <sound> animation by changing the value in the State field. The value for a sound object's <sound> may be set from stimulus, control, or a client application program.

From a control block, use a SET statement, such as:

```
SET 1_sound 1
```

## Sound Object Animations

### <sound>

Each sound object has an animation named <sound>, with <sound> representing the name of the sound (**beep** in the above SET statement). Setting the state of <sound> to any value starts the playing of the sound. If a sound is already playing, the new sound may interrupt the current sound or play after the new sound finishes, depending on sound hardware.

### soundStop

For PC Windows systems with a Sound Blaster compatible sound card, setting the state of **soundStop** to any value stops the playing of the current sound. State changes for this animation are ignored with all other types of sound hardware.

### <sound>\_done

For PC Windows systems with a Sound Blaster compatible sound card, the value of <sound>\_done is set to 1 when the <sound> sound stops playing, when it is stopped with **soundStop**, or when it is stopped because another sound is started.

A timer stimulus, control **WHEN** block, or a client application program can be made to execute when **<sound>\_done** is set.

**NOTE:** A **<sound>\_done** with state 1 is only generated for PC Windows systems with a Sound Blaster compatible sound card.

#### **soundShow**

Each sound object found in the `sound.dsn` library displays as text the name of the sound it will play. While this is helpful during the design of an interface, it may be desirable to hide the sound objects when you run your designs. This can be done by setting the state of **soundShow** to 0. Because all objects in `sound.dsn` have this animation, they will all hide their text.

#### **<sound>\_name**

The sound file played by a particular sound object can be changed dynamically by sending a string of character values to **<sound>\_name**, allowing a design to have a few actual sound objects that are changed to play any number of different sounds. When **<sound>\_name** is given a new text string, the text displayed by the object automatically updates to the new name.

The sound file referenced by the new **<sound>\_name** must be located in the current working directory, a directory named `sound` under the current working directory, or in the standard Altia sound directory.

## **The Sound Playing Process**

For PC Windows systems, the actual playing and stopping of a sound object is performed directly by Altia Design.

On PC Windows, sound hardware such as a Sound Blaster compatible card is an extra hardware component. Altia's sound object playing process will work with all sound cards that are Windows compatible. If you do not have a sound card, you can use the small speaker built into your system.

## **Adding Your Own Sounds**

Altia has provided a number of sounds for you to use. PC sounds are stored as .wav files. UNIX sounds are stored as .au files. Sounds available from other sources (including recording tools provided with your sound hardware) can also be played by Altia Design.

To add your own sounds to Altia, do the following:

1. Move the new sound file to the Altia sound directory. The Altia sound directory is typically `\usr\altia\sound`.

2. Within the Altia Design editor, copy an existing sound object from the `sound.dsn` library into your own design.
3. Edit the text in the object by selecting the object and then double-clicking on it and retyping to match your new sound name. The file name extension is not necessary if you use standard file formats—`.wav` on the PC and `.au` on UNIX.
4. From the Animation Editor's **Edit** menu, open the Rename Dialogue Box and change the current prefix for the animation names to a new prefix. For example, if you started with the `beep` object and your new sound is `goodbye.wav` (`goodbye.au` on UNIX), change the prefix `beep` to `goodbye`.

## Converting Sound Files

All UNIX systems support the SUN `.au` sound file format while the `.wav` format is most popular on the PC. UNIX users can convert between these formats and additional formats using the `/usr/altia/sound/soundconvert` program. For more details on this freely distributed program, UNIX users should read the text file `/usr/altia/sound/soundconvert.doc`.

## Creating an Altia Design Runtime Application with Sound Objects

If you are distributing your design as an Altia Runtime application and it uses sound objects, there are several ways to organize your files.

If you are setting up a hierarchy of directories, you should place the sound files, along with the `soundit` program on UNIX, in a sub-directory named `sound`.

If you are setting up a single directory containing your Altia design(s) and the Altia Runtime program, simply copy all sound files you are using to this directory. UNIX users must also copy the `soundit` program.

For more information on creating an Altia Runtime application, please refer to [Appendix C: Altia Runtime](#).

## 10.5.15 Text Input/Output Object Models

Inputs/Inputs.dsn, Text/Readouts.dsn, Text/Text IO.dsn

Text input/output objects are special objects that help a user enter text into an Altia design while it is running. They also allow a control statement or client program to display a string of text dynamically. The standard inputs.dsn and readouts.dsn libraries contain versions of text input/output objects that have connections and properties for handling text and number entry (inputs.dsn) as well as text and number displaying (readouts.dsn). The more/textio.dsn library, on the other hand, contains a primitive text input/output object and various primitive text input areas.

As of version 7.0, text I/O objects have built-in animations that allow cut, copy, and paste, keyboard shortcuts, swiping (with mouse or using keyboard shortcuts), and other features.

**NOTE:** Instances of Text I/O objects used for text entry in existing designs will get new animations as a result of these enhancements if the cursor\_mode animation still exists on them. However, stimulus must be added to activate these new animations. Text entry components from the standard Altia Design 7.0 (or later) inputs.dsn models library fully support these features and use them to implement many new behaviors.

**NOTE:** When generating DeepScreen code, Text I/O objects are optimized to reduce their memory usage. For detailed information on Text I/O memory optimization, refer to the "Optimizing Text I/O Objects" section of the DeepScreen User's Guide. There may also be additional optimization information in your target's DeepScreen Target User's Guide.

To add a text input/output object to your design, click on it and drag it into the design's work area. Changing animation names for these objects is automatically done which gives you the ability to have multiple text input/output objects working independently in a single design. To manually change names for existing objects, choose **Rename Animation...** from the Animation pane's right-click context menu, or select **Rename** from the **Home** ribbon.

You can change the font, color, outline or pattern of any text input/output object. You can also move it, stretch it, etc. To perform these operations, put the Graphics Editor into **Edit** mode and click on the object to select it. Choose the tool or command you wish to execute and proceed as you would with a normal object.

A text input/output object contains:

- Animations that output data to the screen:
  - **character**
  - **text**
  - **integer**
  - **float**
  - **clear**
- Animations that control the format of the output:
  - **justify\_mode**
  - **max\_char\_count**
  - **max\_pixel\_count**
  - **base\_mode**
  - **append\_on**
  - **clip\_on**
  - **decimal\_pts**
  - **length\_mode**
  - **length**
- Animations that control the data's displaying behavior:
  - **scroll**
  - **scroll\_on**
  - **cursor\_mode**
  - **output\_cursor**
  - **select\_on**
  - **select\_now**
  - **hilight\_color**
  - **shortcut\_on**
  - **jump\_on**

## Output Animations

- **The Text I/O character Animation**

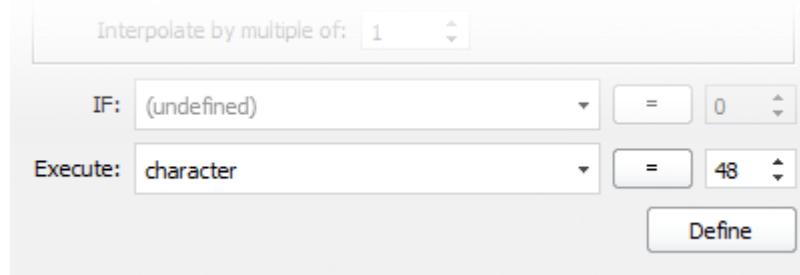
The state value of a **character** animation is interpreted as an ASCII character code. Each time **character** is given a printable ASCII state value, it is added to the ASCII state list and

the new character is added to the displayed text. A new **character** animation with a state of 0 marks the end of the current string of text. The next **character** state will erase the current text before displaying the new character. This mode of displaying a new character immediately upon receipt of the new state is referred to as non-buffered output.

As previously stated, a new **character** animation with a state of 0 marks the end of the current string. It also triggers internal analysis of the current string. If the characters of the current string are entirely representative of a number (for example, "1234" represents an integer, "0x1234" represents a hex integer, or "1234.56" or "1.23e4" represent a float), the number value is routed as an event on the **integer** and **float** animations. The value routed as an event on the **integer** animation is rounded to the nearest integer (for example, "1234.56" is rounded to 1235 or "1234.49" is rounded to 1234).

The **text** animation, by contrast, works in a buffered mode. Buffered mode allows better character throughput, but non-buffered mode is sometimes necessary for building text input areas.

The **character** animation is often used as the execute name in a stimulus definition. For example, in the following stimulus definition, the Stimulus Editor sets the **character** animation's state value to the ASCII value for the character 0 (the ASCII value for the character 0 is 48).



The **character** animation is also useful in the Control Editor's **SET** statement. In the following example, character 0 would be added to the text input/output object.

```
SET character 0
```

If the cursor is on (see **cursor\_mode** description), a -1 value sent to the **character** animation deletes the selected text. If no text is selected, the character following the cursor is deleted. If the cursor is not on, no special action is taken. A character value of -1 is not a defined character in any character set. In Unicode, 0xFFFF is explicitly guaranteed not to be a character at all so this holds true even for 16-bit DeepScreen generated code.

- **The Text I/O `text` Animation**

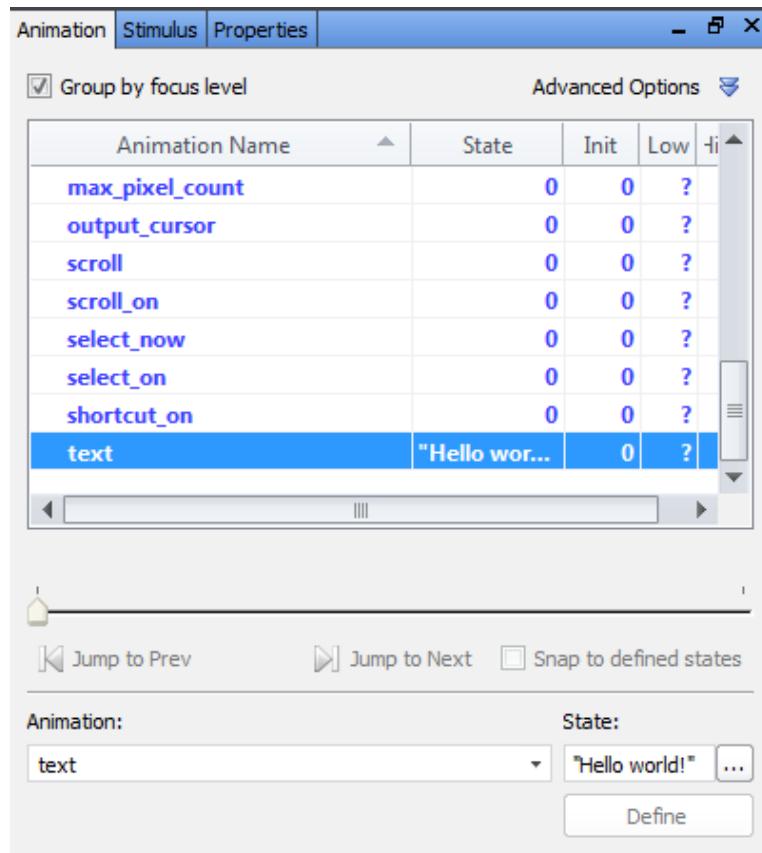
Use this animation to display a string of characters, such as a word or phrase. Like the **character** animation, the **text** animation interprets new state values as ASCII character codes. However, instead of displaying a new character immediately, the character is stored internally. When a 0 state value (which is the “end of string” identifier) is received, the characters stored internally replace the current string of text which is written to the display.

As previously stated, a new **text** animation with a state of 0 marks the “end of string”. It also triggers internal analysis of the current string. If the characters of the current string are entirely representative of a number (for example, “1234” represents an integer, “0x1234” represents a hex integer, or “1234.56” or “1.23e4” represents a float), the number value is routed as an event on the **integer** and **float** animations. The value routed as an event on the **integer** animation is rounded to the nearest integer (for example, “1234.56” is rounded to 1235 or “1234.49” is rounded to 1234).

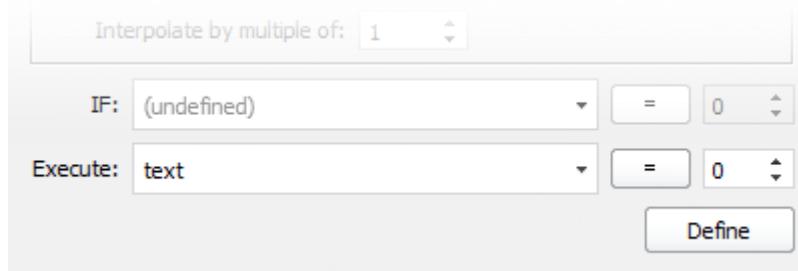
In a client program, use **text** with the **AtSendText** function to display a string of characters. For example,

```
AtSendText(id, "text", "Hello world")
```

You can use the Animation Editor to quickly view or edit a string of characters for the selected text object. Double click the string in the **text** animations **State** column's cell in the Animation List or in the **State:** text input field below to edit the text string.



To generate a single ASCII state (in this case, 0) with a stimulus definition, use an Execute statement that looks like:



With the Control Editor, use the **SET** or **EXPR** statement to display a string of characters.

```
SET text "Hello World"
```

Control or client application code can also query a **text** animation to determine the current contents of a text I/O object. This is true even if the **character**, **float**, or **integer** animations were used to update the object. For example, the control statement below will compare the current string content of the text I/O object with the string Hello World.

```
IF text == "Hello World"
```

To implement a query capability, the **text** animation handles a new state value of -1 in a special way. When it receives a new state of -1, it automatically generates a new set of **text** animation state changes. These states are the ASCII character codes for the currently displayed text ending with a state change of 0 to identify the end of the text.

When a **text** animation name is used in a control IF statement and the other operand of the **IF** statement is double-quoted text or another **text** animation name (as shown above), a -1 state for **text** is automatically generated. The resulting states are then compared against the characters of the other operand.

The client application library functions **AtGetText()** and **altiaGetText()** work in a similar fashion. They send a -1 state for the **text** animation and then listen for the returning **text** states and capture them in a character string.

A client application program queries the object by calling the library function

```
AtGetText(id, "text", char *buf, int bufSz) or  
altiaGetText("text", char *buf, int bufSz)
```

These routines return the object's currently displayed text in the caller's character buffer pointed to by **buf** whose size is given by **bufSz**. For more details, see the manual page for **AtGetText** in the [Altia API Reference Manual](#).

- **The Text I/O integer Animation**

Use this animation to display an integer. The state value of the **integer** animation is interpreted as a number and that number is displayed in the text object based on the current state of **base\_mode** and **decimal\_pts**. It can also be used in a client program. For example,

```
AtSendEvent (id, "integer", 5)
```

In a control statement, it would take the form:

```
SET integer 5
```

Such statements would set a text I/O object to the text string 5.

In addition, **integer** can be renamed to correspond to an existing animation in your design. This is useful for monitoring the state of another object's animation, stimulus, or control. For example, if you have a knob object with an animation named **knob**, you might rename **integer** to **knob** to monitor the values of **knob** as the knob is turned in Run mode.

- **The Text I/O float Animation**

Use this animation to display a floating point number. How the number is displayed depends on the current state of **base\_mode** and **decimal\_pts**.

Client programs must use the float version of the API library to send values to the float animation via **AtSendEvent()**. In Control, the **float** animation can be used like the **integer** animation. The **float** animation can also be renamed to monitor other animations.

- **The Text I/O clear Animation**

Use this name to clear the text being displayed in a design. The state value that will execute **clear** can be any integer. When the state value of **clear** is executed or changed (regardless of its value), the displayed text will be cleared.

For example, create an object stimulus, such as **LeftDown**, that sets the state of **clear** to 1. When the left mouse button is clicked on this object, a **clear** state event is executed, and the displayed text will be cleared.

## Format Animations

- **The Text I/O `justify_mode` Animation**

By default, `text`, `float`, or `integer` values are left justified in the display. For a different justification, choose `justify_mode` in the Animation Editor's List Area and assign it one of the following values:

- 0** Left justified (default).
- 1** Right justified.
- 2** If `text` has a numerical value, it will be right justified with leading zeroes. Otherwise, `text` is left justified.
- 3** Center justified.

**NOTE:** For values of 1, 2, or 3 to work, the `max_char_count` animation must be set to a value above zero. This `max_char_count` value, which specifies how many characters are allowed in the text area, helps determine text positioning. A `justify_mode` value of 1, 2, or 3 automatically disables any non-zero `cursor_mode` setting. For values of 4, 5, and 6, `max_char_count` should be set to 0 for best results.

- 4** Left justify relative to the sibling object immediately beneath the text I/O object.
- 5** Right justify relative to the sibling object immediately beneath the text I/O object.
- 6** Center relative to the sibling object immediately beneath the text I/O object.
- 7** Right justifies to a sibling object immediately beneath the text I/O object. Clips text on the left side if the Text I/O's `clip_on` animation is non-zero. If `clip_on` is 0, this justify mode behaves just like the justify mode of 5 (that is, it right justifies to a sibling object without clipping).

Use the Animation Editor to manually set `justify_mode` or set it dynamically in Run mode by using stimulus, control, or a client program.

**NOTE:** Fonts with a fixed width, such as Courier, are recommended when using justification. Variable width fonts, such as Helvetica and Times, will shift position slightly as characters are changed.

- **The Text I/O `base_mode` Animation**

By default, state changes for the integer and float animations are displayed as an integer decimal (in string format). To display numbers in hexadecimal, set the state value of `base_mode` to 1. To display numbers in octal, set `base_mode`'s state value to 2. To display in decimal point style float, use 3. To display in exponent style float, use 4.

Setting the state of `base_mode` to any other value will return the display to decimal. Use the Animation Editor to manually set `base_mode` or set it dynamically in Run mode by using stimulus, control, or a client program.

**0** Decimal integer (default)

**1** Hexadecimal integer

**2** Octal integer

**3** Float

**4** Float with exponent

For 0, 1, or 2, the `decimal_pts` animation determines the number of leading zeroes. For modes 3 and 4, the `decimal_pts` animation determines the number of digits to the right of the decimal point.

- **The Text I/O `decimal_pts` Animation**

When using the integer or float animations, the state of this animation determines the number of leading zeroes for `base_mode` 0, 1, or 2; or the number of digits after the decimal point for `base_mode` 3 or 4.

- **The Text I/O `max_char_count` Animation**

If `scroll_on` is 0, the `max_char_count` animation, when set to a value greater than 0, limits the character length of the text input/output object to the given value. If a new character, text, float, or integer state results in a string that is too long, the output is truncated to the specified number of characters.

If `scroll_on` is 1, the text scrolls after the number of visible characters reaches `max_char_count`. Therefore, when `scroll_on` is 1, it is possible for the total length of the string to grow beyond `max_char_count`.

- **The Text I/O `max_pixel_count` Animation**

Set `max_pixel_count` to a positive non-zero value to limit the visible characters of a Text I/O to the given pixel length. The visible string is the whole number of characters that does not exceed the given pixel length. The entire string is still stored in the Text I/O so `max_pixel_count` can be changed to vary the number of visible characters on-the-fly.

When `max_pixel_count` is a positive non-zero value, it takes precedence over the state of `max_char_count`.

Using `max_pixel_count` in conjunction with non-zero values for `clip_on`, `cursor_on`, `scroll`, and/or `scroll_on` is **not** supported. Using `max_pixel_count` with `justify_mode` animation states 1 through 3 (the non-sibling right and center justify modes) is also not supported.

The `max_pixel_count` animation **does** work in conjunction with the `justify_mode` animation states of 4 through 7 (the sibling justify modes).

Text I/O objects in existing designs will get a `max_pixel_count` animation automatically when they open into Altia Design or Runtime 9.0 or newer if they have a `length_mode` and `length` animation. If these animations have been deleted, a `max_pixel_count` animation is not automatically created. To add a new Text I/O object to a design that has a `max_pixel_count` animation, copy one of the Text I/O objects from the Text/text IO.dsn models library.

- **The Text I/O `append_on` Animation**

This mode of operation is quite useful for displaying text that is a combination of several existing values. When the state of the `append_on` animation is set to 1, text generated by new character, text, float, or integer values is appended to the currently displayed text. For example, take the following sequence of control statements:

```
SET append_on 1
SET integer hours
SET text ":"
SET integer minutes
```

```
SET text ":"  
SET integer seconds
```

If the states of hours, minutes, and seconds are 10, 26, and 56 respectively, the displayed text would be **10:26:56**.

- **The Text I/O clip\_on Animation**

When the **clip\_on** animation is set to a value of 1 and the text input/output object is placed in a group with other objects, it will constrain itself to the boundaries defined by the other objects in the group. Any characters that do not fit into the text input/output object will be ignored. To lengthen a text entry field, stretch the background object that is part of the input area group. Do not stretch the object itself, as this will simply change the shape of the text, not the number of allowed characters.

If the state of **max\_char\_count** is a value greater than zero, its behavior takes precedence over clipping.

- **The Text I/O length\_mode Animation**

The **length\_mode** animation controls how the length of the string contained in the text I/O object is reported. The different modes allow you to determine the width of the visible or complete text (reported in either characters or pixels). When the **length\_mode** animation is set to a valid state other than zero, the **length** animation reports the calculated length. The modes are listed below.

- 0 No length reporting will be done.
- 1 Displayed size of the text I/O in pixels (visible only).
- 2 Actual size of the text I/O in pixels (visible or not).
- 3 Displayed number of characters.
- 4 Actual number of characters.

- **The Text I/O length Animation**

The **length** animation displays the current length of the text I/O object. The length is recalculated and reported on this animation each time an event occurs that causes a change in the size of the text I/O object. The information reported on this animation is controlled by the **length\_mode** animation. The length can be reported in terms of pixels or characters and can

display the actual size of the text I/O object or just for the currently displayed size (for instance, if the text I/O object is being clipped).

## Behavioral Animations

- **The Text I/O scroll and scroll\_on Animations**

Use these animations to scroll text. Scrolling is especially useful for creating text entry areas, which are described later.

If `scroll_on` is set to 1 and `clip_on` is set to 1 or `max_char_count` is set to a value greater than 0, text will scroll right to left as new characters that would force the text to exceed its maximum length are added. It is also possible to directly scroll the text from left to right or right to left by setting the state of `scroll` to -1 or 1, respectively.

If `scroll` is set to -10000 and the cursor is on (see `cursor_mode` description), the cursor is positioned at the very beginning of the text and the text scrolls as needed to show the cursor. This behavior is similar to setting `cursor_mode` to 7. If `select_on` is set to 1 and the cursor is on, text is also selected from the initial cursor position up to and including the first character of text.

If `scroll` is set to 10000 and the cursor is on (see `cursor_mode` description), the cursor is positioned at the very end of the text and the text scrolls as needed to show the cursor. This behavior is similar to setting `cursor_mode` to 8. If `select_on` is set to 1 and the cursor is on, text is also selected from the initial cursor position up to and including the last character of text.

- **The Text I/O cursor\_mode and output\_cursor Animations**

The text input/output object supports a cursor for creating text entry/edit areas. For text output only (with no cursor), `cursor_mode` is set to a state of 0. The other listed states produce more interesting behavior:

**0** No cursor (default).

**1** A vertical cursor shown in the background color for the text input/output object appears at the beginning of the currently displayed text.

**2** A vertical cursor shown in the background color for the

text input/output object appears at a position within the displayed text determined by the current position of the system's mouse cursor.

- 3 A vertical cursor shown in the background color for the object appears at the end of the currently displayed text.
- 4 The built-in cursor is turned off and a new state value of 0 is automatically given to the **output\_cursor** animation.
- 5 The built-in cursor is turned off and a new state value is automatically given to the **output\_cursor** animation. The value is determined by the current position of the system's mouse cursor. The value is equivalent to the distance, in pixels, between the front of the text and the current mouse position.
- 6 The built-in cursor is turned off and a new state value is automatically given to the **output\_cursor** animation. The value is equivalent to the distance, in pixels, from the front of the text to the end of the displayed text.
- 7 Position the cursor at the very beginning of text (whereas state 1 positions it at the beginning of the currently visible text).
- 8 Position the cursor at the very end of text (whereas state 3 positions it at the end of the currently visible text).

State values of 1, 2, 3, 7, and 8 are used to create a text entry/edit area that has a built-in cursor. Stimulus for a left button press typically executes **cursor\_mode** state 2.

Stimulus associated with alpha-numeric key presses executes **character** state changes to allow inserting of new text in front of the cursor. The Backspace key deletes text to the left of the cursor, while stimulus for the left and right arrow keys sets the state of the **scroll** animation to -1 or 1 to move the cursor.

The 4, 5, and 6 state values automatically generate events for the **output\_cursor** animation. A custom cursor object that has its own **output\_cursor** animation can be

created using the Animation Editor. For state 0, it would position itself at the beginning of the text. For a maximum state, 500 for example, it would position itself 500 pixels to the right of the 0 state (state 500 would be initially defined by using a precise move of 500 pixels horizontally).

Example text entry areas that use the built-in cursor as well as a custom cursor are found in the more/textio.dsn models library.

- **The Text I/O `select_on` Animation**

If the `select_on` animation is set to 1 and the cursor is on (see `cursor_mode` description), characters are selected or unselected when non-zero values are sent to the `scroll` animation according to the following rules:

- `scroll < 0` (usually -1) selects characters to the left of the current cursor position or unselects them if they are already selected. The absolute value of scroll determines how many characters are selected or unselected. The cursor also moves left based on the absolute value of scroll.
- `scroll > 0` (usually 1) selects characters to the right of the current cursor position or unselects them if they are already selected. The value of scroll determines how many characters are selected or unselected. The cursor also moves right based on the value of scroll.
- `scroll = -10000` selects characters from the current cursor position to the first character of text. The cursor is positioned in front of the first character of text.
- `scroll = 10000` selects characters from the current cursor position to the last character of text. The cursor is positioned after the last character of text.

If the `select_on` animation is set to 1 and the `jump_on` animation is also set to 1 or the `shortcut_on` animation is set to 2, the behavior is similar except that the selection (or unselection) is by a word instead of a single character. For `scroll` values of -10000 or 10000, the behavior is the same independent of the state of `jump_on` or `shortcut_on`.

The `select_on` animation is typically set to 1 in stimulus for a Shift key press and set to 0 in stimulus for a Shift key release.

The `scroll` animation is typically set to -1 in stimulus for a Left Arrow key press and set to 1 in stimulus for a Right Arrow key press.

The `scroll` animation is typically set to -10000 in stimulus for a Home key press and set to 10000 in stimulus for an End key press.

- **The Text I/O `select_now` Animation**

When the cursor is on (see `cursor_mode` description), this animation takes immediate action for special selection needs (whereas the `select_on` animation modifies the behavior of future scroll animation events when the cursor is on). Set `select_now` to one of the following values:

- 1 Immediately triggers a recalculation of the selected text based on the mouse cursor position. If the mouse cursor position is to the far left (or far right) of visible text, the text automatically scrolls one character and the new visible character becomes selected. A typical scenario for setting this state is from stimulus when there is mouse motion with the left mouse button held down.
- 2 Immediately triggers a selection of all text and places the text cursor at the beginning of the text. If necessary, the text is scrolled so that the beginning of the text is showing.
- 3 Immediately triggers a selection of all text and places the text cursor at the end of the text. If necessary, the text is scrolled so that the end of the text is showing. A typical scenario for setting this state is from stimulus on detection of a double-click of the left mouse button.

- **The Text I/O `hilight_color` Animation**

This animation takes a string like the `text` animation.

For Altia Design, Altia FacePlate, or Altia Runtime, the string is a color name (for example, “red”) or an RGB value (for example, “255 0 0”).

For DeepScreen generated code, the string must be an RGB value (for example, “255 0 0”). A color name string does nothing in DeepScreen because generated code does not contain a color name lookup table.

The color given is applied to the highlight bar shown when selecting text.

- **The Text I/O `shortcut_on` Animation**

If the `shortcut_on` animation is set to 1 or 2 and the cursor is on (see `cursor_mode` description), the character animation processes the following character values as keyboard shortcut requests. The characters themselves are not added to the displayed text if they are processed as keyboard shortcut requests.

**C or c** Copy the currently selected text to the clipboard. If no text is selected, the clipboard is not changed.

**X or x** Cut the currently selected text to the clipboard. If no text is selected, the clipboard is not changed.

**V or v** Paste from the clipboard to the currently selected text. If no text is selected, the paste is performed at the current cursor position. If the cursor is not on, the result is undefined.

Setting `shortcut_on` to 2 is the same as setting `shortcut_on` to 1 and also setting `jump_on` to 1. It is a convenience feature to minimize stimulus definitions for PC style behavior where the Ctrl key enables copy/cut/paste as well as jumping by words when pressing the keyboard arrow keys.

Setting `shortcut_on` to -1 is the same as setting `shortcut_on` to 0 and also setting `jump_on` to 0.

For PC style behavior, the `shortcut_on` animation is typically set to 2 by stimulus for a Ctrl key press and set to -1 in stimulus for a Ctrl key release. To support the case where the modifier key for jump is different than the modifier key for shortcuts, using 0 and 1 for `shortcut_on` implies that jump is being driven independently and should not be modified.

The intended use is either switching `shortcut_on` between -1 and 2 (to automatically set `jump_on`); or switching `shortcut_on` between 0 and 1 if `jump_on` will be manually updated. Mixing these values (for instance, switching between 2 and 1) can cause unexpected results.

- **The Text I/O `jump_on` Animation**

If the `jump_on` animation is set to 1 and the cursor is on (see `cursor_mode` description), `scroll` events move the cursor by words instead of just single characters. A word is

interpreted as any combination of alphabet or number characters. Any other character or white space is interpreted as a break in a word.

Setting **shortcut\_on** to 2 is like setting **jump\_on** to 1 and **shortcut\_on** to 1. It is a convenience feature to minimize stimulus definitions for PC style behavior where the Ctrl key enables copy/cut/paste when simultaneously pressing the C, X, or V key as well as scrolling by words when simultaneously pressing the keyboard arrow keys.

Setting **shortcut\_on** to -1 is like setting **jump\_on** to 0 and **shortcut\_on** to 0.

For a behavior style where a certain key enables copy/cut/paste and a different key enables scrolling by words (let's call this different key the jump key), set **jump\_on** to 1 when the jump key is pressed and set it to 0 when the jump key is released.

## More on Selection and Shortcut Stimulus Definitions

See the text I/O based components in `inputs.dsn` for good examples of how to use the `select_on`, `select_now`, `shortcut_on`, and scroll animations from stimulus definitions. The components in `inputs.dsn` implement common capabilities such as:

<b>Shift+Arrow</b>	Select by character.
<b>Ctrl+Arrow</b>	Move the cursor to the next word.
<b>Shift+Ctrl+Arrow</b>	Select the next word.
<b>Home</b>	Move cursor to beginning of text.
<b>End</b>	Move cursor to end of text.
<b>Shift+Home</b>	Select to the beginning of text.
<b>Shift+End</b>	Select to the end of text.
<b>Delete</b>	Delete selected text or next character.
<b>Left Mouse Button</b>	Select all text.
<b>Double Click</b>	
<b>Left Mouse Button</b>	Select text under mouse cursor.
<b>Press and Drag</b>	
<b>Ctrl+C</b>	Copy selected text to clipboard.

**Ctrl+X** Cut selected text to clipboard.

**Ctrl+V** Paste text from clipboard.

To minimize the amount of stimulus, a few stimulus definitions trigger control **WHEN** blocks instead of directly acting on text I/O built-in animations. These **WHEN** blocks are generally very simple.

There are properties on each component associated with the **hilight\_color** animation, text I/O object foreground color (determines the text color), and text I/O object background color (determines the cursor color). These properties and others make it easy to customize the appearance of a component.

## Initializing the Text I/O Object

With the Graphics Editor in Edit mode, you can edit the display text of a text output object by simply double-clicking on the body of the text (not on any of the selection handles) when the object is selected. All tool and command buttons must be disengaged to ensure that the double-click is recognized as a text edit request.

The various animations for a text I/O object are initialized using the Animation Editor. Select the object so that its animation function names appear in the Animation Editor's **Defined STATES and FUNCTIONS:** list. Highlight the animation name whose state you wish to change. The name will appear in the **Name** field of the Animation Editor. In the **State** field, enter a new value or use the up/down arrows to increment or decrement the current value.

For the **text** animation, it is possible to enter an entire text string. In the **State** field, enter your string delimited by double-quote marks.

As soon as you press Enter while in the **State** field or press a mouse button with the mouse cursor out of the field, the text in the delimited string will be sent in sequence to the text I/O object and you will see it update.

When you save a design to a file, the current states for the various animations are also saved and the displayed text is saved as well. The next time you open the design, the various modes of the object will be the same as when the design was saved and the displayed text will also look the same.

## The Text I/O Examples in more/textio.dsn

- **Text/Integer/Character Output**

This is a text I/O object with no frills added. It is best suited for text and/or integer output. It has all of the built-in animations described above. The various mode animations (`justify_mode`, `max_char_count`, `base_mode`, `clip_on`, `append_on`, `scroll_on`, `cursor_mode`) are set to 0 so this object will display left justified text without limits. Any one or more of the mode states may be changed to produce varied behavior.

- **Text Input Areas**

Several examples show the use of the text I/O object as a component in a text input area. A text input area is created by adding mouse and keyboard input stimulus to a group containing a text I/O object and a shadowed background rectangle.

A left mouse button press over the area turns on a cursor and enables input. Keystrokes generate character events that update the text area. Left or right arrow key presses move the cursor through the displayed text. Characters can then be inserted between existing characters or deleted by pressing the Backspace key.

A left mouse button press outside of the text area or a Enter key press anywhere in the design hides the cursor and terminates input mode. At the same time, the stimulus executes the name `done` with a state value of 1. A designer can write a control block **WHEN** statement like:

```
WHEN done == {1}
```

and it will execute each time text input terminates. An **IF** statement like:

```
IF text == "Hello"
```

in the **WHEN** block can compare the contents of the text I/O object with a desired result.

Client application programs can select to receive `done` events and call

```
AtGetText(id, "text", char *buf, int bufSz) or  
altiaGetText("text", char *buf, int bSz)
```

to get the text from the input area. For more details, see the manual page for **AtGetText** in the [Altia API Reference Manual](#).

Immediately before the `done` event, the text input area also generates a set of text events - one event for each displayed character and a final event with a value of 0 to terminate the set. Instead of receiving the `done` event, client applications can register a callback for the `text` events with **AtAddTextCallback**. For more details, see the manual page for **AtAddTextCallback** in the [Altia API Reference Manual](#).

Note that the input areas accept any length of input. When the end of the area is reached, the text begins to scroll. This behavior results because the states for the `clip_on` and `scroll_on` animations are set to 1.

The text input area examples are of various sizes and fonts. The background and shadow colors can be changed by selecting a new background color for the group object from the color picker. To change the font, select a new font from the font selector in the Text Tool toolbar. The color of the text is changed by choosing a new foreground color from the color picker. To make the area longer or shorter, focus into the group, select the background rectangle, and stretch its right side with the editor's stretch tool. Make the background taller or shorter by stretching its top or bottom edge. You must also focus in to change the cursor's color. Select the text and choose a new background color for it.

- **Custom Cursor Input Area**

This example behaves like the text input areas just described but uses a custom cursor instead of the built-in cursor.

- **Number-Only Input Area**

This example is like the text input areas described above, but it only accepts number or minus sign (-) key presses. When a control block or client application program detects a done == 1 event, it can immediately check the state of the object's integer animation to get the number from the input area. Here is an example control block:

```
WHEN done == 1
    IF integer >= 100
        ...
END
```

It is also possible to check the input as a text string using the text name as shown above for the text input areas.

- **Digital Clock Example**

This text I/O object has control and timer stimulus to simulate a digital clock. It demonstrates how control can manage the dynamic display of text. The buttons provide a way to generate events to start, stop, and reset the clock.

To view the control for this text I/O object, copy it to the Graphics Editor's drawing area from a Models View. With it selected, press the Control button at the top of the editor to open the Control Editor. To view the object's timer stimulus, press the **Stimulate** button to open the Stimulus Editor.

## Using Text I/O Objects for Text Storage

It is often useful to get text input from the user and store it so it can be compared with new input at a later time. A text I/O object that acts as a temporary storage area can serve this purpose.

As an example, a design might have a text input area with a **text** animation renamed to **input\_text**. To save the previous input text for the area, one could copy the simple Text/Integer/Character Output example from more/textio.dsn into the design and place it in an area of the design that is not normally visible. Or, place it and use the **Hide** tool to hide it. If a prefix of **save** is added to the animation names using the rename dialog, then this object's **text** animation would have the name **save\_text**. A control block like the following could be written to compare and copy text between the objects:

```
WHEN input_done == 1
    IF input_text != save_text
        SET save_text input_text
    END
    ELSE
        SET err_text "Error! Try again..."
    END
END
```

where **err\_text** is the **text** animation for a third text I/O object which is used for displaying error or status messages.

## The Text I/O Object Indirection Feature

In control block statements, the contents of a text I/O object can be used as a name to reference the name's current state value. This is known as an indirect reference. It is used when it is inconvenient to reference an animation name directly during the creation of a control block.

As an example, a design might have a set of text I/O objects that form a list of 20 items and the animations for the objects have prefixes “item1” through “item20” so each item can be accessed uniquely. Assume that a push button in the design has stimulus that increments the state of **next\_item** by 1 on each press. The value of **next\_item** identifies the currently selected text I/O item in the list. Then assume that a second button has stimulus that sets the state of **clear\_item** to 1. The desired functionality is to press the next item button until **next\_item** is a desired value between 1 and 20 and then press the clear item button to clear the text at the current item.

To do this in control, one would just add a text I/O object to the design to serve as temporary text storage. If this object has a prefix of “temp”, then a **WHEN** control block like the following could be used to clear the selected item:

```

WHEN clear_item == 1
    SET temp_clear 1
    SET temp_append_on 1
    SET temp_text "item"
    SET temp_integer next_item
    SET temp_text "_clear"
    SET @temp_text 1
END

```

This control block writes a text string of the form “item#\_clear” into the temporary text I/O object where # is the current value for **next\_item**. The **append\_on** animation for the temporary text I/O object is set to 1 to allow text and integer values to be combined into one string.

Note that the last **SET** statement uses an @ symbol in front of **temp\_text**. This means that the text held by the temporary text I/O object should be used as an animation name and the state for that name should be set to 1. If **next\_item** has a state value of 5, then the text in the temporary text I/O object would become **item5\_clear**. This would set the **clear** animation to 1 for the text I/O item that has “item5” as a prefix.

The @ symbol only works in front of text I/O **text** animation names. Use of the @ symbol in front of other names will cause errors.

## 10.5.16 3D Scene Object

Purchased/3D\_Scene.dsn

The 3D Scene object allows Altia users to display 3D assets and animate them in their user interface designs. Artists may use their favorite 3D editing tool such as 3D Studio Max or Maya to create 3D scenes. A 3D scene may include meshes, lights, and cameras all of which may be animated. It is then exported to a COLLADA or OSG file that can be imported into the 3D Scene object in Altia Design. The 3D Scene object supports features commonly provided by OpenGL. These features are described further below.

**NOTE:** The 3D Scene object is not available by default. To use it, a license key must be purchased. Please contact your Altia representative for pricing information.

### 3D Scene Object Animations

The following table describes the Altia animations defined for the 3D Scene object.

Animation	Description
<b>3DScene_objRender</b>	Sets the 3D scene render policy. <ul style="list-style-type: none"><li><b>-2:</b> Renders the scene in wireframe when another animation is changed. <b>Note:</b> Not supported in DeepScreen.</li><li><b>-1:</b> Renders the scene in full shading when another animation is changed.</li><li><b>0:</b> Turns rendering off.</li><li><b>1:</b> Renders one frame of the scene in full shading.</li><li><b>2:</b> Renders one frame of the scene in wireframe. <b>Note:</b> Not supported in DeepScreen.</li></ul>
<b>3DScene_objWidth</b>	Sets the width of the 3D Scene object. The object will not visually change if <b>3DScene_objRender</b> is 0.  The width of the 3D Scene object can be set to a maximum value of 1024. To set the width larger than 1024, the environment variable <b>ALTIA_MAX_3D_DIM</b> must be set to the desired dimensions before running Altia

Design. For example, if **ALTIA\_MAX\_3D\_DIM** is set to "1280x1024", then **3DScene\_objWidth** can now be set up to 1280.

**3DScene\_objHeight** Sets the height of the 3D Scene object. The object will not visually change if **3DScene\_objRender** is 0.

The height of the 3D Scene object can be set to a maximum value of 768. To set the height larger than 768, the environment variable **ALTIA\_MAX\_3D\_DIM** must be set to the desired dimensions before running Altia Design. For example, if **ALTIA\_MAX\_3D\_DIM** is set to "1280x1024", then **3DScene\_objHeight** can now be set up to 1024.

**3DScene\_objMeshFn** Load a mesh file into the 3D Scene object. Set to "" to clear the scene. **3DScene\_objRender** does not affect this animation.

Although it is possible to type in the mesh filename, the **3D Scene Viewer** provides a more convenient way to browse for the mesh file. The **3D Scene Viewer** also converts COLLADA (.dae) files to OSG format for use on DeepScreen targets. No conversion is done if the mesh file is entered manually.

**3DScene\_lightName** Search for a light in the scene by name. The light's name can be set in the 3D editing tool. If found, information about the light, including its ID, are shown in the corresponding light animations below. Any changes to those light animations will now only affect the found light.

**Warning:** Because it takes longer to handle animation strings versus numerical values, do not use this animation to switch between lights. Instead use this animation to find the

light ID, then use the light ID to switch between lights.

#### **3DScene\_lightId**

Select a light by ID. If the ID is valid, information about the light are shown in the corresponding light animations below. Any changes to those light animations will now only affect the chosen light. Lights are assigned IDs in sequential order as they are imported starting with 0.

#### **3DScene\_lightType**

The type of light.

**Note:** This is read only.

- **0:** Directional light
- **1:** Point light
- **2:** Spotlight

#### **3DScene\_lightOn**

Indicates whether the light is on or off.

**Note:** This animation is reserved for future use.

#### **3DScene\_lightPosX**

The X component of the light's position in 3D space. For directional lights, this serves as the X component of the vector starting from the origin for the light's direction.

**Note:** This is read only.

<b>3DScene_lightPosY</b>	The Y component of the light's position in 3D space. For directional lights, this serves as the Y component of the vector starting from the origin for the light's direction.
	<b>Note:</b> This is read only.
<b>3DScene_lightPosZ</b>	The Z component of the light's position in 3D space. For directional lights, this serves as the Z component of the vector starting from the origin for the light's direction.
	<b>Note:</b> This is read only.
<b>3DScene_lightDirX</b>	The X component of a spot light's direction in 3D space. It is meaningless for other light types.
	<b>Note:</b> This is read only.
<b>3DScene_lightDirY</b>	The Y component of a spot light's direction in 3D space. It is meaningless for other light types.
	<b>Note:</b> This is read only.
<b>3DScene_lightDirZ</b>	The Z component of a spot light's direction in 3D space. It is meaningless for other light types.
	<b>Note:</b> This is read only.
<b>3DScene_lightColorDiff</b>	The diffuse color component of the light. It is stored as a 4 byte value packed as RRBBGGAA in hex.

<b>3DScene_lightColorSpec</b>	The specular color component of the light. It is stored as a 4 byte value packed as RRBBGGAA in hex.
<b>3DScene_lightColorAmb</b>	The ambient color component of the light. It is stored as a 4 byte value packed as RRBBGGAA in hex.
<b>3DScene_lightIntensity</b>	<p>The intensity of the light.</p> <p><b>Note:</b> This value cannot be imported into the 3D Scene object. It must be set as part of the design logic.</p> <ul style="list-style-type: none"> <li>• <b>&lt;= 0:</b> The light is not contributing to the scene and is considered turned off.</li> <li>• <b>0 - 1:</b> The light is dimmed.</li> <li>• <b>&gt; 1:</b> The light is brighter than normal.</li> </ul>
<b>3DScene_lightAttenConst</b>	The constant term of the attenuation equation. The equation controls how much the light is contributing to objects based on their distance from the light.
<b>3DScene_lightAttenLinear</b>	The linear term of the attenuation equation. The equation controls how much the light is contributing to objects based on their distance from the light.
<b>3DScene_lightAttenQuad</b>	The quadratic term of the attenuation equation. The equation controls how much the light is contributing to objects based on their distance from the light.

<b>3DScene_lightSpotCutoff</b>	The angle, in degrees, from the center of a spot light's cone to its edge. For a spot light, this value must be between 0 and 90. For any other light type, it must be set to 180. Other values are undefined.
<b>3DScene_lightSpotFade</b>	The exponential fade value for a spot light. It controls how much the light fades from the center of the spot light's cone to its edge.
<b>3DScene_camName</b>	Search for and set the current camera in the scene by name. The camera's name can be set in the 3D editing tool. If found, information about the camera, including its ID, are shown in the corresponding camera animations below. Any changes to those camera animations will now only affect the current camera. The view of the 3D Scene object switches to that of the camera.
	<b>Warning:</b> Because it takes longer to handle animation strings versus numerical values, do not use this animation to switch between cameras. Instead use this animation to find the camera ID, then use the camera ID to switch between cameras.
<b>3DScene_camId</b>	Select the camera by ID. If the ID is valid, information about the camera are shown in the corresponding camera animations below. Any changes to those camera animations will now only affect the current camera. The 3D Scene will render its view using the selected camera. Cameras are assigned IDs in sequential order as they are imported starting with 0. They may be customized as described in the section below.

<b>3DScene_camFovy</b>	The camera's field of view. The greater the value, the more of the scene is in view. See the camera section below on how to match what you see in Maya to that of the 3D Scene object.
<b>3DScene_camAspect</b>	The camera's aspect ratio given the 3D Scene object's width and height. A value of -1 will automatically calculate the aspect ratio.
<b>3DScene_camMode</b>	<p>The mode of the camera.</p> <p><b>Note:</b> This animation is reserved for future use.</p>
<b>3DScene_camPosX</b>	<p>The X component of the camera's position in 3D space.</p> <p><b>Note:</b> This is read only.</p>
<b>3DScene_camPosY</b>	<p>The Y component of the camera's position in 3D space.</p> <p><b>Note:</b> This is read only.</p>
<b>3DScene_camPosZ</b>	<p>The Z component of the camera's position in 3D space.</p> <p><b>Note:</b> This is read only.</p>
<b>3DScene_camTgtX</b>	<p>The X component of the point that the camera is looking at.</p> <p><b>Note:</b> This is read only.</p>

<b>3DScene_camTgtY</b>	The Y component of the point that the camera is looking at.
	<b>Note:</b> This is read only.
<b>3DScene_camTgtZ</b>	The Z component of the point that the camera is looking at.
	<b>Note:</b> This is read only.
<b>3DScene_camUpX</b>	The X component of the camera's up unit vector. This vector determines the orientation of the camera in 3D space.
	<b>Note:</b> This is read only.
<b>3DScene_camUpY</b>	The Y component of the camera's up unit vector. This vector determines the orientation of the camera in 3D space.
	<b>Note:</b> This is read only.
<b>3DScene_camUpZ</b>	The Z component of the camera's up unit vector. This vector determines the orientation of the camera in 3D space.
	<b>Note:</b> This is read only.
<b>3DScene_animName</b>	Search for and set the current 3D animation by name. The 3D animation name can be set in the 3D editing tool. If found, information about the 3D animation, including its ID, are shown in the corresponding animations below. Any changes to those animations will now only affect the current 3D animation.
	<b>Warning:</b> Because it takes longer to handle animation strings versus numerical values, do

not use this animation to switch between 3D animations. Instead use this animation to find the 3D animation ID, then use the ID to switch between 3D animations.

#### **3DScene\_animId**

Select the 3D animation by ID. If the ID is valid, information about the 3D animation are shown in the corresponding animations below. Any changes to those animations will now only affect the current 3D animation. 3D animations are assigned IDs in sequential order as they are imported starting with 0. They may be customized as described in the section below.

#### **3DScene\_animFrame**

The current 3D animation frame. This value can be directly set to manipulate the 3D animation.

**Hint:** Set **3DScene\_objRender** to -1 to cause the view to update when setting a new frame value.

#### **3DScene\_animFPS**

The rate to play the 3D animation using the **3DScene\_animCmd**.

**Note:** This is read only.

#### **3DScene\_animCmd**

Action to take on the 3D animation.

**Hint:** Setting an action does not result in seeing the 3D animation automatically. Set **3DScene\_objRender** to 1 periodically to see the 3D animation at the **3DScene\_animFPS** rate.

- **0:** Play the 3D animation once. When done, set the **3DScene\_animDone** value to the 3D animation ID.

- **1:** Play the 3D animation in a loop.
- **2:** Play the 3D animation in a ping-pong loop.
- **3:** Pause the 3D animation. The next play command will continue from the current frame.
- **4:** Move the play head to the beginning.
- **5:** Stop the 3D animation. The next play command will start from the first frame.
- **6:** Stop all 3D animations.

#### **3DScene\_animDone**

When a 3D animation is done playing once, this value is set to the 3D animation ID that was playing.

**Note:** This is read only.

## Lights

A 3D scene must have lights to be visible. Up to 8 lights are supported. However, there is a performance cost (see the Performance Consideration discussion below). Currently, lights cannot be animated via Altia's animations. However, they can be animated on the timeline of the 3D editor and imported into the 3D Scene object where they are manipulated as with any other 3D animation. See the 3D Animations description below.

The color of a light is separated into three different components: diffuse, ambient, and specular. The diffuse color is the color of the light when it shines on a surface. It attenuates or fades the more distance there is between the light source and the surface it's affecting. Attenuation is calculated using a quadratic equation governed by the three light attenuation factors. The light's diffuse color is combined with the diffuse color of the surface material for the final color.

The ambient color is usually a very muted shade of the diffuse color. It is used to give a very little bit of visibility to places in the scene where no light can reach. It is also combined with the surface material's ambient color, but is not affected by the distance between the light's position and the surface.

The specular color is used on parts of surfaces where the light is reflected directly into the view. This is also known as specular highlights. It is combined with the surface material's specular color for the final color. It is not affected by distance, but rather by the angle of reflection based on the position of the camera.

The 3D Scene object supports the three types of lights defined by OpenGL: directional lights, point lights, and spotlights. Directional lights can be thought of as lights coming from very far away like the Sun. Because of the long distance between the light source and the surface that it's affecting, the light rays are essentially parallel to each other going in one direction. The position of the light is used as the direction vector relative to the origin. For example, if the position of the light is (1.0, 0.0, 0.0), then the light rays are considered to be going in the X direction. Directional lights do not lose their power so the attenuation factors are not considered. The result is that all objects in a scene are affected equally by directional lights.

Point lights are closer to the scene and their rays are omni-directional similar to a light bulb. Objects in a scene are affected differently based on the position of the lights, objects, and camera. The energy of the light falls off as objects are farther away from the light. The amount of fall off is controlled by the three attenuation factors.

Spotlights are point lights that are limited to a certain area. The area of affect is a cone defined by the light's direction and an angle from the center of the cone to its edge. The spotlight's fade factor can be used to fade the light from the center of the cone to its edge.

A light's intensity controls how much the light is contributing to the scene. It is not supported by OpenGL so it must be set individually after a mesh file is imported.

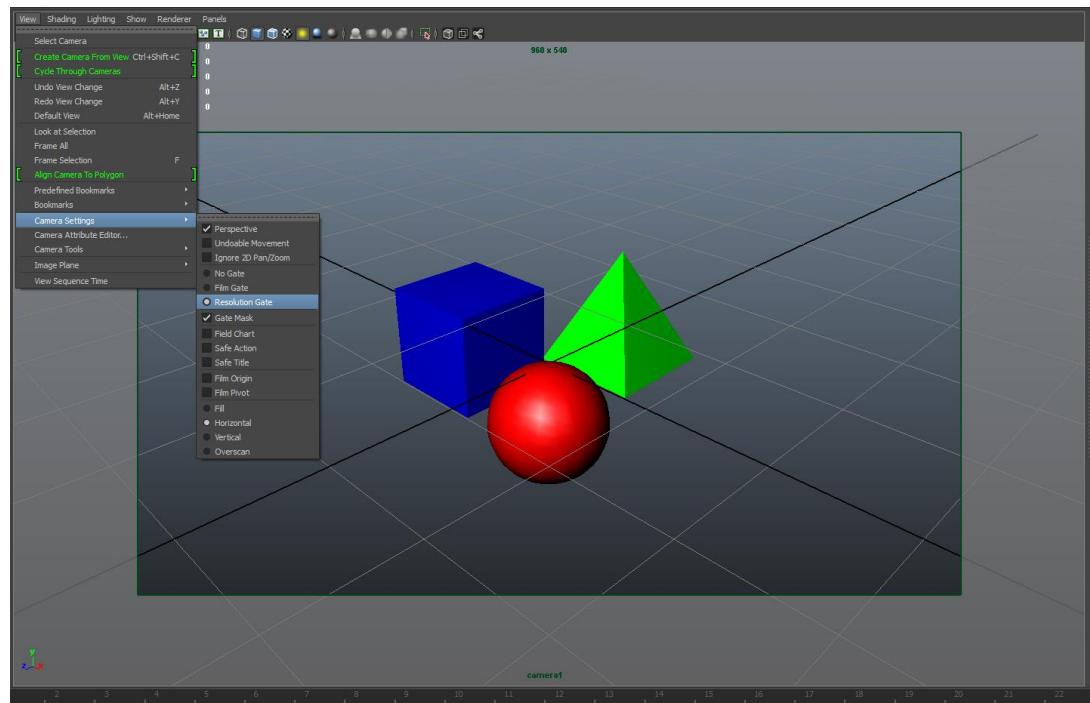
## Cameras

Cameras are used by the 3D Scene object to render its view. By default the 3D Scene object will render using the first camera it finds. This is the camera with ID 0. Multiple cameras may be created in the 3D editing tool to view the scene from different angles. Simply switch the camera ID to switch the 3D Scene object's view. Camera IDs are customizable as described in the section below.

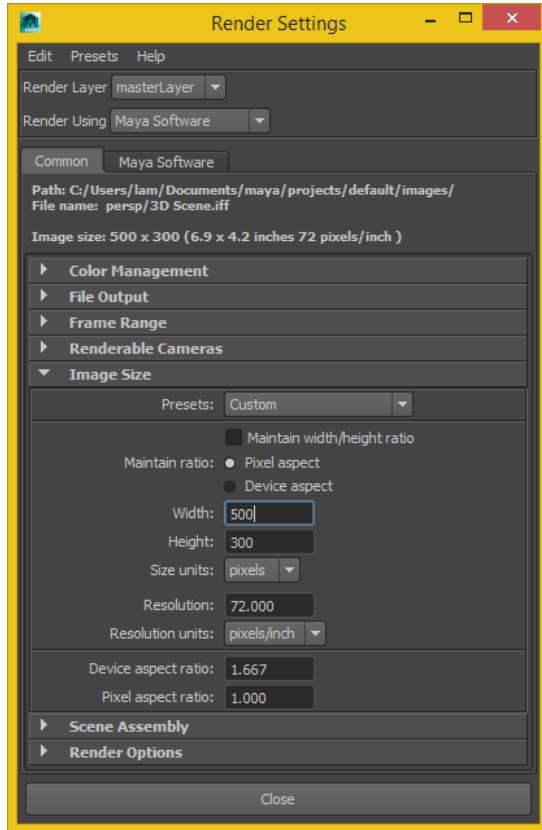
A camera's position defines where it is in 3D space. Its attitude is defined by its target and up vectors. Currently, cameras cannot be animated via Altia's animations. However, they can be animated on the timeline of the 3D editor and imported into the 3D Scene object where they are manipulated as with any other 3D animation. See the 3D Animations description below.

How a camera is setup will affect the way it renders the scene. Follow these steps to ensure that what you see in the 3D Scene object will be what you see in Maya.

1. In Maya, turn on the **Resolution Gate** by selecting **View->Camera Settings->Resolution Gate**. This will create a border around the field of view.

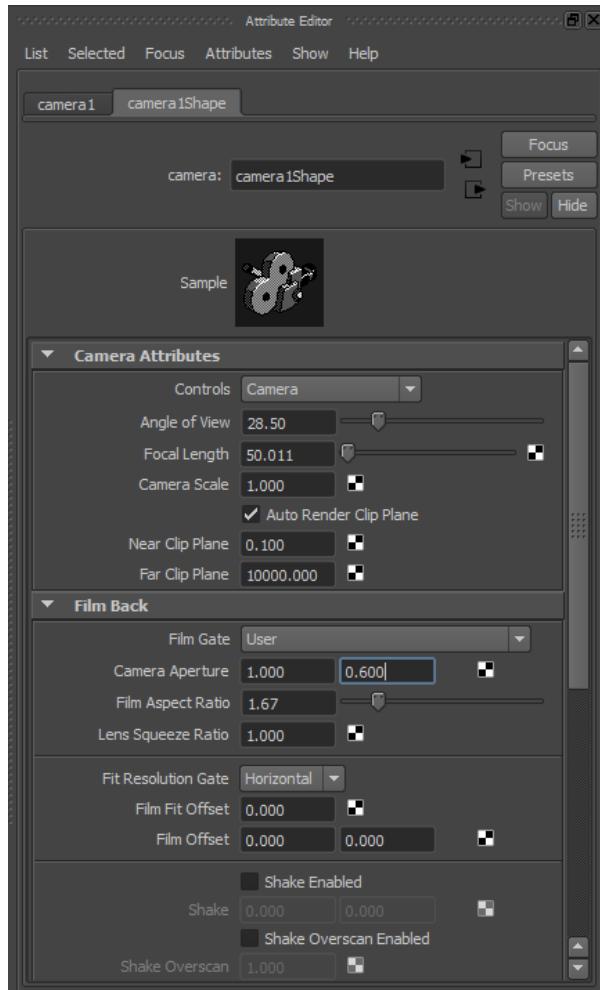


- Set the gate's dimensions to match that of the 3D Scene object by selecting **Window->Rendering Editors->Rendering Settings** and setting the **Image Size Width** and **Height** as desired.



- Select each camera in the scene using the **Outliner** and set the view to see through each camera by selecting **Panel->Perspective->[camera name]**.
- In the camera shape tab of the **Attribute Editor** set the **Angle of View** as desired. The larger this value, the more of the scene will be in view. Alternately, set the **Focal Length** to achieve the inverse effect.

5. The **Camera Apperture Width** must be set to **1.000** and the **Camera Apperture Height** set to the height to width ratio. For example, for a width of 500 and a height of 300, then the **Camera Apperture Height** must be set to **0.600**.



6. In Altia Design, select the 3D Scene object and set its **3DScene\_objWidth** and **3DScene\_objHeight** to match the width and height set in Maya.

## 3D Animations

3D animations may be manipulated directly or played as defined on the timeline. Direct manipulation is useful when the 3D scene is presenting some information such as a speedometer displaying the speed of a car. Playing a predefined 3D animation is useful in the case of a welcome screen.

To setup direct 3D animation manipulation, set 3D animation ID to the desired one. Then set **3DScene\_objRender** to -1. This will cause the 3D Scene object to render a new image very time a new frame value is set. For example, suppose we want to display the speed of a car using a 3D gauge. In the 3D editing tool, create a 3D animation of the gauge's needle that goes from 0 to 120mph. Any number of frames may be used, but to keep the client code that feeds this value simple, use 121 frames. See the [Altia API Reference Manual](#) for more information on how to write client code. Import the 3D scene into the 3D Scene object and set the animation values as described above. When the client code sets the animation frame value to the speed of the car, the 3D gauge will display the correct speed.

To use the 3D Scene object to display a welcome screen, set the 3D animation ID to 0 as above. This time, set **3DScene\_objRender** to 0. Then create a timer to periodically set **3DScene\_objRender** to 1. A timer of 40ms is equivalent to 24fps. See the chapter on the [Stimulus Editor](#) for how to create a timer. As a word of caution, creating timers for individual objects in large designs is bad for performance and not recommended. Instead, use a global timer to run a state machine that can distribute values to the objects. To play the 3D animation, set the 3D animation command to 0. The timer will tell the 3D Scene object to render every time it expires. With each render request, the 3D Scene object will calculate its current frame based on how much time has elapsed since it was first told to play and the number of frames per second defined in the 3D editing tool. At the end of the 3D animation sequence, the **3DScene\_animDone** value is set to the 3D animation ID.

Animations defined for distinct meshes of a scene on the 3D editor's timeline can be controlled individually in the 3D Scene object. When imported, the original timeline, as defined in the 3D editor, is kept as 3D animation ID 0. Additionally, a separate timeline is created for each mesh with animations defined. They are named the same as their mesh name with IDs starting with 1. These 3D animations can be manipulated as with the one from the original timeline. For example, a car model is designed with a front right door and a front left door that can rotate on a pivot to open and close on the timeline. Additionally, on the same timeline, the car is rotated about its center. The door meshes are children of the car. When imported into the 3D Scene object, four 3D animations are available. The first animation with ID 0, will manipulate the car's rotation and both doors, all at the same time. The second animation with ID 1, may manipulate just the car's rotation. The third animation with ID 2, may manipulate just the left or right door. And the last animation with ID 3, may manipulate just the other door. IDs are assigned as the animations are imported. The 3D Scene Viewer shows a list of the 3D animations and their IDs. 3D Animation IDs may be customized as described in the section below.

## The 3D Scene Viewer

The 3D Scene Viewer allows users to interact with 3D contents more intuitively. It can be used to load mesh files, see the position of lights and cameras, and view the scene from different angles. To display the 3D Scene Viewer pane, right click on the 3D Scene object in the Universe window or the Navigator pane, and select the **3D Viewer...** menu item. Each 3D Scene object will have its own tab in the viewer. Click on the X icon to the right of the tab to close the viewer for that 3D Scene object. Click on the X icon of the pane to close the pane. The 3D Scene Viewer pane can also be displayed using the **View** ribbon's **Windows->3D Viewer** option.

Pressing the **Open...** button will bring up a file chooser dialog allowing the user to choose a COLLADA (.dae) or OSG (.osgb) file to import. If a COLLADA file is chosen, it must be converted to an OSG file to be used by the 3D Scene object. A dialog is displayed to inform the user of this action. Press the **Convert** button to continue. The resulting file will have the same filename as the original COLLADA file with an extension of osgb. The converted file must be used on the DeepScreen target platform.

The **Objects** list displays the lights, cameras, and meshes in the 3D scene. Lights and cameras are displayed with their IDs. Names given to the scene objects in the 3D editor are also shown.

The **Animations** list displays 3D animations defined in the 3D editor. Each 3D animation has an ID, name, and length in seconds. Select the animation to enable the 3D animation controls below the list. Slide the **Timeline** slider to manipulate the 3D animation's frame value. Press the **Play** button to play the 3D animation one time. While the animation is playing, press the same button to pause it. Press the **Jump to Start** button to reset the play head. Press the **Play Mode** options button to choose the play mode. Changing the play mode does not start the animation. Press the **Play** button to play the 3D animation using the new play mode.

The **Scene View** options button allows the user to change the view of the scene.

- **Perspective** - This is the default view. If the 3D Scene Viewer is thought of in terms of a movie set, then this view is from the director's point of view. It allows the user to move through the scene freely.
- **Orthographic** - This group of views include Top, Bottom, Front, Back, Left, and Right views. These options restrict the view to their respective angle of the scene.
- **Lights** - Selecting a light will set the view to the position and attitude of the light.
- **Cameras** - Selecting a camera will set the view to the position and attitude of the camera.

The **Object Shading** option button defaults to textured rendering. In this rendering mode, the objects are rendered fully shaded. Viewing the scene in wireframe can reveal any hidden objects that might not be needed in the scene.

The **Show Grid**, **Show Lights**, and **Show Cameras** check boxes are used to show and hide the grid, light avatars, and camera avatars in the 3D Scene Viewer. These supporting scene assets are not rendered in the final view of the 3D Scene object. They are rendered here as a visual reference to help locate them in the scene.

The **3D View** displays the rendered scene with the selected options. Left-click and drag in the view to rotate it. Right-click and drag to pan the view. Scroll the mouse wheel to zoom in and out. Press the **Home** key to reset the view.

## Custom Camera and Animation IDs

Cameras and animations in a 3D Scene object are given IDs as they are imported. These IDs may be customized by the user in order to keep their use from having to change. To change the ID, simply double click it in the 3D Scene Viewer. A numeric spin box will be displayed to allow it to be changed. Only non-negative numbers are allowed. If the user attempts to change an ID to one that is currently being used, an option dialog is displayed giving the user these options.

- **Swap** - Swap the IDs of the selected item and the item with the desired ID.
- **Push** - Increment the IDs of the item with the desired ID and all other items with sequential IDs greater than the desired ID. Then set the ID of the selected item to the desired value.
- **Cancel** - Don't change the ID.

When an ID is changed, a mapping is created between the ID and the name of the camera or animation. This mapping is saved in a file with the same base name as the 3D scene file name with the extension ".dat". For example, if the 3D scene file name is "myScene.osgb", the mapping file will be named "myScene.dat". When generating DeepScreen code, the 3D scene file and the mapping file will be embedded in that code by default. If the code generation option, **3D Scene object user file system**, is checked, then both the 3D scene file and mapping file will be needed on the target to be used by the 3D Scene object.

**Warning:** Because of the mapping between the IDs and object names, it is important that names given to 3D objects stay constant as the scene is modified in the 3D editing tool.

## Performance Considerations

3D assets require a lot of computing resources. Here are some tips on how to keep performance reasonable on target hardware.

- Keep the number of polygons to the minimum when creating meshes. Use fewer polygons where details are not necessary to compensate for other places where more details are needed. Remove anything that is not visible. As an example, a typical car model should only need about 10-20k triangles.
- Keep the number of lights to a minimum. The shader code that renders each pixel of a 3D view calculates the color of the pixel based partially on how many lights are in the scene. More lights mean more render time. A typical scene should only need one light. A reflection map can improve the quality of the scene without the performance burden of multiple lights.
- Use textures with dimensions that are power of two. On the PC, this is not a problem because there are enough hardware resources. However, on embedded devices, this is a requirement. To conform to this requirement, textures are resized at runtime if needed. To skip this performance killing procedure, manually convert textures to the correct size when skinning your meshes.

- Keep texture sizes as small as possible. This reduces the graphic memory requirement.
- Keep the 3D Scene object dimensions as small as possible. This reduces the number of pixels that must be rendered. A good size to use on target is around 480x480.
- Keep the 3D rendering frequency to a minimum to allow good overall HMI performance. Since the results of 3D renders are cached, it doesn't need to be re-rendered when other parts of the HMI is updated. Smooth 3D animation can still be achieved with a rendering frequency of 16fps (60ms).

## Using the 3D Scene object with DeepScreen

When generating code for a design with one or more 3D Scene objects, the following options should be considered:

- **3D Scene object use file system** - If checked, 3D Scene objects will load their scene assets from the file system. This allows a 3D Scene object to load different scene assets or scene assets to be swapped without the need to rebuild the application. Your DeepScreen target must support a file system – see the Altia DeepScreen target User's Guide for your target.

When unchecked, the code generator will compress and store scene assets in a block of memory for 3D Scene objects to load. This removes the need for a file system and also reduces the memory footprint needed for 3D scene assets. However, a 3D Scene object will only be able to load scene assets that are already assigned to itself or another 3D Scene object. This is because the code generator can only know about assets that are in use at code generation time.

- **Use Dynamic Memory Allocation** - This option must be checked when a model contains a 3D Scene object.

# Chapter 11: Validator

Altia Design's Validator pane is used check a design for problems. The Validator has multiple uses including:

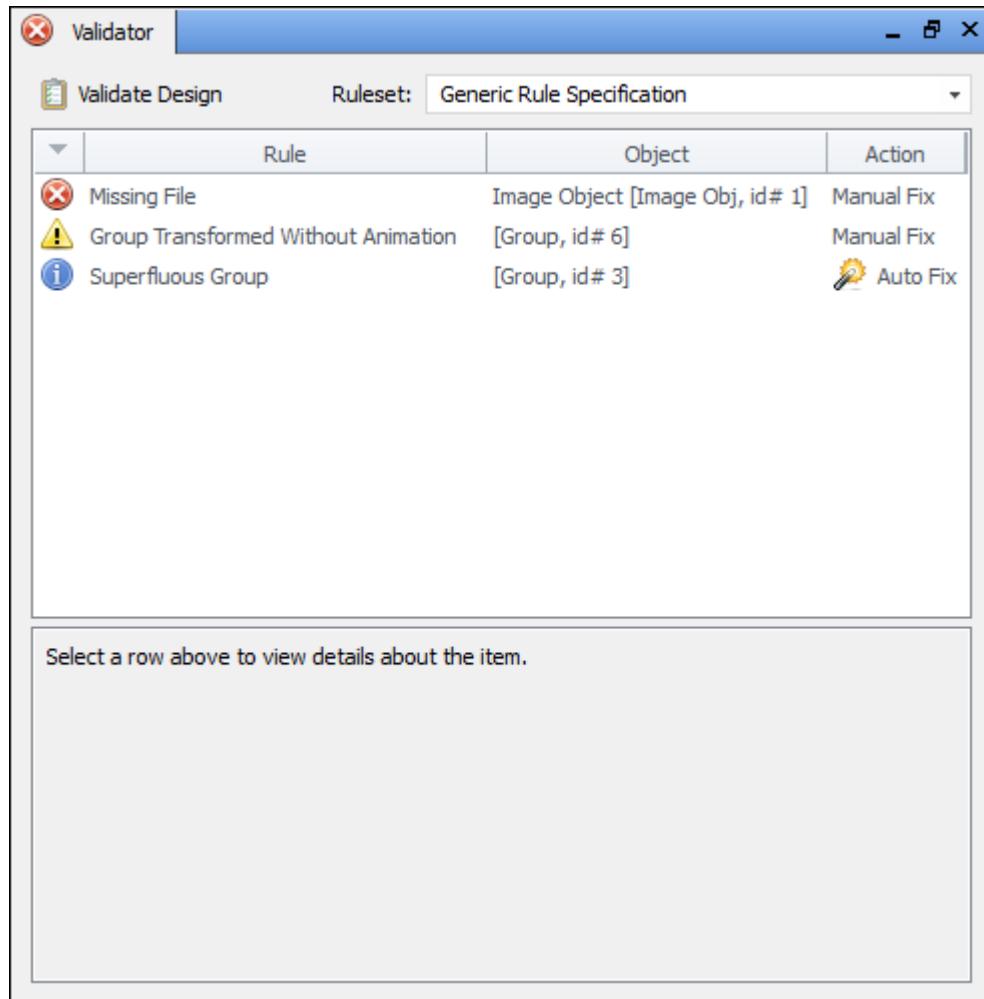
- Checking a design for problems in real time, as the design is manipulated
- Checking a design prior to DeepScreen code generation for target specific problems
- Evaluating the validity of a design for different DeepScreen targets (hardware platforms)
- Learning how to fix common problems in a design

If the Validator pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Validator** menu option is checked.

## 11.1 Validator Layout

The Validator pane is divided into three sections:

- **Toolbar**
- **Problem List**
- **Problem Details**



## 11.1.1 Toolbar



The toolbar contains the following controls:

- **Tab (with problem icon)**
- **Validate Design Button**
- **Ruleset Selector**

**Tab** - The tab for the Validator Pane will contain an Error icon if there is at least one Error in the list. If there are no errors and at least one Warning in the list, then a Warning icon will appear in the tab. This provides a quick indicator as to problem status when the Validator Pane is behind another pane in the User Interface.

**Validate Design Button** - This button will perform a full validation of the current design. It is the only way to validate a design when Real Time Validation is turned off. When pressing the button, all problems in the Problem List will be removed, and the design fully checked.

**Ruleset Selector** - This combo-box contains a list of all the rule sets available on your computer for the Validator. Rule sets will be sorted into the following categories:

- **Installed Targets** - Rules that shipped with DeepScreen targets currently installed on your computer
- **General** - Rules that are not specific to any DeepScreen target
- **Other Targets** - Rules for DeepScreen targets, but these targets are not installed on your computer

Selecting a new Rule Set will clear the Problem List. If Real Time Validation is enabled, problems detected by the new Rule Set will begin to appear in the Problem List automatically. If Real Time Validation is disabled, the Validate Design button must be pressed to check for problems with the new Rule Set.

**NOTE:** It is not necessary to use the Validate Design button if Real Time Validation is enabled. This is because the validator is always running in Real Time, checking your work for problems. The Real Time Validation setting can be found on the Code Generation Ribbon.

## 11.1.2 Problem List

▼	Rule	Object	Action
	Missing File	Image Object [Image Obj, id# 1]	Manual Fix
	Group Transformed Without Animation	[Group, id# 6]	Manual Fix
	Superfluous Group	[Group, id# 3]	Auto Fix

The Problem List contains a table of all problems found in your design. These problems may have been found by pressing the Validate Design button, or they may have been found by the Real Time Validator.

Each row in the list contains the following information:

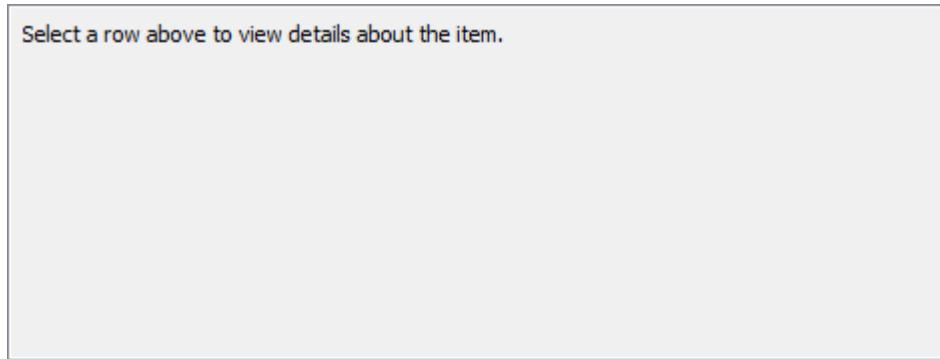
- **Icon** - A graphic indicating if the problem is an error, warning, or informational.
- **Rule** - Title of the rule that was violated.
- **Object** - Name of the object that caused the violation (may be blank for rules that are not object specific).
- **Action** - Whether the problem can be fixed automatically, or if it must be fixed manually.

If a problem can be fixed automatically, an **Auto Fix** button will appear in the list under the Action column. Pressing this button will fix that specific problem and remove it from the list.

The problems in the list can be sorted by clicking on the column headers.

### 11.1.3 Problem Details

The Problem details will show information on the currently selected problem in the Problem List. If no problem is selected the Problem Details will show the following:



If a problem is selected the Problem Details will show the following (example for Superfluous Group):

A screenshot of a software window titled "Problem Details". The window is divided into two main sections: "INFO" on the left and "HOW TO FIX" on the right. The "INFO" section contains the following text:

This group object has one child and no substantive definitions like animations, stimulus, timers, control code, or connections. Consider removing the group because it serves no purpose.

This reduces draw performance and increases resource requirements (like memory consumption).

The "HOW TO FIX" section contains the following text:

Remove this group using the Ungroup command. This command can be found on the Home ribbon or in the context menu that appears when right-clicking on this object.

At the bottom right of the window is a button labeled "Fix It Now" with a gear icon.

The Problem Details contain the following information:

- **Type** - Icon and text showing the type of problem (error, warning, informational).
- **Description** - A description of the problem including object-specific details (if required).
- **How To Fix** - A description of the steps required to fix the problem and how to avoid the problem in the future.
- **Fix It Now** - A button to fix the problem. Pressing this button will fix that specific problem and remove it from the list.

**NOTE:** The Fix It Now button will be disabled for problems that cannot be fixed automatically.

## 11.2 Ribbon Controls

Additional controls for the Validator can be found on the Code Generation Ribbon. This includes the following:

- **Validate Design Button** - Same as the Validate Design Button on the Toolbar.
- **Real Time Validation** - Turns Real Time Validation on and off.
- **Validate Before Generating Code** - Runs the Validator before generating code and block code generation if problems exist.

**NOTE:** Please refer to the Code Generation Ribbon chapter of this manual for further details.

## 11.3 Real Time Validation

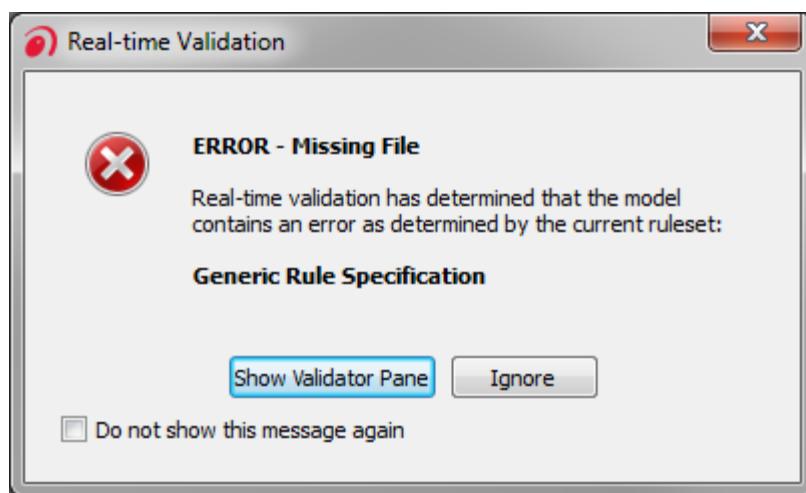
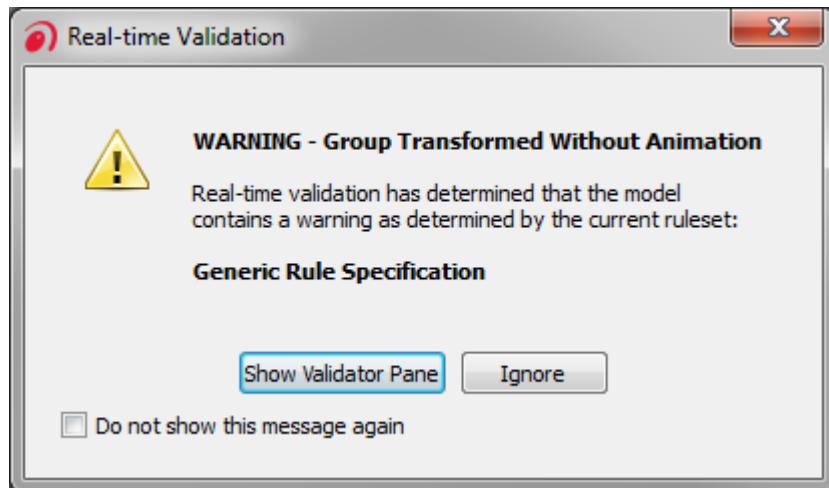
### 11.3.1 Operation

The Real Time Validation feature is enabled using the control on the Code Generation Ribbon. When enabled, the Validator is always running in the background checking your design for problems. As problems are found, they will be added to the Problem List. In addition, fixed problems will be removed from the Problem List.

The Real Time Validator will consume background CPU cycles in order to check for problems. The computations are limited to a single thread so the impact should be minimal on computers with multiple cores. If you find that your system performance becomes sluggish when running Altia Design, you can disable Real Time Validation using the ribbon controls.

## 11.3.2 Problem Detection

When the Real Time Validator detects a problem, it will be added to the Problem List. When this happens, the Validator will present a dialog if the problem was an Error or Warning type. The dialog describes the problem detected and what Rule Set was used to find the problem. The following are examples for both a Warning and an Error:



There is some spam prevention logic for the dialogs. Only the first occurrence (during a check cycle) of each problem will be presented. A check cycle occurs by:

- Pressing the Validate Design Button (in which case the cycle covers checking the entire design).
- A tick of the Real Time Validator (in which case the cycle covers one rule for a single object).

**NOTE:** Checking "Do not show this message again" will block all problem dialogs (both warnings and errors).

Problems will always be shown in the Validator Problem List, regardless if the dialogs are shown or not.

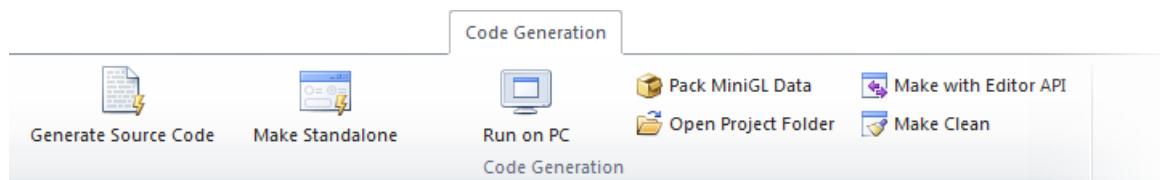
# Chapter 12: DeepScreen Code Generation

Altia DeepScreen is a graphics code generator that converts your Altia Design prototype graphics into tight, deployable graphics code. With DeepScreen you can reserve your precious programming time for more important things, like building your application code and getting it to run on your hardware.

DeepScreen generates fully compliant ANSI C source code. There are no added libraries or engines that must go on target.

**NOTE:** Generating source code requires a licensed DeepScreen code generation target.  
Please see your Altia representative for more information.

## 12.1 Code Generation Ribbon

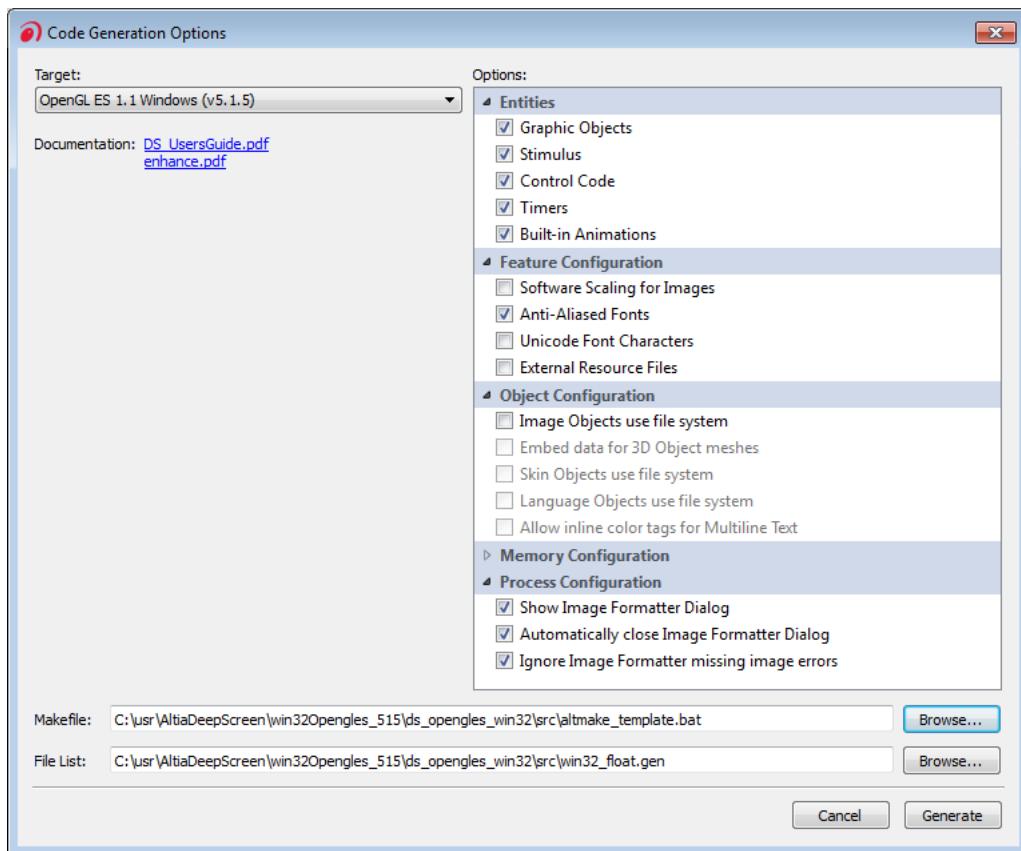


The Code Generation ribbon provides controls to generate source code from your Altia Design model to compile and run on your hardware target.

### 12.1.1 Generate Source Code

Pressing the Generate Source Code button displays the **Code Generation Options** dialog. The Code Generation Dialog allows you select a DeepScreen code generation target, configure code generation options and optimizations, and generate the source code. The Code Generation Options dialog also provides convenient links to the documentation for the target you have selected.

**NOTE:** For complete and detailed information on generating code and configuring code generation options, refer to the DeepScreen User's Guide and the DeepScreen Target User's Guide for your target.



## 12.1.2 Make Standalone

After generating code for your target, **Make Standalone** will compile a standalone executable. refer to the Altia DeepScreen target User's Guide for your target for details.

## 12.1.3 Run on PC

Runs the most recently compiled executable (if the selected DeepScreen target creates a Windows desktop executable).

## 12.1.4 Pack MiniGL Data

Converts code generated for the AltiaGL target into code for MiniGL. See the MiniGL DeepScreen target documentation for specific information and instructions.

## **12.1.5 Make with Editor API**

The results of the **Make with Editor API** are target-specific. Refer to the Altia DeepScreen target User's Guide for your target for details.

## **12.1.6 Make Clean**

The **Make Clean** option will run the makefile script specified in the Build File Selection field with the "clean" directive. For example if the makefile script is "altnake.bat" the Make Clean button will execute "altnake.bat clean" in the command line interface (generally CMD.exe on Microsoft Windows)

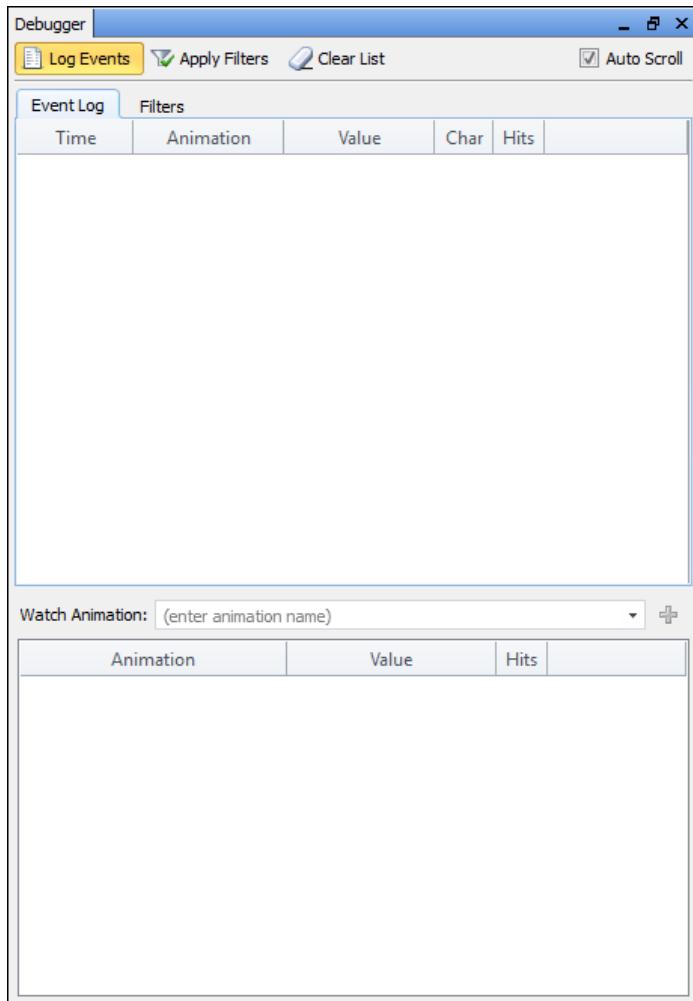
## **12.1.7 Open Project Folder**

The **Open Project Folder** button provides quick access to the file system location where your .dsn and its generated code are saved.

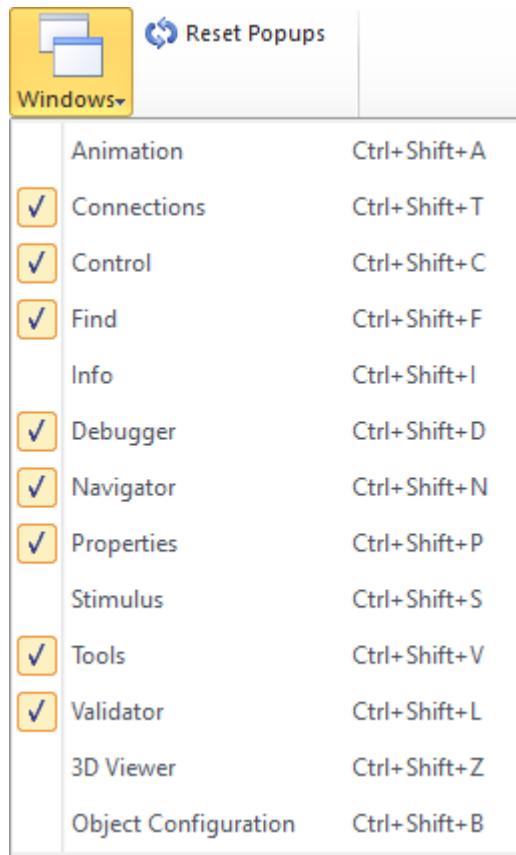
# Chapter 13: Debugger

The Debugger pane provides an easy way to monitor animation events that occur in your design whether they're generated from Control Code, Stimulus Events, or even external applications that are connected to Altia Design as clients. All of the animations in your design are accessible in the Debugger. When enabled, the Debugger captures all animation events that occur and makes their history available for future analysis. Animations can be individually watched and filters can be applied that allow you to focus on a subset of your design.

## 13.1 Debugger Pane Overview



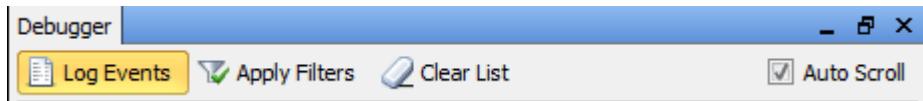
The Debugger Pane can be hidden or shown just like all of the other Panes in Altia Design 11; Select the **View** Ribbon and then the **Windows** button to show a list of available panes and to select the **Debugger** pane as shown below.



The Debugger Pane is also available via the **Ctrl+Shift+D** hot key combination.

The Debugger Pane has several distinct control areas; the **Control Buttons**, the **Event Log** and **Filters** lists, and the persistent **Watch** list.

## 13.2 Control Buttons



- **Log Events** - This toggle button enables and disables the capture of the animation events. While toggled all animation events that occur in your design will be logged by the debugger, regardless of any filter settings. No animation events are captured or recorded when Log Events is toggled off.
- **Apply Filters** - When toggled all animation events will be logged and recorded but only those animation events that match the filter criteria will be displayed in the Event Log list area. See [Section 13.4, Filters](#) for more information about controlling which animation events are displayed in real-time..
- **Clear List** - This button clears all recorded animation events from the Event Log list.

**NOTE:** The Debugger Pane will record animation events indefinitely as long as the Log Events button is toggled. While being extremely memory efficient the Debugger will consume memory as it records animation events and this can potentially be an issue on PCs with limited amounts of RAM.

The Clear List button clears any memory consumed by the Debugger and can be used to reclaim RAM after long periods of use on resource limited systems. Under normal circumstances however the Debugger can log many millions of events without an issue.

- **Auto Scroll** - This check box controls the behavior of the Event Log area while animation events are being logged and displayed. By default the Event Log will always scroll to the most recently recorded event animation (unless the scroll bar is being actively dragged/held). Unchecking the Auto Scroll check box will prevent the Event Log from scrolling automatically without affecting the recording of animation events.

### 13.3 Event Log

The Event Log displays recorded animation events as they occur in real-time. Each row of the **Event Log** corresponds to a single animation event. The time the event was received, the name of the animation, the numerical value of the event, the character value of the event, and the number of times the event has been received are displayed in separate columns as shown below.

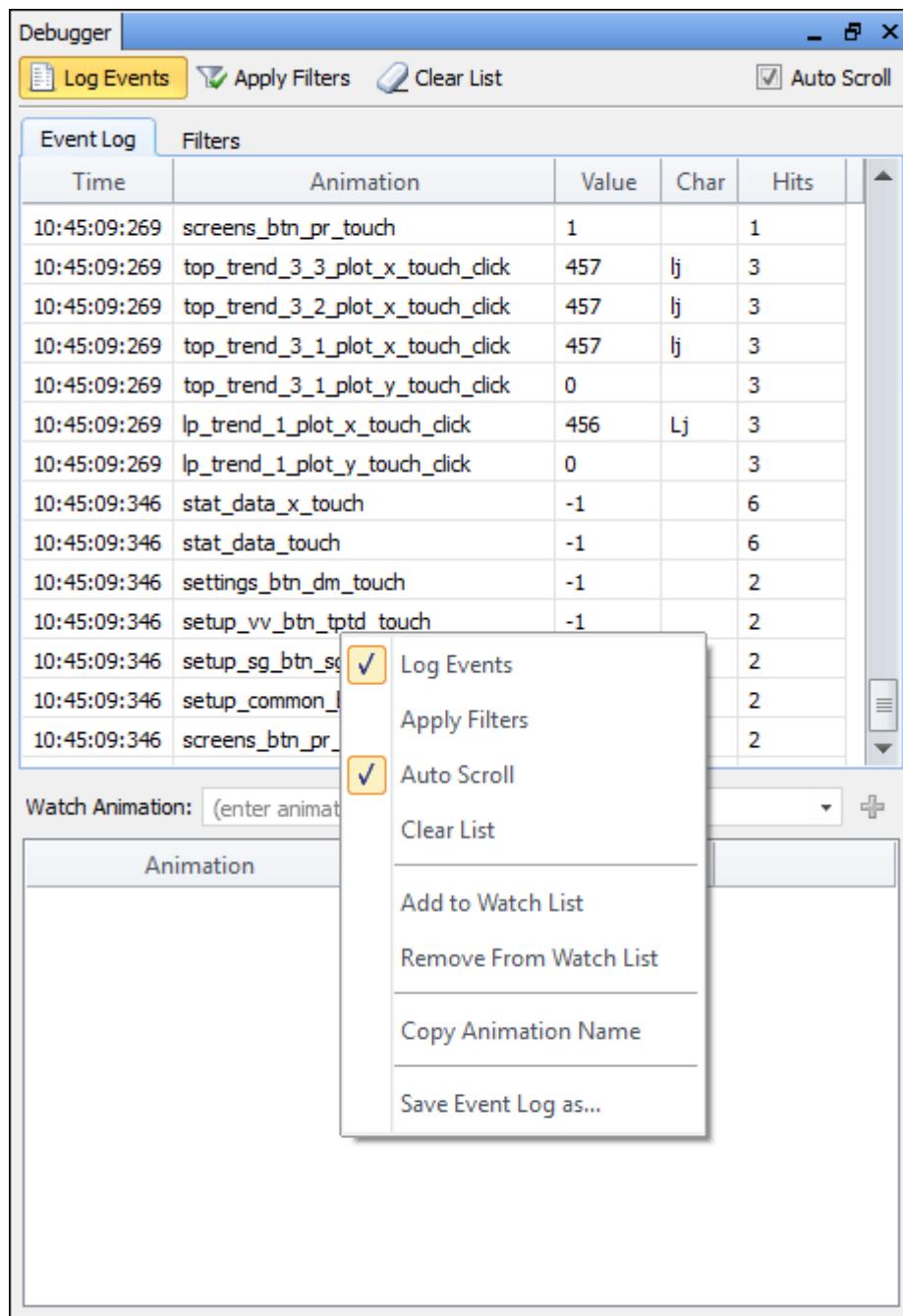
The screenshot shows the 'Event Log' tab selected in the Debugger pane. The window title is 'Debugger'. At the top, there are buttons for 'Log Events' (highlighted in yellow), 'Apply Filters', 'Clear List', and a checked 'Auto Scroll' checkbox. Below the toolbar is a header row with columns: Time, Animation, Value, Char, and Hits. The main area contains a list of 16 events with the following data:

Time	Animation	Value	Char	Hits
10:45:09:269	screens_btn_pr_touch	1		1
10:45:09:269	top_trend_3_3_plot_x_touch_click	457	lj	3
10:45:09:269	top_trend_3_2_plot_x_touch_click	457	lj	3
10:45:09:269	top_trend_3_1_plot_x_touch_click	457	lj	3
10:45:09:269	top_trend_3_1_plot_y_touch_click	0		3
10:45:09:269	lp_trend_1_plot_x_touch_click	456	Lj	3
10:45:09:269	lp_trend_1_plot_y_touch_click	0		3
10:45:09:346	stat_data_x_touch	-1		6
10:45:09:346	stat_data_touch	-1		6
10:45:09:346	settings_btn_dm_touch	-1		2
10:45:09:346	setup_vv_btn_tptd_touch	-1		2
10:45:09:346	setup_sg_btn_sgs_touch	-1		2
10:45:09:346	setup_common_btn_oc_touch	-1		2
10:45:09:346	screens_btn_pr_touch	-1		2

Below the event log is a 'Watch Animation:' dropdown set to '(enter animation name)' with a '+' button. At the bottom is a small table with columns: Animation, Value, and Hits.

The width of each column is adjustable as is the width of the Debugger Pane itself. The height of the Event Log relative to the Watch list is also adjustable; hovering over the bottom of the Event Log area will show a resize handle that can be used to either increase or decrease the height of the Event Log. The scroll bar is used to show previous events (by scrolling up) or more recent events (by scrolling down). The

Event Log list area also provides a context menu interface that allows quick customization of the Debugger or to perform operations on a single animation event.



The context menu is raised by right clicking a row of the Event Log and it has the following options:

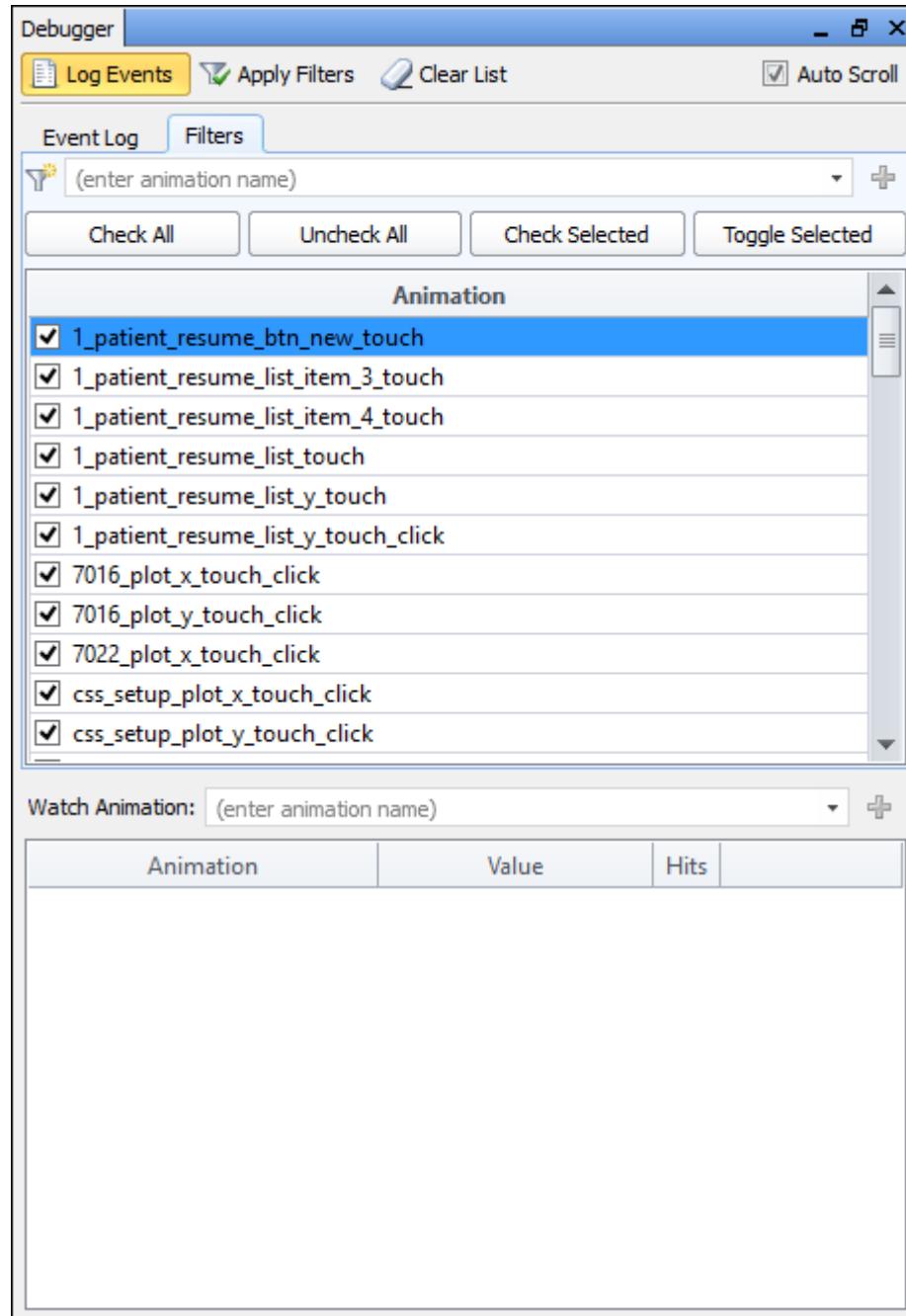
- **Log Event** - This option is identical to the Log Event button at the top of the Debugger Pane.
- **Apply Filters** - This option is identical to the Apply Filters button at the top of the Debugger Pane.
- **Auto Scroll** - This option is identical to the Auto Scroll check box at the top of the Debugger Pane.

- **Clear List** - This option is identical to the Clear List button at the top of the Debugger Pane.
- **Add to Watch List** - This option adds the animation event of the selected row with the right-click to the Watch list (see [Section 13.5, Watch List](#)).
- **Remove From Watch List** - This option removes the animation event of the selected row with the right-click to the Watch list (see [Section 13.5, Watch List](#)).
- **Copy Animation Name** - This option copies the name of the animation event selected to the clipboard. This is useful for copying a name and then pasting it in the Animation pane in order to quickly control an animation.
- **Save Event Log As...** - This option raises the Save File dialog for saving a copy of the Event Log to a text or CSV file (see [Section 13.6, Saving the Debugger Log](#)).

As mentioned, the Event Log displays both the numeric value (integer or floating point, depending on the value) of each animation event in addition to a character value. For most animation events the character value will not be relevant. The character value, however, is useful for displaying string animation events (like those that set the string value of a Text IO). When a string is sent to a animation name it is parsed into a stream of individual characters where each character will be displayed on a single row in the Event Log.

## 13.4 Filters

The Filters interface is accessible by selecting the **Filters** tab at the top of the list area. The Filters tab provides comprehensive tools to control exactly which animation events are, or are not, being displayed in the Event Log.



Notable controls include:

- **Filter Text Entry Add Button** - This drop list is also a text entry field that can be used to search the filter list for a given animation name. All of the animation names that the Debugger has recorded are displayed in the filter list. You can also enter a new animation name that has yet to be recorded by the Debugger and add it to the filter list with the Add button. As you enter text into this field a drop down menu with auto-complete suggestions will be displayed to speed the entry and search process.
- **Animation List** - This list displays all of the animation names that the Debugger is aware of along with check boxes that control if an animation event will be displayed. You can click on a check box to set or unset it thereby controlling whether or not that animation name is displayed in the Event Log.
- **Check All** - When this button is clicked all of the animation names in the filter list will be checked. When an animation name is checked it will be displayed in the Event Log.
- **Uncheck All** - When this button clicked all of the animation names in the filter list will be unchecked. When an animation name is unchecked it will not be displayed in the Event Log (but it will still be recorded for future display, filters allowing).
- **Check Selected** - The Filters list supports multi-selection behaviour where by you can click, shift+click, and ctrl+click, to select an arbitrary number of animation names. When the Check Selected button is pressed all of the animation names that are selected will become checked and, as a result, will be displayed in the Event Log.
- **Toggle Selected** - When this button is clicked the check state of all animation names selected in the Filters list will be toggled (i.e. checked animation names will be unchecked and unchecked animation names will become checked).

Filter options that you have defined are only applied to the Event Log when the **Apply Filters** toggle button is active. Toggling the Apply Filters button does not affect the state of the defined filters.

## 13.5 Watch List

The **Watch** list provides a different mode for monitoring animation events that occur in your design. Animations added to the Watch list are displayed in a table that shows their most recent value and the number of times ("hits") that animation event has occurred since the Debugger started recording events. There are three ways to add or remove animation names to the Watch list.

- **Event Log Context Menu** - You can right click on a row in the Event Log to raise the context menu from which you click **Add To Watch List** (or Remove From Watch List)
- **Filters Context Menu** - You can right click on a row in the Filters List to raise the context menu from which you click **Add To Watch List** (or Remove From Watch List)
- **Watch Animation Text Entry** - The **Watch Animation** text entry field operates much like the **Filter** text entry field. As you enter an animation name an autocomplete list is displayed that shows matching animations name to the text you've entered so far. You can either select a preexisting animation name or enter a new animation name that the Debugger hasn't recorded yet. Once you've entered an animation name click the **Add** button to add the animation to the Watch List.

The Watch Animation Text Entry field also provides "thumb pins" in the auto-complete drop down that you can use to pin an animation name to the top of the list. This is useful for selecting between commonly watched animation names quickly and easily.

Finally, the Watch List provides a context menu that can be used to either remove an animation from the list or to copy an animation name to the clipboard for use elsewhere in Altia Design. Simply right click on a row in the Watch List to display the context menu.

## 13.6 Saving the Debugger Log

The Debugger gives you the ability to save the current content of the **Event Log** to either a tab delimited text file (.txt) or a comma delimited text file (.csv). This is useful for capturing debug information, importing event data into an external tool like Microsoft Excel®, or sharing debug information with other users. The Debugger provides several different ways to save this file.

- **Event Log Context Menu** - You can right click on a row in the **Event Log** to raise the context menu from which you click **Save Event Log As...**. This will open a standard Windows Save File dialog where you can select the name of the file, location to save the file, and the extension (and thus format) of the log file (either .txt or .csv).
- **Filters Context Menu** - Just like the **Event Log Context Menu** you can right click in the **Filters** list and select the **Save Event Log As...** option from the context menu.

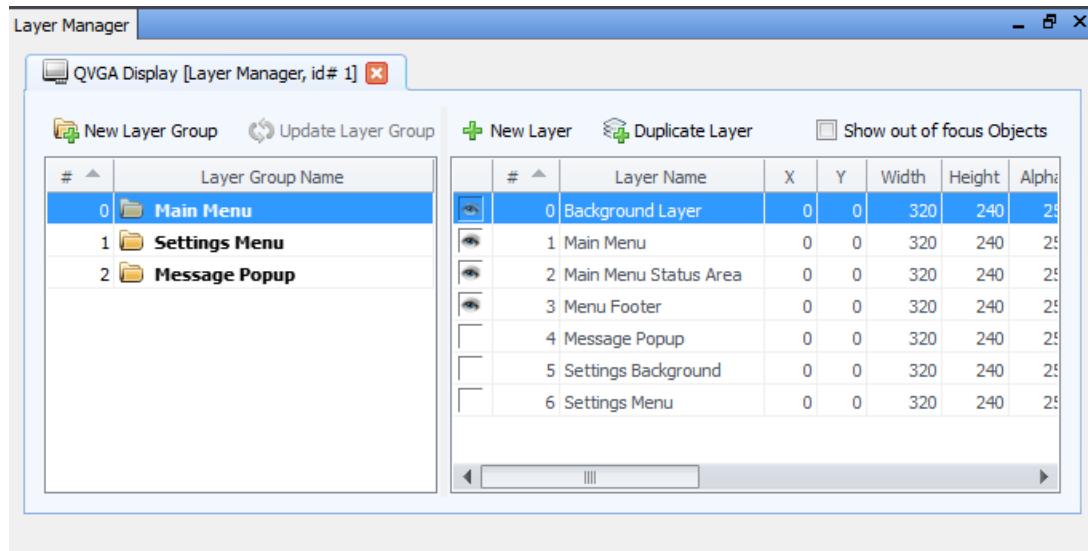
# Chapter 14: Layer Manager

Altia Design's **Layer Manager** pane is used in conjunction with **Layer Manager objects** that have been imported from the **Layer Manager library**. If one or more Layer Manager objects exist in your model, use the Layer Manager pane to create and manage its layers.

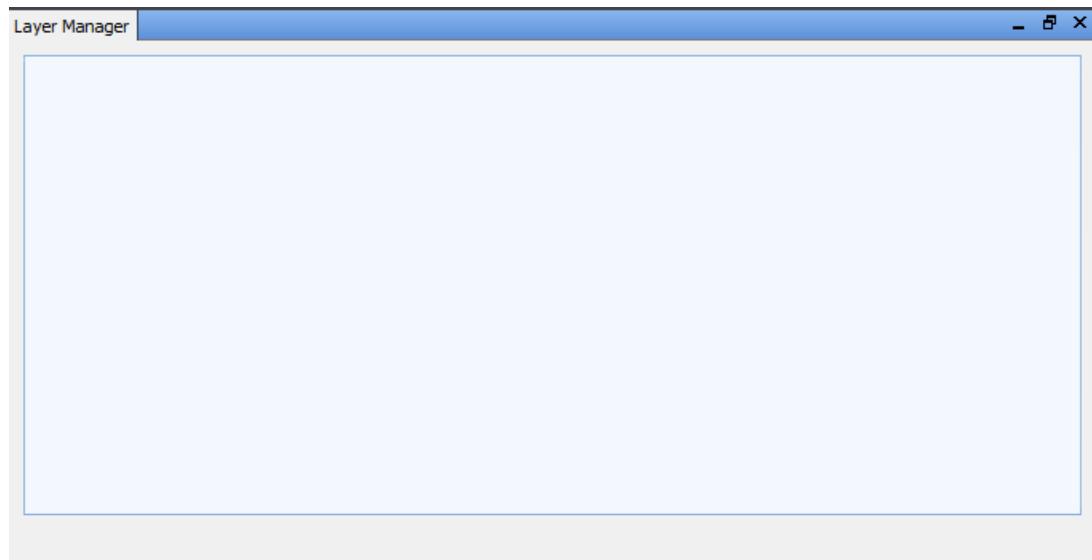
**NOTE:** The Layer Manager pane works best if you understand the underlying concepts of the Layer Manager object. It is suggested that you first review the documentation for the Layer Manager object in [Chapter 10, Section 10.5.6 – Layer Manager Object Model](#).

If the Layer Manager pane is not visible, you can turn it on by clicking the **Windows** dropdown button on the **View** ribbon. Ensure that the **Layer Manager** menu option is checked. Alternatively, if you select a Layer Manager object in the Altia universe, right-click and select **Layer Manager...** from the context menu to display the Layer Manager pane.

When a Layer Manager object is selected in the universe, the Layer Manager pane displays information about that Layer Manager object's defined **Layers** and **Layer Groups**. If there are multiple Layer Manager objects in your design, additional tabs will appear for each additional Layer Manager object that has been selected.



If a Layer Manager object does not exist within the current model, or if the Layer Manager object within the current model has not yet been selected, the Layer Manager pane will be empty.



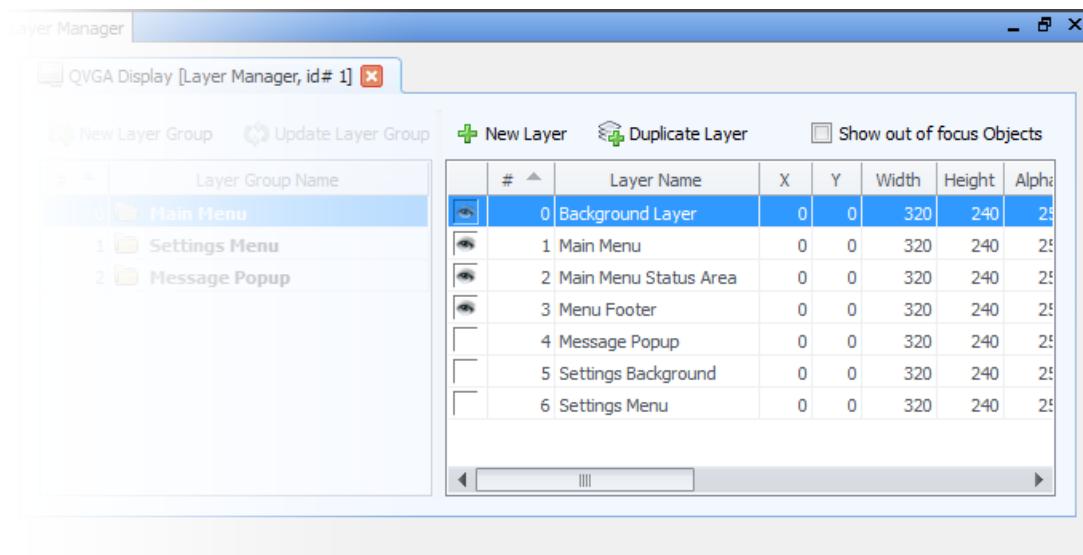
# 14.1 Layer Manager Pane Layout

The Layer Manager pane is divided into two sections:

- **Layer List**
- **Layer Group List**

## 14.1.1 Layer List

The Layer List displays the currently defined layers and their settings for the selected Layer Manager object .



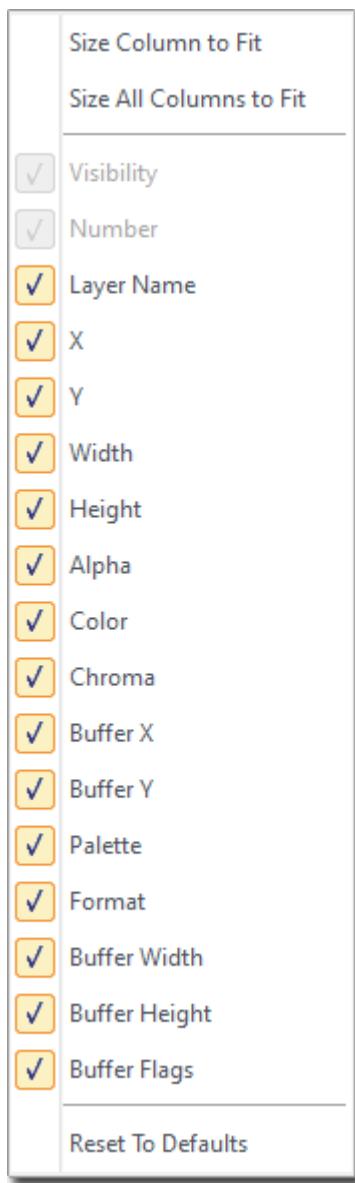
The screenshot shows the 'Layer Manager' window with the title 'QVGA Display [Layer Manager, id# 1]'. The window contains a toolbar with buttons for 'New Layer Group', 'Update Layer Group', 'New Layer', 'Duplicate Layer', and a checkbox for 'Show out of focus Objects'. On the left is a tree view showing a 'Main Menu' layer group containing 'Settings Menu' and 'Message Popup' sub-layers. On the right is a scrollable table listing six layers with columns for #, Layer Name, X, Y, Width, Height, and Alpha.

#	Layer Name	X	Y	Width	Height	Alpha
0	Background Layer	0	0	320	240	255
1	Main Menu	0	0	320	240	255
2	Main Menu Status Area	0	0	320	240	255
3	Menu Footer	0	0	320	240	255
4	Message Popup	0	0	320	240	255
5	Settings Background	0	0	320	240	255
6	Settings Menu	0	0	320	240	255

Each row of this scrollable list contains information about one layer within the selected Layer Manager object. Click on a cell within the selected row to modify the Layer's settings. A Layer's visibility can be turned on or off by toggling the  icon in the leftmost column. Layers can be assigned a Layer Name for easier identification.

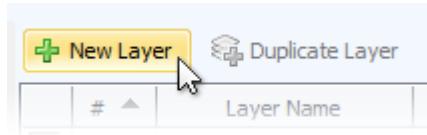
You can modify additional layer attributes (**X position**, **Y position**, **Width**, **Height**, **Alpha**, **Color**, **Chroma**, **Buffer X position**, **Buffer Y position**, **Palette**, **Layer Format**, **Buffer Width**, **Buffer Height** and **Buffer Flags**) within the selected row and the Layer Manager object in the Universe view will update in real-time. For more detailed information about these attributes, refer to [Chapter 10, Section 10.5.6, Layer Manager Animations](#).

Individual columns within the Layer list can be shown or hidden as desired by right-clicking on any column heading. On the menu that appears, check or uncheck the columns you wish to display within the list.



## Creating a new Layer

To create a new layer, press the **New Layer** button.

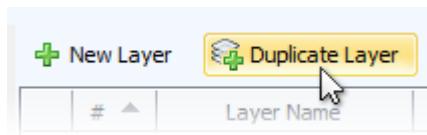


You may also manually create a new layer by using the Animation editor as described in [Chapter 10, Section 10.5.6, Adding Objects to a Layer in the Layer Manager](#).

When a new layer is created, Altia will automatically focus you in to the new Layer within the Layer Manager object. This allows you to quickly add objects inside the new layer.

## Duplicating a Layer

To duplicate an existing layer, select a layer row and then press the Duplicate Layer button. The selected layer, any objects the layer contains, and all layer attributes will be copied to a new layer with a new unique layer ID value.

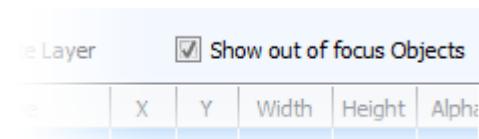


## Deleting a Layer

Highlight a layer row within the list and press the Delete (**DEL**) key or select **Delete Layer** from the selected row's right-click context menu.

## Show out of focus Objects

Like a Deck object, the Layer Manager object is a container. When focused into a layer within the Layer Manager, the contents of different layers within the Layer Manager are not visible. Sometimes it is desirable to see the contents of other layers while editing your model. To do this, check the **Show out of focus Objects** checkbox above the Layer List.

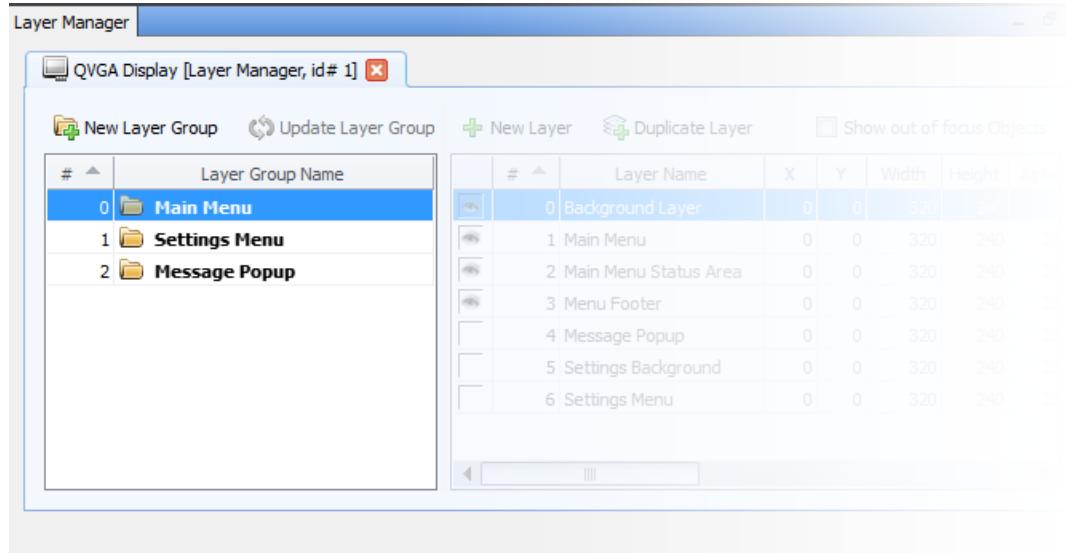


**NOTE:** You can also check Show out of focus Objects on the View ribbon or on the status area above the Universe view.

When **Show out of focus Objects** is active, while focused into a layer within the Layer Manager object, other layers within the same Layer Manager object will be visible but will appear dimmed.

## 14.1.2 Layer Group List

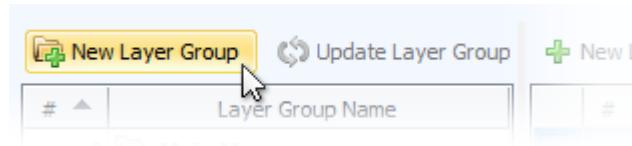
The Layer Group list displays any defined **Layer Groups**. Layer Groups allow you to store the state of separate layers and their attributes. This is often used to combine multiple layers into a screen within a UI.



Layer Groups can be activated within the Layer Manager pane by selecting a Layer Group row. Layer Groups can be programmatically activated by setting the `group_id` animation as described in [Chapter 10, Section 10.5.6, Layer Manager Animations](#).

## Creating a new Layer Group

To create a new Layer Group, press the **New Layer Group** button.

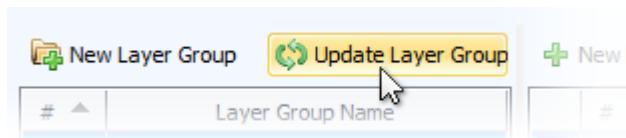


The visible state of all layers within the Layer List will be saved and applied to the Layer Group that is created. In addition, the state of all attributes for each visible layer will also be saved for columns within the layer list that have a white background.

**NOTE:** Some columns within the Layer List have a light-blue background color - the attributes within those cells will NOT be saved with the Layer Group. These attributes include Format, Buffer Width, Buffer Height and Buffer Flags.

## Updating a Layer Group

If changes to a previously defined Layer Group are desired, make any necessary changes to the appropriate Layer's attributes, then click the **Update Layer Group** button.



**NOTE:** Some columns within the Layer List have a light-blue background color - the attributes within those cells will NOT be saved with the Layer Group when Update Layer Group is clicked. These attributes include Format, Buffer Width, Buffer Height and Buffer Flags.

## Deleting a Layer Group

Highlight a layer row within the list and press the Delete (**DEL**) key or select **Delete Layer Group** from the right-click context menu.

### 14.1.3 Layer Manager Layer List Context Menu

Right click on one of the rows in the Layer List within the Layer Manager pane list to display the Layer List's right-click context menu.

Some of the options on the Layer List right-click context menu include:

- **Show / Hide Layer**
- **Show All Layers**
- **Hide All Layers**
- **Copy Layer Values**
- **Paste Layer Values**
- **Duplicate Layer**
- **Delete Layer**
- **Rename Layer**
- **Edit Layer Contents**

## Show / Hide Layer

This menu option will change based on the current visible state of the selected Layer row within the Layer Manager pane's Layer List.

If the highlighted Layer is visible, the context menu will display **Hide Layer**. Selecting Hide Layer will turn the selected Layer's visibility to not visible (off).

If the highlighted Layer is not visible, the context menu will display **Show Layer**. Selecting Show Layer will turn the selected Layer's visibility to visible (on).

## Show All Layers

Choose this option to set the visibility of all Layers within the Layer Pane's Layer List to visible (on).

## Hide All Layers

Choose this option to set the visibility of all Layers within the Layer Pane's Layer List to not visible (off).

## Copy Layer Values

Choose this option copy the selected Layer's attribute values to the copy buffer.

## Paste Layer Values

Choose this option to paste the copied Layer's attributes to the selected Layer.

## Duplicate Layer

This option duplicates the selected Layer. This works exactly as clicking the **Duplicate Layer** button described above.

## Delete Layer

This option deletes the selected Layer. This works exactly as the **Delete Layer** functionality described above.

## Rename Layers

Choose this option to rename the selected Layer.

## **Edit Layer Contents**

Choose this option to focus in to the selected Layer in the Universe view.

### **14.1.4 Layer Manager Layer Group List Context Menu**

Right click on one of the rows in the Layer Group List within the Layer Manager pane list to display the Layer Group List's right-click context menu.

Some of the options on the Layer Group List right-click context menu include:

- **Duplicate Layer Group**
- **Delete Layer Group**
- **Rename Layer Group**

#### **Duplicate Layer Group**

This option creates a duplicate of the selected Layer Group and assigns the new Layer Group a new unique layer group ID value.

#### **Delete Layer Group**

This option deletes the selected Layer Group. This works exactly as the **Delete Layer Group** functionality described above.

#### **Rename Layer Group**

Choose this option to rename the selected Layer Group.

# Appendix A - Keyboard Shortcuts

## Window Shortcuts

These shortcuts show the various editor windows in Altia Design.

**NOTE:** Altia Design must be in Edit Mode for these shortcuts to function.

Key Sequence	Description
Ctrl Shift A	Show Animation Editor.
Ctrl Shift C	Show Control Code Editor.
Ctrl Shift F	Show Find Editor.
Ctrl Shift I	Show Information Window.
Ctrl Shift N	Show Navigator.
Ctrl Shift P	Show Property Editor.
Ctrl Shift S	Show Stimulus Editor.
Ctrl Shift T	Show Connection Editor.
Ctrl Shift V	Show Tools Window.

## Tool Shortcuts

These shortcuts activate the different draw tools in Altia Design.

**NOTE:** Altia Design must be in Edit Mode for these shortcuts to function.

Key Sequence	Description
\	Activate Line Tool.
B	Activate Scale Tool (requires selection on canvas).

D	Activate Distort Tool (requires selection on canvas).
H	Activate Pan Tool.
L	Activate Ellipse Tool.
M	Activate Selection Tool (same as 'V').
Q	Activate Rectangle Tool.
R	Activate Rotate Tool (requires selection on canvas).
S	Activate Text Tool.
V	Activate Selection Tool (same as 'M').

## Editing Shortcuts

The following shortcuts are used for editing the graphical objects on the canvas.

**NOTE:** Altia Design must be in Edit Mode for these shortcuts to function.

Key Sequence	Description
Ctrl A	Select all objects in the drawing area.
Ctrl B	Toggle BOLD setting for fonts.
Ctrl C	Copy the selected objects to the Copy/Paste buffer (paste with Ctrl V).
Ctrl D	Deselect all objects in the drawing area.
Ctrl E	Focus In to the currently selected container object (i.e. group, deck, clip, etc.).
Ctrl F	Open the Find dialog.
Ctrl G	Put the selected objects into a Group.
Ctrl H	Open the Animation Rename dialog.
Ctrl I	Toggle ITALIC setting for fonts.

Ctrl J	Send the selected objects backward one level in the object stacking order.
Ctrl K	Bring the selected objects forward one level in the object stacking order.
Ctrl L	Show the Precise Scale dialog (requires selection on canvas).
Ctrl M	Open the Precise Move dialog (requires selection on canvas).
Ctrl N	Show the New Design dialog.
Ctrl O	Show the Open Design File dialog.
Ctrl Q	Snap the selected objects to the current drawing grid.
Ctrl R	Open the Precise Rotate dialog (requires selection on canvas).
Ctrl S	Save the design file. Will show the Save File dialog if the design has never been saved.
Ctrl U	Ungroup the selected container objects (i.e. groups, decks, clips, etc.).
Ctrl V	Paste copies of the objects currently in the Copy/Paste buffer.
Ctrl W	Focus Out of the currently focused container object (no action if already at the top level of the project).
Ctrl X	Cut the currently selected objects into the Copy/Paste buffer.
Ctrl Y	Redo the last undo.
Ctrl Z	Undo the last canvas operation.
DEL	Delete the currently selected objects (does not put objects into the Copy/Paste buffer).
INS	Paste copies of the objects currently in the Copy/Paste buffer (same as CTRL V).
Arrow Keys	Nudge the selected objects up, down, left, or right one pixel at a time.
Shift + Arrow Keys	Nudge the selected objects up, down, left, or right ten pixels at a time.

## Canvas Manipulation Shortcuts

These shortcuts are for manipulating the canvas in Altia Design. This includes switching the canvas between EDIT and RUN modes.

**NOTE:** Altia Design must be in Edit Mode for these shortcuts to function (except for Ctrl+Shift+E).

Key Sequence	Description
HOME	Center the view over the selected objects. If no objects are selected same as Ctrl HOME.
Ctrl HOME	Center canvas region in the display.
Ctrl +	Zoom in by a factor of 2x (also works with '=' key).
Ctrl -	Zoom out by a factor of 2x.
Ctrl + Arrow Keys	Move canvas region in the display.
Ctrl Shift E	Switch Altia Design to EDIT mode. Functions in RUN mode.
Ctrl Shift R	Switch Altia Design to RUN mode.

## Control Editor Shortcuts

These shortcuts are used to quickly insert control code statements.

**NOTE:** The Control Editor window must be visible with a text cursor in the code editing area.

Key Sequence	Description
Alt A	Add an AND statement to the end of an IF/LOOP.
Alt D	Add a CALL Routine statement.
Alt E	Add an END statement.
Alt G	Add a GLOBALS statement.
Alt I	Add an IF statement.

Alt K	Add a COMMENT (//) statement.
Alt L	Add an ELSE statement.
Alt M	Add a MATH statement.
Alt N	Add an ELSEIF statement.
Alt O	Add an OR statement to the end of an IF/LOOP.
Alt P	Add an EXPR statement.
Alt R	Add a ROUTINE block.
Alt S	Add a SET statement.
Alt T	Add a RETURN statement.
Alt U	Add a LOOP statement.
Alt W	Add a WHEN block.

# Appendix B - Altia Design File Settings and Altia Editor Settings

## B.1 Altia Design File Settings (.rtm files)

### The RTM File

Altia will load and save a .rtm file with each .dsn file. The .rtm file contains Altia Design settings that are specific to the associated .dsn file. Both files will have the same name by default. The .rtm file contains settings for:

- The canvas size and color
- The screen size and off-canvas color
- Grid settings
- Text scaling settings

### Sample .rtm file

```
! Altia Design Run-Time View Configuration File - Writable
!
! Autogen .rtm file - Copyright (c) 1991-2001 by Altia Incorporated
!
! This file is automatically generated whenever an Altia Design File
! is saved, reflecting the state of the Altia Graphics Editor's Main
! View. If present at run-time, the run-time view will adopt these
! attributes.
!
Altia*quitControlChar: d
Altia*quitFunction: altiaQuit
Altia*AltiaScene*foreground: #0000ff
Altia*AltiaScene*canvas: on
Altia*AltiaScene*background: #000000
Altia*AltiaScene*offscreen: #999999
Altia*AltiaScene*width: 480
Altia*AltiaScene*height: 272
Altia*AltiaScene*x: 0
Altia*AltiaScene*y: 0
Altia*AltiaScene*magnification: 1.00
Altia*routeCaching: 1
*BiDirectional: False
```

```

Altia*inputEditorFont: *-Arial Unicode MS-medium-r-normal--22-*-*-*
*--*
!
! GUI Settings
!
Altia*switchMode: Off
Altia*showFocusOnly: True
Altia*selectionOptional: False
Altia*grid_1: 1 1 pixel pixels
Altia*gridSnap: On
Altia*scaleAffectsText: False

```

## Managing RTM Files

It is not necessary to modify the .rtm file by hand because it will be written **every** time the .dsn file is saved in Altia.

The .rtm file is also loaded by Altia Runtime to configure the window appearance for the .dsn file to run.

**NOTE:** The .rtm file can be made "read only" by changing the first line. If the first line doesn't match the example above, Altia Design will not update the .rtm file when the .dsn file is saved. This is not recommended as Altia Design 11.x provides easy User Interfaces to manipulate the .rtm file settings.

## B.2 Altia Editor Settings (altia.ini file)

The Altia Design Editor stores default settings in the altia.ini file. This file is located in the **defaults** folder where Altia Design is installed.

The altia.ini file is well documented and explains each of the defaults that you can set. When you are finished editing the file, save and exit. The next time you start Altia Design, it will read the new defaults file and adjust the settings to your new preferences.

### Start-up Modes

Following are some of the many Editor properties that can be changed in the altia.ini file. Please see the comments in the file for more details on these settings (and many others).

- **Altia\*undoDepth** - depth of the undo stack (count of commands that can be undone).
- **Altia\*keepTool** - When set to "True" will keep the current tool enabled after using it.

- **Altia\*showFocusOnly** - when set to "True" only objects in the current focus will be shown. (Also saved in .rtm file)
- **Altia\*gridSnap** - when set to "True" it will snap all objects to the grid. (Also saved in .rtm file)
- **Altia\*gridShow** - when set to "True" it will show the grid on the canvas if the grid is larger than a pixel. (Also saved in .rtm file)

## Memory Settings

These settings determine how memory is allocated for various draw operations. The settings trade memory consumption for performance.

An off-screen pixmap is utilized during screen area redraws to minimize the flashing that commonly occurs. **OffScreenPixSize** determines the maximum size of the off-screen pixmap (its area in pixels). Areas smaller than the pixmap size are drawn to the pixmap, larger areas are drawn directly to the screen. A large off-screen pixmap can be slower to draw and will use up more display memory (a problem for X terminals). A small off-screen pixmap will cause more flashing during redraw operations. You can adjust the pixmap size for your system's configuration and performance by overriding the default size (supports a 1280x1024 area) with a new setting. NOTE: You must use **\*OffScreenPixSize**. **Altia\*OffScreenPixSize** doesn't work.

**\*OffScreenPixSize: 1310720**

Off-screen pixmaps are utilized for imported raster style images to improve performance during redraw. This is especially helpful if the image has been rotated or scaled after it was imported. The image can be transformed and permanently stored in an off-screen pixmap so that subsequent redraws don't require a recalculation of the image's pixels. **RasterPixSize** controls the maximum size allowed for these types of off-screen pixmaps. You can change the size to better suit your system's configuration and performance by overriding the default size (supports 1280x1024 image) with a new setting. NOTE: You must use **\*RasterPixSize**. **Altia\*RasterPixSize** doesn't work.

**\*RasterPixSize: 1310720**

# Appendix C - Altia Runtime

## C.1 Creating A Custom Altia Runtime

This section describes the steps necessary for creating a custom Altia Runtime application.

### Files Required for a Custom Runtime:

Several files supplied with the standard Altia Design installation must be part of the custom runtime:

- **altiart.exe** - The Altia Runtime executable (located in the **bin** folder).
- **fonts.ali** - The font name alias file (located in the **bin** folder). (OPTIONAL)
- **colors.ali** - The colors name alias file (located in the **bin** folder). (OPTIONAL)
- **sounds** - If your design uses Altia Sound Objects. (OPTIONAL)
- **altdde32.dll** - If you use a VB application to interface with Altia (located in the **lib** folder). (OPTIONAL)
- **Design File** - The design project (e.g., **demo.dsn**).
- **Runtime File** - The Altia Runtime configuration for project (e.g., **demo.rtm**).
- **Application File** - The application program to control the design (e.g., **demo.exe**). (OPTIONAL)

In this section, it is assumed that the custom runtime will be installed on a removable disk so the files listed above would simply be copied to the removable disk from a system containing the full Altia Design installation. The end-user would have the choice of executing directly off of the removable disk or copying the files on the removable disk to a hard disk directory and executing from there.

**NOTE: No codeword or license file is necessary in order to execute Altia Runtime.**

### Sound Objects

If your design uses the sound capabilities provided by Altia Design, you will need to include the sound files for the sounds you use. All standard sound files are found in the Altia **sound** directory. Copy the sound files you use to the same directory containing the other Runtime files; or, you can copy them into a sub-directory named **sound**.

## C.2 Executing A Custom Altia Runtime

To execute your custom Altia Runtime, certain command line parameters must be passed to **altiart.exe**. The parameters include the design file name (.dsn), optional runtime configuration file name (.rtm), and optional application executable file name.

Command Line Options Summary	
-file DESIGNFILE	Loads the design file (.dsn) into Altia Runtime. (Will also load a .rtm with same name if not specified otherwise).
-exec PROGRAM	Runs a separate program - usually an application controlling the Altia interface.
-defaults RTMFILE	Uses the specified runtime file to configure Altia Runtime.
-argfile ARGFILE	Specifies an optional text file of command line arguments to parse.
-port PORTSTRING	Specifies the port at which to open a listening socket. Default is ":5100" (without the quotes).

### Specify The Design File

Use the **-file** command line parameter to specify the the design file to run with Altia Runtime:

Example:

```
altiart.exe -file demo.dsn
```

### Specify The Runtime File (optional)

Altia Runtime will try to open a runtime configuration file with the same name as the design file. For demo.dsn, the associated demo.rtm will be opened. It's possible to override the default runtime file using the **-defaults** command line parameter.

Example:

```
altiart.exe -file demo.dsn -defaults mydefaults.rtm
```

## Specify The Application File (optional)

Altia Runtime can also start an application program to interface with the design. For demo.dsn, the associated demo.rtm will be opened. Use the **-exec** command line parameter to specify the application program.

Example:

```
altiart.exe -file demo.dsn -exec demo.exe
```

## Starting Altia Runtime From An Application Program

If your custom runtime has an application program, the program can start **altiart.exe** with the appropriate design file. This is accomplished very easily with the **AtStartInterface** function supplied with the Altia Design Application Programming Interface (API) library.

Example: An application program can start **altiart.exe** and load the design file auto.dsn by adding the following C-source lines near the beginning of the **main** function:

```
if (AtStartInterface("auto.dsn", 0L, 0, argc, argv)<0)
    exit(1);
```

Upon successful return from **AtStartInterface**, the application is ready to execute. If **AtOpenConnection** is already being used, then simply replace it with a call to **AtStartInterface** and make the necessary parameter changes.

**NOTE:** These functions are completely documented in the [Altia API Reference Manual](#).

Using this approach, it is only necessary for the end-user to double-click on the application program from a file browser directory list to execute the custom runtime.

## Starting Altia Runtime From A Batch Script

A **.bat** command script can initiate the execution of **altiart.exe** and the application program.

Example: A **go.bat** file for executing a design named **stress.dsn** with an application program **stress.exe** might contain:

```
altiart.exe -file stress.dsn -exec "stress.exe -retry 1"
```

## C.3 Customizing The Runtime Configuration File (.rtm)

As described in [Appendix 2](#) a .rtm file is written each time a design file is saved in Altia Design. The file contains runtime configuration information. The file is a simple text file so it is easy to edit it and make changes to existing configuration options or add new options. The file `\usr\altia\defaults\Altia.ini` file describes numerous available options.

If you customize a .rtm file, you can also change the Writable flag in the first line of the file to anything else (for example, Read-only) and the editor will not overwrite the file the next time you edit and save the design. This is not recommended as it will prevent modification of the settings in Altia Design.

## C.4 Terminating a Custom Altia Runtime

### Terminating a Runtime Session From Stimulus or Control

A runtime session can be closed by a stimulus definition that executes `altiaQuit` to a state value of 1. The session can also be closed by setting `altiaQuit` to 1 using control code:

Example:

```
SET altiaQuit 1
```

**NOTE:** The animation name that forces a quit is specified in the .rtm file.

### Terminating a Runtime Session From the Screen

A runtime session can be terminated by closing the runtime window or by pressing Ctrl+D when the cursor is in the runtime window.

**NOTE:** The Ctrl+D quit sequence is specified in the .rtm file.

### Terminating a Runtime Session Programmatically

A runtime session can be closed programmatically with an Altia library call of:

```
altiaSendEvent("altiaQuit",1)
```

or a toolkit call of:

```
AtSendEvent(aid, "altiaQuit",1)
```

The animation function that forces a quit, by default `altiaQuit`, is set by the `.rtm` file and can be changed by simply editing the `.rtm` file. Another option is to use the functions `altiaStopInterface` or `AtStopInterface`. These functions send `altiaQuit` and may perform additional operations in an attempt to ensure the termination of the interface.

To disable the termination of Altia Runtime by closing the runtime window, a program must select to receive `altiaCloseViewPending` events. If this is done, Altia Runtime will not respond to the close request. Instead, it will send an `altiaCloseViewPending` event to the program and let it decide how to respond. The value of the event is 0 when closing the main view. Otherwise it is the view number of a user-defined view (see the [Altia API Reference Manual](#) for more information on creating and manipulating user-defined views).

## C.5 Special Operations Using Altia Runtime

### Executing Control Blocks to Initialize a Design

To execute one or more control blocks when a design is opened in runtime or the editor, use a WHEN statement that reads WHEN `altialInitDesign == 0` for each control block. The Graphics Editor and Altia Runtime automatically generate an event with this name and a value of 0 when a design is opened. For the Graphics Editor, the event is generated each time a design file is opened using the Open... option from the File menu.

### Starting Runtime with a FullSize Window

If you include the option `-fullsize` in the command line when starting Altia Runtime, the runtime window will size itself to cover the entire display. The usual window banner at the top of the window will not appear, but you can still close the runtime task using <`alt-F4`> or `Ctrl+D`. You can also start in full size mode by placing the line: `Altia*fullSize:True` in your `.rtm` file.

### Making the Runtime Window Transparent

It is possible to make a specific color in your design file transparent, so that windows behind the Runtime window show through. For example, if there were a design with a blue box in it and blue was specified as the transparent color, then a movie playing in another window behind the Runtime window would show through where the blue box had been in the design. This is a handy way to overlay Altia graphics onto another window.

Setting which color will be transparent is done using the `.rtm` file. Put in a line similar to the following:

```
Altia*AltiaScene*transColor: 0 0 255
```

The value to the right is the RGB value of the color that should be made transparent. The RGB value can be entered as a triplet (0 0 255) or as a hex value (#0000ff).

For efficiency and flexibility, you can limit the area affected by the transColor feature by also adding a line like this to the .rtm file:

```
Altia*AltiaScene*transLimit: 300,300,600,600
```

The fields on the right describe a rectangular region to which the transparency will be limited. The fields represent left, bottom, right, and top (or **x1, y1, x2, y2**), respectively. The values for these coordinates are relative to the Altia design universe. Limiting the area that has to be searched for bits to make transparent can significantly improve performance. If the transColor option is used, then the use of transLimit is highly recommended.

For a special effect, the borders of the Altia Design or Altia Runtime window can be hidden when transColor is used. This is done by adding a line like the following to the .rtm file:

```
Altia*AltiaScene*transBorder: True
```

Be careful with this option. The borders are gone so Altia Runtime cannot be closed or moved the window border. Instead of using the Close (or "X") button to close the window, use the **Ctrl+D** key combination on the keyboard.

**NOTE:** If these changes are made to the .rtm file, then the Writable flag in the first line of the file should be changed to anything else (for example, Read-only). If this is not done, the editor will overwrite the .rtm file the next time the design is saved and the changes (to the .rtm) will be lost.

## UserView Naming from Control or a Client Application Program

User views opened with an OPEN\_VIEW control statement, or from an application program using the AtOpenView()/altiaOpenView() Altia API library calls, can be named. The name appears in the banner for the view window. To name a view using the OPEN\_VIEW control statement, enter the desired name, surrounded by double-quotes (" "), in the name field of the statement (to add an OPEN\_VIEW statement to a control block, choose the Control Editor's Add... button, select the Views option from the Control Dialog's Advanced menu, and then select OPEN\_VIEW from the Views Dialog). To name a view using AtOpenView()/altiaOpenView(), enter a double-quoted string for the instanceName parameter.

## Setting Initial States for Animations at Runtime

Control blocks, application programs, or stimulus definitions can execute an event to set all animations to their initial states. The event name to use is **altiaExecuteInitialStates** and it must have a value of 1.

Example using control code:

```
SET altiaExecuteInitialStates 1
```

Application programs can make Altia API library calls that look like:

```
altiaSendEvent("altiaExecuteInitialStates",1);  
or  
AtSendEvent(aid,"altiaExecuteInitialStates",1);
```

Also, stimulus can be defined to execute **altiaExecuteInitialStates** with a state of 1.

**NOTE:** The execution will not affect the appearance of text inputs or outputs, plots, strip charts, dynamic lines or polygons, LCD panels, or clips.