DeepScreen® User's Guide

Altia® Design 11.2 for Windows®

HMI Software

January 2015

1 Contents

1 Overview

1.1 About DeepScreen

Altia DeepScreen is the graphics code generation product that complements Altia Design.

DeepScreen generates C source code for the graphical objects in an Altia HMI **`.dsn`** design file. This code contains the necessary logic to draw the objects in their proper size, position, color, etc. and selectively refresh objects when portions of the screen require updating.

If objects have animation names and states, the logic for this behavior is generated as well. A simple programmatic interface into the generated code allows the changing of animation state values. As in the Altia Design and Altia Runtime desktop products, no direct manipulation of graphics code is required for exercising an object's dynamic behavior. The Altia animation name/value paradigm is alive and well in DeepScreen!

If objects have mouse/keyboard/touch stimulus and/or timer stimulus, the generated code can also create the logic for this behavior. A simple interface allows a target specific mouse, touchpad, and/or key event handler to pass information to this logic. It in turn translates the information into Altia name/value events just like you would expect from an Altia product.

Logic for Altia control can also be generated. This logic is converted directly to C code (versus the interpreted nature of control in Altia Design and Runtime).

At the time of this writing, DeepScreen supports full code generation for Windows desktops (for example, XP and Windows 7), Linux framebuffer devices, and a variety of embedded devices with graphics accelerators (for example, devices with OpenGL ES engines, OpenVG engines, as well as custom engines). This code generation includes complete interfaces to the graphics and event handling portions of the supported targets (for example, the Win32 API for Windows desktop systems). This means no target-specific code to draw or handle messages needs to be integrated by users targeting these platforms.

For users writing C application code that interfaces to Altia Design, and/or Altia Runtime using the Altia C/C++ API (see the ***Altia API Reference Manual*** for more details), DeepScreen has its own fully compatible version of the API. This allows for seamless integration of the C application code with DeepScreen generated code to create a single, high performance executable!

For a Windows desktop target, the generated code uses Win32 API calls for drawing to the display and Windows messaging functions to receive screen refresh, mouse and keyboard events. The source code does require some standard Windows libraries to link a fully functional executable (for example, the Windows **`user32.lib`** and **`gdi32.lib`** libraries). On other targets, linking with some standard system libraries is usually required to create a fully functional executable.

DeepScreen connections to other code generation products such as The MathWorks Real-Time Workshop, Stateflow Coder, and IBM Statemate Code Generator may be available as well.

Additional targets are constantly under development and consideration. Please contact your Altia sales representative to discuss your own requirements.

2 Installation Requirements

2.1 Install Altia Design and one or more DeepScreen Targets

Before you can generate DeepScreen code for a target device, you must install the Altia Design product and one or more DeepScreen products. Altia Design is the product in which an Altia HMI **`.dsn`** design file is created, modified, and even exercised in the unique **Run** mode of Altia Design.

When the Altia Design installer is executed, it provides a default destination folder similar to **`c:\usr\altia`**. In this document, references to **`c:\usr\altia`** are intended to refer to the Altia Design software installation.

When a DeepScreen target installer is executed, it provides a default destination folder similar to **`c:\usr\AltiaDeepScreen\<target_type>`** where **`<target_type>`** is unique for a target and its version (for example **`<target_type>`** of **`sw_renderwin32_542`** for the Intel x86 Software Render Windows target software version 5.4.2). In this document, references to **`c:\usr\AltiaDeepScreen\<target_type>`** are intended to refer to a DeepScreen target software installation.

You can remove/uninstall an Altia Design or DeepScreen software product installation from the Windows Control Panel's Add/Remove Programs dialog (Programs and Features dialog on Windows 7). Uninstalling a product does not remove any of your own files that you may have created in the Altia Design or DeepScreen software product folder unless you pasted over an existing file that was part of the installation. If you have a DeepScreen software product installation for which patches have been applied (internally or from Altia), please note that uninstalling the product will very likely remove some or all of the files that were patched.

2.2 Altia Design and DeepScreen Target Program Group Items

Installing Altia Design creates a program group with items to start the Altia Design editor and view documentation such as recent enhancements, product manuals, and tutorials. On Windows 7, an Altia Design program group is found under **Start > All Programs > Altia > Altia Design**.

Installing a DeepScreen target creates a program group with items to view documentation such as recent enhancements, product manuals, and tutorials. Actual DeepScreen code generation is initiated from the Altia Design editor. On Windows 7, a DeepScreen target program group is found at **Start > All Programs > Altia > Altia DeepScreen > <Target_Type> > <Target_Version>**. For example, **Start > All Programs > Altia > Altia DeepScreen > Intel x86 > Software Render Windows (v5.4.2)**.

2.3 A Typical DeepScreen Target Requires the Installation of Target Development Tools

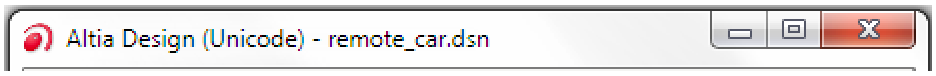
To compile the DeepScreen generated code requires compiler/linker tools. These required tools are typically unique to each type of target. For example, the Intel x86 Software Render Windows target requires an installation of Microsoft Visual Studio to compile/link the generated code.

Each DeepScreen target has its own unique User's Guide describing the required compiler/linker tools and any special configuration requirements. The separate target User's Guide appears in the program group for each DeepScreen target software installation. For example, **Start > All Programs > Altia > Altia DeepScreen > Intel x86 > Software Render Windows (v5.4.2) > User's Guide** for an installation of the Intel x86 Software Render Windows target version 5.4.2 on Windows 7.

3 Preparing to Generate DeepScreen Code

3.1 Start the Altia Design Editor. As of Altia Design 10.2, the only Type of Editor to Start is the Unicode Altia Design Editor

From an Altia Design program group under the Windows **Start > All Programs > Altia > Altia Design** menu, select the **Altia Editor** item to start the Altia Design editor. The **Altia Editor** item now refers to the Unicode version of the Altia Design editor. The Unicode version of the Altia Design editor is now the only available editor. It can open, display, and save all existing design (**.dsn**) files that were saved with a non-Unicode version of the Altia Design editor. You can recognize when a Unicode Altia Design editor is running because the banner at the top of the editor's window identifies it as a Unicode editor. The picture below shows the top banner for a Unicode Altia Design editor running with the design file **remote_car.dsn** opened.



To enable generation of Unicode (wide character) DeepScreen code, there is now a **Generate Unicode Font Characters** option in the Code Generation Options dialog. This option is explained in more detail later in the section *Select Code Gen Options*. If this option is not enabled, the generated code is for 8-bit character support like the generated code from the previously available non-Unicode versions of the Altia Design editor.

Do Not Open/Save Designs Containing High/Extended ASCII Text Characters or Unicode Text Characters in a Non-Unicode Altia Design Editor

A Unicode editor saves high/extended ASCII text characters (character values 128 to 255) and Unicode text characters (character values 256 to 65535) to a design (**.dsn**) file in a multi-byte format. This format is not recognized by any non-Unicode editors from earlier releases of Altia Design. A non-Unicode editor processes text characters only as single bytes. A non-Unicode editor will incorrectly display these special characters when it opens a design file that contains them. For example, it displays 2 or 3 unexpected byte characters for each special character. If the design is saved from a non-Unicode editor, the special characters are permanently replaced with the 2 or 3 unexpected single byte characters. The original special characters are only recovered by manually changing Label objects, Text I/O objects or Multi-line Text objects (MLTOs) to show the special characters instead of the 2 or 3 unexpected single byte characters.

3.2 You Must Have an Existing Design File Open

The design opened in Altia Design must already have a design (**.dsn**) file name associated with it before you can generate code for any of its objects. If it is a new design that has never been saved, it must be saved to a file before code can be generated.

3.3 Code is Generated to the Design File's Folder

Code is always generated to the folder containing the design **(.dsn**) file and a sub-folder named **altia** within that folder. If the **altia** sub-folder does not already exist, it is automatically created during the generation of code.

The folder containing the design (**.dsn**) file is referred to as the destination folder, destination directory, working folder or working directory in the rest of this document.

> **NOTE: All of the generated files have generic names within the design file's folder and the altia sub-folder so you should not attempt to use the same folder for generating code for more than one design file. If you do, the generated code files for one design file will overwrite the generated code files for any other design file residing in the same folder.**

3.4 If You Change Targets, You Must Generate New Code

A specific target (for example, **Intel x86 Software Render Windows**) is chosen at code generation time. The generated code is very unique to this target and not acceptable to compile/link/run on a different target.

3.5 Each Object's Current Visible State is its Initial State

The current visible state of an object when code is generated will be its initial visible state in the generated code. This implies that you want to have all of your objects in their desired initial states prior to generating code.

3.6 Each Object's Pixel Size is its Pixel Size in Generated Code

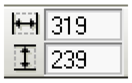
The pixel size of an object in the Altia Design drawing area is also the pixel size for the object in generated code.

To say it another way, objects will not adjust their sizes at generated code execution time based on the size of the target system's actual display. They will attempt to take up as many pixels on the target display as they take up in the Altia Design drawing area on the desktop display.

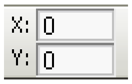
If the target display is too small, then portions of objects or entire objects will not be visible on the target display.

If your target display has specific size limits, a suggestion is to start out in Altia Design by drawing a rectangle that is the exact pixel width and height of the target display. To do this, you should have grid spacing set to 1 pixel in Altia Design. Grid spacing is set from the **View > Grid Spacing**... option. Its default setting is 1 pixel so there is normally no need to change it.

When you draw the rectangle, a status line of blue characters appears at the top of the drawing area giving you the width and height of the rectangle in pixels. Adjust your cursor position until the width and height are one less than your target system's display width and height in pixels. For example, if your target display is 320 x 240, draw the box to be 319 x 239. After drawing the rectangle, if it is not exactly the correct size, you can adjust it from the editor's Object width and height controls as shown in the picture below.



Set the color of the rectangle to be a color different than Altia Design's drawing area so that you can distinguish it from the rest of the drawing area. Send the rectangle to the back of the object stack with the **Send To Back** option in Altia's **Object > Arrange** menu and then position it as desired. For example, position its lower-left corner at 0,0 of the drawing area from the editor's Object X and Y controls.



If you place the objects that you will be generating code for on top of the visible rectangular area, then you know they will all fit on the target system's screen (because the rectangle should be the exact size of the target system's screen in pixels).

3.7 Selected Objects Determine Window Size

When generating code for a selected set of objects (this includes selecting all objects with the **Select All** tool), the default window size of the generated code will be the exact extent of the selected objects even if some objects are out of view or hidden at the time they are selected for code generation.

If the default window is too large for the target display, it is clipped from top to bottom and from right to left as required.

3.8 Editor Size is Window Size with No Selected Objects

If no objects are selected, code is generated for all objects, the default window size of the generated code is the size of the Altia Design drawing area at the time of code generation, and the objects will be identically positioned when the compiled code executes on a target system. Even though some objects may not be visible in this case, code is generated for them and they will be present in the final executable.

This technique of sizing the drawing area to the screen size of the target system is an alternative approach to drawing a rectangle in the design that is the screen size of the target system. This technique does require that you do most of the design work at a Zoom factor of 100% otherwise the size of the drawing area is not in sync with the magnification of objects in the drawing area (that is, 1 pixel of the drawing area is not 1 pixel of object content).

To make it easier to maintain the drawing area size, a good approach is to edit the **`.rtm`** file that is associated with a **`.dsn`** file to assign a specific size to the drawing area when the design file is opened. The **`.rtm`** file is a simple text file that can be edited with Windows Notepad. Here is example content of the **`.rtm`** file:

```
! Altia Design Run-Time View Configuration File - Writable
!
! 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*AltiaView*name:                   Main Altia View
Altia*AltiaScene*foreground:            #0000ff
Altia*AltiaScene*background:            #999999
Altia*AltiaScene*width:                 912
Altia*AltiaScene*height:                593
Altia*AltiaScene*x:                     0
Altia*AltiaScene*y:                     0
Altia*AltiaScene*magnification:         1.00
```

The first step is to set the **.rtm** file as read-only so that Altia Design will not overwrite it each time the associated .dsn file is saved. To set the **.rtm** file to read-only simply requires changing the first line of the file in some way. If the first line of the file is not exactly what Altia Design expects, it will not overwrite the **.rtm** file when it writes the **.dsn** file. For example:

```
! Altia Design Run-Time View Configuration File – NOT Writable
```

The next step is to adjust the width and height elements of the **.rtm** file to be the width and height of the target's screen size. For example, the following changes would set the width and height of the Altia Design drawing area to 320 x 240 pixels when the Zoom factor is 100%:

```
Altia*AltiaScene*width:                 320
Altia*AltiaScene*height:                240
```

Save the **.rtm** file changes.

Now, each time the associated **.dsn** file is opened using the Altia Design **File > Open…** menu option, the width and height information from the **.rtm** file will automatically adjust the width and height of the drawing area in Altia Design.

Please note that you must use the Altia Design **File > Open…** menu option for proper processing of the width and height information in the **.rtm** file. It does not work to double-click on the **.dsn** file from a Windows Explorer window. The **.dsn** file will open into Altia Design, but the **.rtm** file width and height information is not applied to the drawing area.

3.9 The Zoom Factor of Altia's Drawing Area May Be Applied

If you select no objects prior to generating code, the zoom factor of Altia's drawing area is applied to the generated code. The zoom factor is set from the **View > Zoom In** and **View > Zoom Out** menu options. If some or all objects are selected, the zoom factor is ignored.

Please note that drawing performance may be severely degraded on some targets if all objects must be drawn with a scale factor such as when the zoom factor is anything other than 100%. This is especially true for bitmap images (identified as Raster, Stencil, or Image objects in Altia) and text (Labels, Text I/Os, or Multi-line Text objects).

Some DeepScreen targets may not support the zoom factor at generated code execution time. Please see the target's separate User's Guide for possible restrictions.

3.10 Window Resizing on Target Windowing Systems

On a target system that has a window manager that allows resizing of windows (for example, the Microsoft Windows desktop), the size of a DeepScreen executable window may be adjustable just like any other window. Making the window size larger at execution time will reveal objects to the right and above the initial screen position. This is identical to the behavior seen with an Altia Runtime window or the drawing area in the Altia Design editor.

Support for window resizing is DeepScreen target dependent. Please see the target's separate User's Guide for possible restrictions.

3.11 Full Screen Mode Option

If you select the **Full Screen Mode** option in the Code Generation Options dialog (this dialog is described shortly), the window size will be the entire display size. The "anchor point" for the visible objects is the lower left corner of the window so the window grows taller and to the right to fill the entire screen.

The objects are not any larger or smaller in Full Screen Mode. Rather, the window simply takes up the entire display.

Beware of Full Screen Mode! The entire screen is taken over by the DeepScreen window. Depending on the target, this may cover up all other windowing system icons and task bars making it difficult to close the DeepScreen window. On a Windows Desktop, press **`Alt+Tab`** to bring another application to the front or **`Ctrl+Alt+Del`** to open the tasks list and select a new task or end the DeepScreen task.

Support for Full Screen Mode is DeepScreen target dependent. Please see the target's separate User's Guide for possible restrictions.

3.12 Design File Name Determines Window Name

On targets that display a caption with a window name (such as on the Windows desktop), the design (.**`dsn`**) file name without the .**`dsn`** extension is used as the window name and spaces (" ") in the name are replaced with underscores ("_").

3.13 Text Shaping

Text shaping is a new feature introduced with Altia Design 11.1. This feature uses source from the International Component for Unicode project (site.icu-project.org). The ICU source for shaping uses dynamic memory allocation (malloc and free). The "Use Dynamic Memory" option does not need to be checked at code generation time. However, the code will still invoke the memory calls in driver.c (xalloc, xfree, and xrealloc) as required.

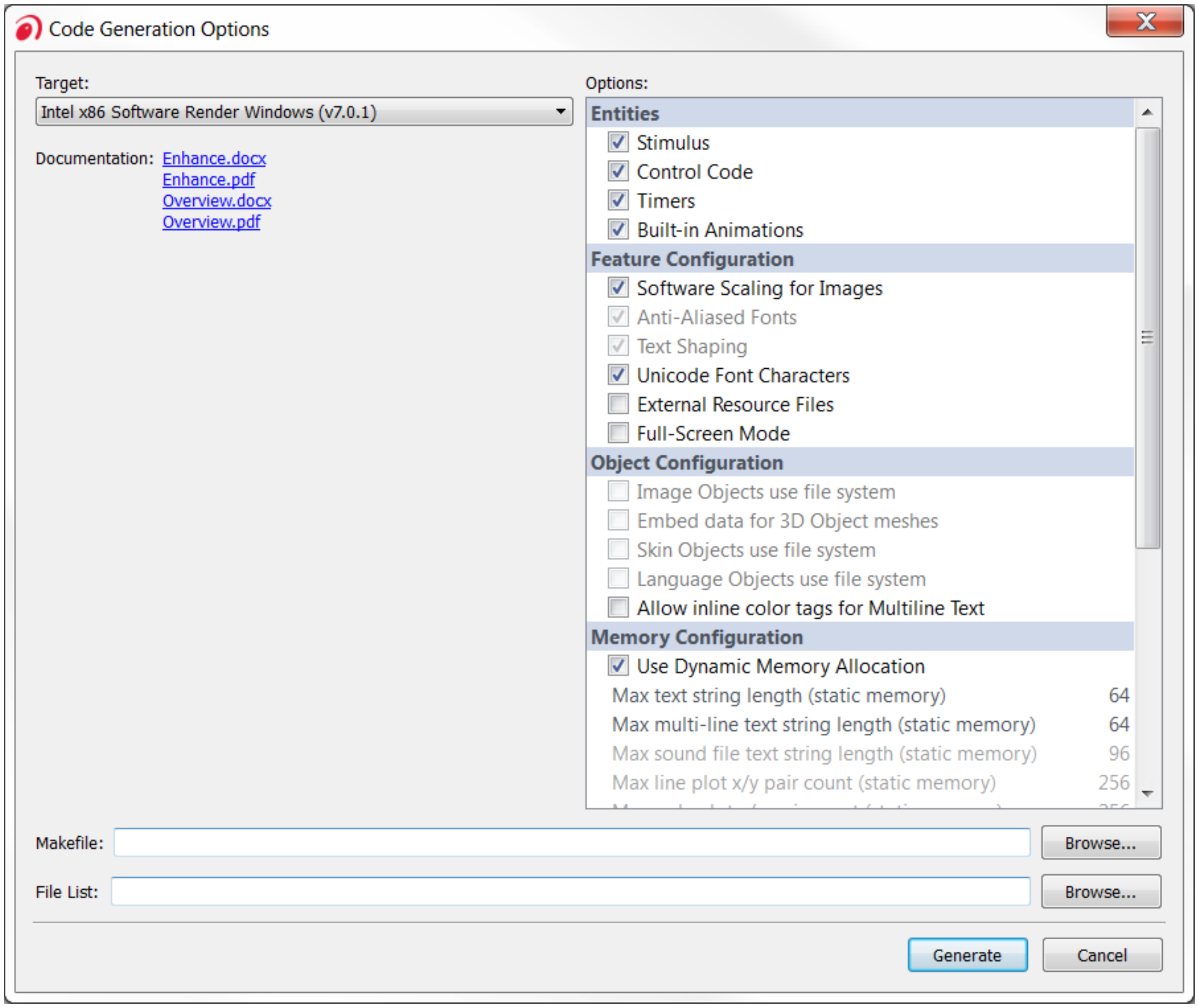
4 Generating DeepScreen Code

Select the objects you want in the generated code or select no objects to generate code for the entire design.

If an object is a Group, Deck or Clip (that is, an object that contains other objects), code is generated for the object itself and all of its children. For a Deck, this means that code is generated for every object in every card of the Deck independent of which card is currently being shown.

Choose the **Code Generation > Generate Source Code** menu option in Altia Design. This opens the Code Generation Options Dialog.

4.1 The Code Generation Options Dialog



Code Generation Options

Target: Intel x86 Software Render Windows (v7.0.1)

Documentation: Enhance.docx
Enhance.pdf
Overview.docx
Overview.pdf

Options:

Entities
- [✓] Stimulus
- [✓] Control Code
- [✓] Timers
- [✓] Built-in Animations

Feature Configuration
- [✓] Software Scaling for Images
- [✓] Anti-Aliased Fonts
- [✓] Text Shaping
- [✓] Unicode Font Characters
- [ ] External Resource Files
- [ ] Full-Screen Mode

Object Configuration
- [ ] Image Objects use file system
- [ ] Embed data for 3D Object meshes
- [ ] Skin Objects use file system
- [ ] Language Objects use file system
- [ ] Allow inline color tags for Multiline Text

Memory Configuration
- [✓] Use Dynamic Memory Allocation

Max text string length (static memory)	64
Max multi-line text string length (static memory)	64
Max sound file text string length (static memory)	96
Max line plot x/y pair count (static memory)	256

Makefile:
File List:

4.2 Choose a Target

In this dialog, choose a target from the **Target:** drop-down list. The available targets depend on the DeepScreen target software installations on the computer. In the example above, the available target is **Intel x86 Software Render Windows**. This target generates code that compiles with Visual Studio to create an executable that runs as a window on the Windows desktop.

4.3 Configure Code Generation Options

Each type of DeepScreen target may support some or all of the options available from the **Options:** list. Please see the target's separate User's Guide for possible restrictions. Here is a general description for each option:

4.3.1 Entities

- **Stimulus** - Check this option if you want the generated code to process mouse and keyboard stimulus for the selected objects. Such stimulus is created and viewed using the Altia Design Stimulus Editor. The target must have a mouse, keyboard, and/or touchscreen driver to support stimulus.
- **Control Control** - Check this option if you want code generated for any blocks of control associated with the selected objects. Control blocks are created and viewed using the Altia Design Control Editor. See the later section, *DeepScreen Supported for Control Elements*, for known restrictions.
- **Timers** - Check this option if you want code generated for any timer stimulus associated with the selected objects. Timer stimulus is created and viewed using the Altia Design Stimulus Editor. Many DeepScreen targets support timer stimulus if the DeepScreen generated code has access to a system clock.
- **Built-in Animations** - Check this option if you want code generated for dynamic built-in animations. See the later section, *DeepScreen Supported Built-In Animation Functions*, for a list of supported built-in animations. Non-dynamic built-in animations are always available regardless of this option.
    - Using this option will disable some memory optimizations (refer to Optimizations in Section 4.3.6).

4.3.2 Feature Configuration

- **Software Scaling for Images** - Check this option for Software Render targets if you wish to generate code for the algorithms that scale, stretch, rotate, and distort bitmap images (identified as Raster, Stencil, or Image objects in Altia) and text (Labels, Text I/Os, or Multi-line Text objects). By default, this option is enabled. For a DeepScreen target that uses an accelerated render engine (such as an OpenGL ES, OpenVG, or custom 2D target), this option is usually disabled (not checked). This type of target uses its accelerated render engine to do scaling, stretching, rotating, and distorting. As a result, it does not require the software algorithms to perform these operations. Please note that drawing performance may be severely degraded on some targets if bitmaps and/or text must be scaled, stretched, rotated and/or distorted. Please see the target's separate User's Guide for any restrictions.
- **Anti-Aliased fonts** – If there are text objects (Labels, Text I/Os, or Multi-line Text objects), check this option to generate the extra data and code to smooth the appearance of the text characters when they render. This requires extra memory on the target and uses more CPU cycles at execution time, but the visual results are usually worth the expense.
    - Starting with Altia Design 11.1, this feature will be disabled in the Options list.
    - Use the "Monochrome Fonts" option on the View Ribbon to change this setting.
- **Text Shaping** – If there are text objects (Labels, Text I/Os, or Multi-line Text objects), check this option to shape the text (bidirectional layout and glyph substitution). This will show

text in the proper left-to-right or right-to-left layout as required by the language. It will also perform glyph substitution to create connecting ligatures in languages like Arabic.

- See section 3.13 for memory requirements when using text shaping
- Use the "Text Shaping" option on the View Ribbon to change this setting.

- **Unicode Font Characters** – Check this option to generate text characters in wide format and generate the source code to manipulate these wide format characters. Unicode allows support for languages beyond western European such as Chinese and Japanese. Please see the target's separate User's Guide for details on whether or not a target supports code generation for Unicode font characters. Please also see the ***Unicode User's Guide*** in the Altia Design software program group for more details about working with Unicode fonts and Unicode string manipulation.
  - Altia recommends creating a custom .gen file when using this feature in order to specify font range flags for the fonts used in your .dsn project. Otherwise the generated code may be very large for international fonts.
- **External Resource Files** – Check this option to generate the data for bitmap images (identified as Raster, Stencil, or Image objects in Altia) and text (Labels, Text I/Os, or Multi-line Text objects) to external binary files named **`altiaDataBank[0-n].bin`** rather than as data structures in the generated source code. This reduces the size of the source code to be compiled and the size of the final executable code. The target, however, must have support for reading the binary files at execution time. For example, the target has a file system or there is support for separately loading the contents of the binary files into Flash memory. Please see the target's separate User's Guide for details on how a target handles external resource files or if it does not support them.
- **Full Screen Mode** - Check this option to request code generation for a window that is the entire size of the target system's screen. See the explanation in the earlier section *Full Screen Mode Option* for more details. This option is typically ignored for target's that do not have a windowing system.

4.3.3 Object Configuration

This section details object specific options used to configure the generated source code. An object setting can only be changed if the object is present in the .dsn used to generate code. For example, "Image Object use file system" will be grayed-out if your .dsn does not contain any Image Objects.

- **Image Object use file system** – Check this option to generate code to access image files for Image objects at execution time. If this option is not checked (which is the default), the image data displayed by an Image object at code generation time is output to the generated source files or, if **Generate external resource files** is checked, the image data is generated to the external binary files named **`altiaDataBank[0-n].bin`**. This code generation option is only supported by targets that run on systems that have a file system which can hold image files. The image files must be the PNG format (not JPG or BMP). Please see the later sections *Image Object Handling* and *Controlling Image Data Loading with Preload Options* for additional details. Also see the target's separate User's Guide for details on how a specific target handles files for Image objects or if it does not support them. If a target User's Guide does not describe how it handles files for Image objects and the target does not have a file system, it implies that the target does not support enabling this code generation option.
  - Only check this option if your target supports an embedded file system

- **Embed data for 3D Object meshes** – Check this option to embed the mesh data for models used in the 3D Scene object into the generated code. If this option is not checked, then the mesh data will be read from the embedded file system at runtime.
- **Skin Objects use file system** – When checked, Skin Objects will load their skin configuration data from the embedded file system at runtime. If not checked, the skin configuration file currently loaded into each Skin Object in your .dsn will be processed into a skin data source code file.
    - Only check this option if your target supports an embedded file system
- **Language Objects use file system** – When checked, Language Objects will load their configuration data from the embedded file system at runtime. If not checked, the language configuration file currently loaded into each Language Object in your .dsn will be processed into a language data source code file.
    - Only check this option if your target supports an embedded file system
- **Allow inline color tags for Multiline Text** – When checked the Multiline Text objects will process their text for inline HTML color tags.

4.3.4 Memory Configuration

- **Use Dynamic Memory** - If this option is checked, Text I/O objects (these are special objects that dynamically accept text or numbers), Multi-line Text objects (MLTOs for short, they dynamically accept multiple lines of text), plots, etc. will use dynamically allocated memory to store their contents rather than predefined fixed length arrays. Raster and Stencil objects will use dynamic memory to store temporary data for rendering, scaling, stretching or rotating. Please see the target's separate User's Guide for possible restrictions or recommendations on the use of this option.
- **Max Settings** – The remaining options in the Memory Configuration section specify the maximum sizes used for various objects in your .dsn. If an object is not present in your specific .dsn project, the size setting for that object will be disabled.
    - These options are only used when "Use Dynamic Memory" is disabled.

4.3.5 Process Configuration

- **Show Image Formatting Dialog** – Check this option to show an extra dialog at code generation time that details the processing of bitmap images (identified as Raster, Stencil, or Image objects in Altia) and text (Labels, Text I/Os, or Multi-line Text objects). It is recommended to have this option enabled.
- **Auto-close Image Formatting Dialog** – Check this option to automatically close the extra dialog that shows at code generation time with details of the processing of bitmap images and text. It is recommended not to check this option (that is, leave the dialog opened). Leaving the dialog open allows you to view the final results for the processing of bitmap images and text. Closing the dialog manually is easy from the dialog's **Close** button.
- **Treat missing images as warnings** – Check this option if Image objects not referencing valid image files should be treated as a warning rather than an error. It is good practice to start out with this option not checked. This allows you to identify if there are Image objects in the design that are missing image files.

4.3.6 Optimizations

This section provides a mechanism to disable optimizations performed by the DeepScreen Code Generator. Disabling any of these options is **not** recommended as it will adversely affect memory consumption on the embedded target.

Refer to Section 13 of this manual for further details on these DeepScreen optimizations.

- **Pack Data into Bit Fields** – Check this option to pack data into bit fields. This is performed on frequently used structures in data.c in order to minimize RAM and ROM consumption.
- **Remove Advanced Text I/O Animations** – Check this option to automatically remove unused, advanced animations from Text I/O Objects. This saves RAM and ROM for each Text I/O object in the generated code.
- **Remove Superfluous Groups** – Check this option to automatically remove groups that serve no functional purpose. Typically these groups are used to organize a .dsn for ease of understand and maintenance. Removing the superfluous groups saves RAM and ROM in the generated code.

Disabling these options will generate warnings during code generation. This is intentional to highlight that the generated code will not be optimum with regards to memory utilization.

4.4 Choose a Makefile script

From the **Makefile:** field, a template command line script is chosen. For example, this is a batch script to execute in a Windows Command Prompt window or a shell script to execute in a Linux shell. The script typically contains logic to configure the environment and invoke a Make file for compiling the generated source code.

A default template script name appears in this field, but it may not be appropriate for the selected target. Press the **Browse…** button to select a script available for the chosen target. The **Browse…** button displays a Windows file chooser dialog that shows the folder containing the template scripts available specifically for the chosen target.

When code is generated, this template is copied to the destination folder as **`altmake.bat`**. The destination folder contains the design (**`.dsn`**) file for which code is being generated.

If the target development tools support compiling from a command line batch or shell script, the **`altmake.bat`** script in the destination folder is available for this purpose.

For targets that support compiling the generated code from a Windows Command Prompt (such as the Intel x86 Software Render Windows target), the generated **`altmake.bat`** script is ready for execution after code generation. The Altia Design **Code Generation > Make Standalone** option is an easy way to execute the **`altmake.bat`** script in the destination folder after generating code.

Please see the target's separate User's Guide for details on the capability of the generated **`altmake.bat`** script and whether or not the target supports the Altia Design **Code Generation > Make Standalone** option for compiling the generated code.

4.5 Choose a Files List

Many targets support custom variations in the style or format of the generated code. The most common example of this is the format of the generated image data and text. The file specified in the **File List:** field typically controls these custom variations.

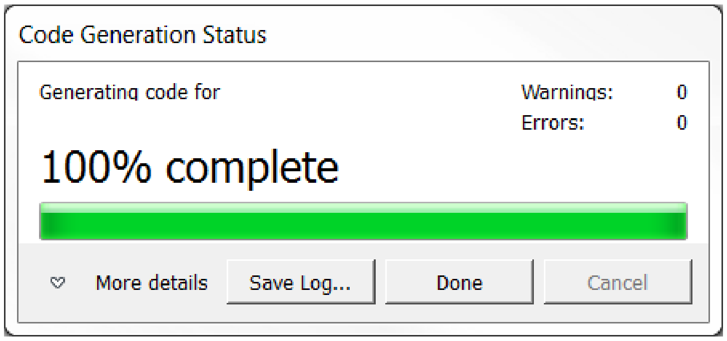
The file is referred to as the **.gen** file because it has a **.gen** file name extension.

This field initially appears empty. For most targets, it requires a file name. Press the **Browse…** button to select a **.gen** file for the chosen target. The **Browse…** button displays a Windows file chooser dialog showing the folder containing the **.gen** files available for the chosen target.

Please see the target's separate User's Guide for descriptions of the available **.gen** files for the target. Each target typically has a unique set of available **.gen** files.

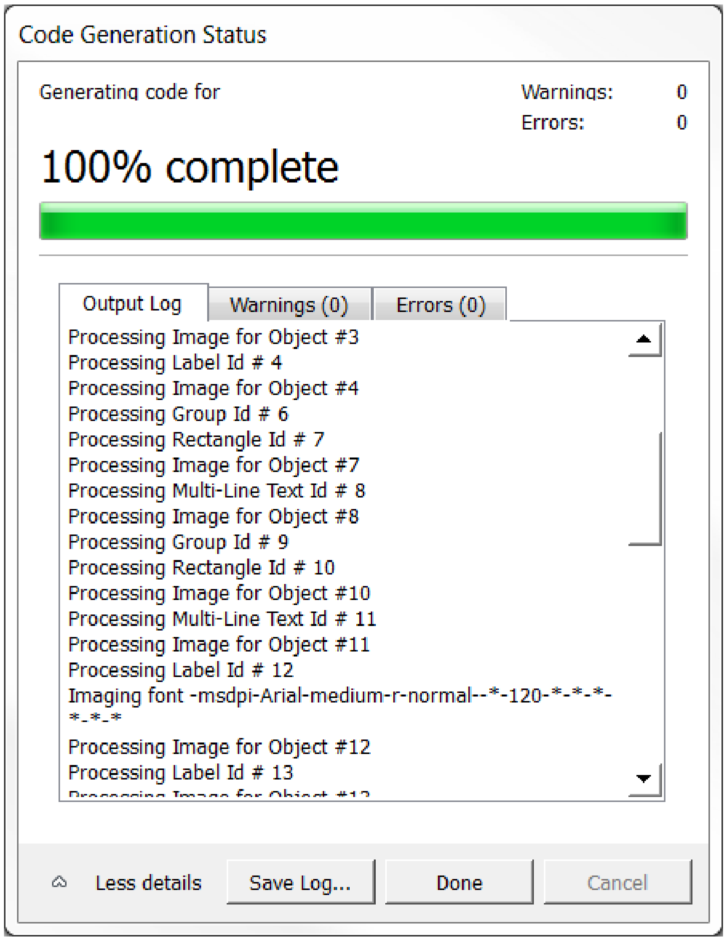
4.6 Press the "Generate" Button to Generate Code

When you are satisfied with your options, press the **Generate** button in the dialog to generate code or press the **Cancel** button to immediately exit the Code Generation Options dialog without saving any of your changes to the fields of the dialog. After pressing the button the Options dialog will disappear and the Status dialog will appear:



The code generation progress is displayed as well as the number of detected Warnings and Errors. When code generation is entirely finished, the Altia Code Generation Output Dialog will enable its **Done** button as pictured above.

Details for the generation process can be seen by clicking the **More Details** arrow at the bottom of the dialog. This will show additional tabs with the output log, warning summary, and error summary:



```
Processing Image for Object #3
Processing Label Id # 4
Processing Image for Object #4
Processing Group Id # 6
Processing Rectangle Id # 7
Processing Image for Object #7
Processing Multi-Line Text Id # 8
Processing Image for Object #8
Processing Group Id # 9
Processing Rectangle Id # 10
Processing Image for Object #10
Processing Multi-Line Text Id # 11
Processing Image for Object #11
Processing Label Id # 12
Imaging font -msdpi-Arial-medium-r-normal--*-120-*-*-*-
*-*-*
Processing Image for Object #12
Processing Label Id # 13
```

Press the **Done** button to exit code generation. Before pressing the button, you can take the opportunity to save the output log to a file for future reference. To save the log press the **Save Log…** button

4.7 Code Generation Error Codes

The table below summarizes the errors that can be encountered during code generation. Altia Design will display a final popup dialog with an error code number (as shown in the **Error Code** column).

Prior to Altia Design displaying a popup dialog with an error code number, the code generator may display its own popup dialog. This is identified as a **Popup dialog message** in the **Description and Message** column.

Alternatively, a message can appear in the Altia Code Generation Output Dialog (pictured above). This type of message is identified as an **Altia Code Generation Output Dialog message** in the **Description and Message** column below.

Finally, an Altia Design popup dialog can appear before the final popup dialog with an error code number appears. This is identified as an **Altia Design popup dialog** in the **Description and Message** column below.

An error identified as an **Internal Failure** is never expected, but crazier things have happened!

Error Code	Internal Failure	Description and Message
1	X	Unable to malloc 1.5MB "overflow" RAM buffer during startup No additional message: **This error is never expected. Contact Altia Support.**
1	X	Cannot load winsock.dll No additional message: **This error is never expected. Contact Altia Support.**
1	X	Cannot open script.ctl Popup dialog message: **Cannot find script.ctl file required for opening XML files**
3		No authorization to run product Popup dialog message: **No Authorization to run Altia Design**
4	X	No design file specified No additional message: **This error is never expected. Contact Altia Support.**
101	X	No code generation path specified No additional message: **This error is never expected. Contact Altia Support.**
102	X	Empty universe in .dsn file for code generation No additional message: **This error is never expected. Contact Altia Support.**
103	X	No behavior router in existence No additional message: **This error is never expected. Contact Altia Support.**
104	X	No top level scene in existence No additional message: **This error is never expected. Contact Altia Support.**
105	X	Invalid or missing object file No additional message: **This error is never expected. Contact Altia Support.**
106	X	Invalid data in object file No additional message: **This error is never expected. Contact Altia Support.**
107	X	Empty graphic set for code generation No additional message: **This error is never expected. Contact Altia Support.**
108	X	Invalid or missing animation file No additional message: **This error is never expected. Contact Altia Support.**
109	X	Invalid data in animation file No additional message: **This error is never expected. Contact Altia Support.**
1001		No authorization for DeepScreen Target Altia Code Generation Output Dialog message: **Cannot checkout license for %s**
1002		Syntax error with options in main .gen file (the base .gen file for the target – not a file chosen by the user) Altia Code Generation Output Dialog message: **Syntax error parsing .gen flags**
1003		Syntax error with options in extra .gen file (the .gen file chosen in the Code Generation Options dialog) Altia Code Generation Output Dialog message: **Syntax error parsing extra .gen flags**
1004		Cannot find the extra .gen file specified Altia Code Generation Output Dialog message: **Files not found %s**
1005		Either 1003 or 1004 occurred (may override 1003 or 1004) No additional message
1006		"Language Object Use Files" conflicts with other code generation options (i.e. Static Memory) Altia Code Generation Output Dialog message: **Dynamic Memory required for language obj to use files!**
1007		"Image Object Use Files" conflicts with other code generation options (i.e. Static Memory) Altia Code Generation Output Dialog message: **Dynamic Memory required for image obj to use files!**
1008		"Skin Object Use Files" conflicts with other code generation options (i.e. Static Memory) Altia Code Generation Output Dialog message: **Dynamic Memory required for skin obj to use files!**
1009		"Image Object Use Files" was not enabled and the .dsn has skin objects Altia Code Generation Output Dialog: **Loading Image Objects from files required when using Skin objects!**
1010		"Generate code for Built-ins" was not enabled and the .dsn has skin objects Altia Code Generation Output Dialog message: **Built-ins required when using Skin objects!**

Error Code	Internal Failure	Description and Message
1011		Unable to load the Image Formatter Engine DLL Altia Code Generation Output Dialog message: **Image Formatting Engine is missing!** Or **Image Formatting Engine is corrupted (E##)**
1012		Error pre-processing a specific object during code generation Altia Code Generation Output Dialog message: **Failed to process object/group #%d**
1013		Error pre-processing a specific object during code generation Altia Code Generation Output Dialog message: **Failed to process object/group #%d**
1014		Error processing a specific object during code generation Altia Code Generation Output Dialog message: **Failed to process object/group #%d**
1015		Error processing a specific object during code generation Altia Code Generation Output Dialog message: **Failed to process object/group #%d**
1016		Design has a Drawing Area Object but Dynamic Memory option was not enabled Altia Code Generation Output Dialog message: **Dynamic Memory required for Drawing Area Object!**
1017		Target is a JAVA type target but Dynamic Memory option was not enabled Altia Code Generation Output Dialog message: **Dynamic Memory required for Java!**
1018		Design is using Clones but Dynamic Memory option was not enabled Altia Code Generation Output Dialog message: **Dynamic Memory required for clones!**
1019		Design is using Snapshot Object but Dynamic Memory option was not enabled Altia Code Generation Output Dialog message: **Dynamic Memory required for Snapshot Objects!**
1020		Code Generator could not create temporary files required for code generation Altia Code Generation Output Dialog message: **Could not create a temp file name!**
1021		Syntax error with file list in main .gen file (the base .gen file for the target -- only one per target) Altia Code Generation Output Dialog message: **Syntax error parsing main .gen file**
1022		Syntax error with file list in extra .gen file (the .gen file chosen in the Code Generation Options dialog) Altia Code Generation Output Dialog message: **Syntax error parsing extra .gen file**
1023		Editor could not create temporary files required for code generation Altia Design popup dialog: **Unable to access file system!**
1024		Editor could not write data to temporary files required for code generation Altia Design popup dialog: **Unable to write design data!**
1025		Editor could not create temporary files required for code generation Altia Design popup dialog: **Unable to access file system!**
1026		Editor could not write data to temporary files required for code generation Altia Design popup dialog: **Unable to write view data!**
1027		Editor could not create temporary files required for code generation Altia Design popup dialog: **Unable to access file system!**
1028		Editor could not write data to temporary files required for code generation Altia Design popup dialog: **Unable to create animation data!**
1029		Editor could not create temporary files required for code generation Altia Design popup dialog: **Unable to access file system!**
1030		Editor could not write data to temporary files required for code generation Altia Design popup dialog: **Unable to create selection data!**
1031		"Multiline Text Object Use Tags" conflicts with other code generation options (i.e. Static Memory) Altia Code Generation Output Dialog: **Dynamic Memory required for using MLTO with tags enabled!**

5 After Generating DeepScreen Code

5.1 Compiling the Generated Code

Each target has its own special instructions for compiling the generated code into a working executable. Please see the target's separate User's Guide for these instructions.

For a target that supports compiling the generated code on the Windows desktop (such as the Intel x86 Software Render Windows target), it typically generates an **`altmake.bat`** script and **`altmake.mk`** Make file to the destination folder (that is, the folder containing the **`.dsn`** design file for which code was generated). In these situations, the Altia Design **Code Generation > Make Standalone** option is a handy feature. It automatically starts a Windows Command Prompt window and executes the generated **`altmake.bat`** script from the Command Prompt window.

The result of a successful execution of **Code Generation > Make Standalone** for the Intel x86 Software Render Windows target is a **`.exe`** executable file in the destination folder with the same base name as the **`.dsn`** design file. For example, **`my_design.exe`** for a design file named **`my_design.dsn`**.

The Altia Design **Code Generation > Make Clean** option executes the **`altmake.bat`** script with a command line option to remove intermediate object files. For the Intel x86 Software Render Windows target, this removes intermediate **`.obj`** and **`.lib`** files. The next **Code Generation > Make Standalone** operation recompiles all generated source files (rather than just recompiling the sources files that have changed).

The results of the Altia Design **Code Generation > Make with Editor API** option are target specific. Please see the target's separate User's Guide for details. If a target User's Guide does not describe the use of **Make with Editor API**, it is not supported by the target.

5.2 Running Generated Code after a Successful Compile

Each target has its own special instructions for executing the generated code after it has been compiled and linked into an executable file or files. For example, the executable file (and other support files) may need to be downloaded to the hardware using a target specific JTAG emulator. Please see the target's separate User's Guide for special instructions.

For a target that creates a Windows desktop executable (such as the Intel x86 Software Render Windows target), the Altia Design **Code Generation > Run on PC** option will run the executable if it has the same base name as the **`.dsn`** design file. For example, **`my_design.exe`** for a design file named **`my_design.dsn`**. Alternatively, use the Altia Design **Code Generation > Explore Working Directory** option to open a Windows Explorer to the destination folder and double-click on the **`.exe`** file to run it.

5.3 Viewing and Modifying the Generated Code

As noted earlier, the code is generated to the folder containing the **`.dsn`** design file currently opened in Altia Design. This is referred to as the destination or working folder. Use the Altia

Design **Code Generation > Explore Working Directory** option to easily open a Windows Explorer window to this folder.

After a successful code generation, you will minimally see a new **altmake.bat** script, often times a Make file named **altmake.mk**, and a sub-folder named **altia** containing generated source files. There may be additional generated files depending on the chosen code generation options and special requirements of the target.

This is a good time to emphasize that the **altmake.bat** script and **altmake.mk** file, if they exist, are always generated each time code is generated. Other source files are typically generated if they do not already exist or they exist, but are different from what the code generator is expecting.

> **Note: If the** altmake.bat **script,** altmake.mk **file, or a generated source file is modified by hand (such as from a text editor) in the destination folder, it will be overwritten the next time code is generated and any changes will be lost.**

A target may have specific files that it generates and will not overwrite on subsequent code generations. It may also have files that it expects the user will modify. Please see the target's separate User's Guide for any details on the target's special handling of generated files.

5.4 Customizing the Compile for a Target that has a Working altmake.bat

The **altmake.bat** file is a common file that a user may wish to modify for a target that supports compiling the generated code from the **altmake.bat** script (such as the Intel x86 Software Render Windows target). For example, it may be desirable to introduce additional compiler options before **altmake.bat** executes the generated **altmake.mk** Make file.

As explained in the previous section, **altmake.bat** is overwritten each time code is generated. Any changes to it will be lost (and not recoverable) the next time code is generated.

A suggested practice is to copy the generated **altmake.bat** to a different file name, customize the copy, and execute the copy to build the generated code. For example, after successfully generating code, copy **altmake.bat** to **altmake_custom.bat**. Edit **altmake_custom.bat** (for example, with Microsoft Notepad) and make the desired changes. Now execute **altmake_custom.bat** instead of **altmake.bat** to build the code. If **altmake_custom.bat** is a Windows batch script, double-click on it in a Windows Explorer window to execute it or open a Command Prompt window, change directory (**cd**) to the destination folder, and execute it in the Command Prompt window.

5.5 Integrating Custom Application Code with the Generated Code

Each target has its own special instructions for integrating the generated code with other application code and system code to create a fully functional target application. Please see the target's separate User's Guide for special instructions.

For a target that supports compiling the generated code on the Windows desktop (such as the Intel x86 Software Render Windows target), it typically generates an **altmake.bat** script and **altmake.mk** Make file to the destination folder (that is, the folder containing the **.dsn** design file for which code was generated). The Altia Design **Code Generation > Make Standalone** option will start a Windows Command Prompt window and compile the generated code into an executable.

By default, the executable has a simple **main()** function that loops on incoming Windows events, gives the DeepScreen generated code the opportunity to process the Windows events, and exits if the user closes the DeepScreen window.

The **altmake.bat** script and **altmake.mk** Make file are typically architected to also look for a file named **<design_name>.c** for a design file named **<design_name>.dsn**. The **altmake.bat** script assumes that this file contains a custom **main()** function that the user would like to substitute for the default **main()** function. This provides a method for easily introducing some custom application code. Perhaps not enough for developing a complete application, but the **<design_name.c>** file can be as large as desired and **#include** other header and source files.

To add a custom **main()** routine, simply create a file in the destination folder with a **.c** extension and the same base name as the Altia design (**.dsn**) file. For example, if the design file is **radio.dsn**, create a **radio.c** file. We will refer to this **.c** file as **<base_name>.c**. Edit **<base_name>.c** with your favorite text editor and start with something like the following as a simple test:

```
#include <altia.h>
int main(int argc, char *argv[])
{
    AtConnectId id;
    id = AtOpenConnection(NULL, NULL, argc, argv);
    while (AtPending(id) >= 0)
    {
        AtSendEvent(id, API_TEXT("needle"), (AltiaEventType) 0);
        AtFlushOutput();
        altiaSleep(2000);
        AtSendEvent(id, API_TEXT("needle"), (AltiaEventType) 10);
        AtFlushOutput(id);
        altiaSleep(2000);
    }
    AtCloseConnection(id);
    return 0;
}
```

If there is an Altia object with an animation named **needle**, this **main()** function will change the **needle** animation between states 0 and 10 every 2 seconds and it will not stop until the connection to Altia graphics is lost.

If **<base_name>.c** is compiled as an editor/runtime Altia API client application (and it very well can be), the connection is lost when Altia exits (that is Altia Design or Runtime) or when the Altia Design **Client > Disconnect All Clients** option is chosen if the connection is to Altia Design. When we say an editor/runtime Altia API client application, we mean compiled and linked with a DDE or TCP/IP socket version of the Altia API library to communicate with Altia Design or Altia Runtime. Please see the ***Altia API Reference Manual*** for more details.

The lost connection is detected by testing the return value from **AtPending()**. If it is less than 0, the program should finish. Actually, any Altia API call that consistently makes contact with Altia can be used in the **while()** statement, but **AtPending()** is the most reliable. For those readers who are Altia API experts, note that it isn't a problem to call **`AtPending()`** when we haven't actually selected any events. It will simply return 0 until the connection is dropped.

Now perform a **Make Stand Alone** from Altia's **Code Generation** menu after **`<base_name>.c`** exists and **`<base_name>.c`** and **`<base_name>.dsn`** reside in the same folder. In this case, **`<base_name>.c`** is automatically compiled into the DeepScreen executable under the assumption that it contains a custom **`main()`** function.

When this happens, the result is a DeepScreen standalone application that runs independently of Altia Design or Altia Runtime. In this situation, the **`AtPending()`** call isn't detecting a lost connection, but instead is going to return -1 if the window displaying the DeepScreen graphics gets closed. The concept is similar when you pause to think about it. Since the DeepScreen window is gone (like Altia closing when there is a client connected), the API calls start returning -1.

NOTE: Current API users are likely to remember that a call to `AtRetryCount(id, 1)` **normally forces the API to exit on its own when it detects a lost connection. This, however, is not true for DeepScreen (we apologize now for the inconsistency). Because the application is now directly managing a window, it is important that it exit by normal means (that is, by returning from** `main()` **like a good application should). For this reason, the DeepScreen API will not do an exit. Instead, the API functions return -1.**

This applies to all API functions except for `AtFlushOutput()` **and** `altiaFlushOutput()`**. These particular functions may return 0 because they detect that there is nothing to flush before they detect that the connection is dropped.**

For the **Code Generation > Make Standalone** option, a Command Prompt window displays the progress of compiling and linking the generated.

When compiling and linking finish, the last few lines displayed in the Command Prompt window should be studied to determine whether or not a DeepScreen executable was successfully created.

Problems usually arise because of syntax errors in **`<base_name>.c`**, improper configuration of the development environment, or the previously generated **`.exe`** file is still executing (the linker cannot overwrite a file that is currently executing).

If there is a syntax error in **`<base_name>.c`**, it will be shown during the compiling of the file `altiaUserMain.c`. BE SURE TO FIX ERRORS IN **`<base_name>.c`** AND NOT **`altiaUserMain.c`** BECAUSE **`altiaUserMain.c`** IS TEMPORARY.

For the Intel x86 Software Render Windows target, the generated **`altmake.bat`** script contains commands for copying **`<base_name>.c`** to **`altiaUserMain.c`** or copying **`altia\altiaSimplMain.c`** to **`altiaUserMain.c`** if **`<base_name>.c`** does not exist. This way, **`altiaUserMain.c`** always contains a **`main()`** function whether it is custom or the default.

For the Intel x86 Software Render Windows target, run the resulting executable after a successful **Code Generation > Make Standalone** by choosing the **Code Generation > Run On PC** option.

For the Intel x86 Software Render Windows target, the DeepScreen executable is a regular 32-bit Windows program that will run on Windows (for example, XP or Windows 7). It resides in the same folder as the generated code and design file. It has the same name as the design file with an **.exe** extension instead of **.dsn**. As you might guess, you can execute it from any Windows Explorer window by showing the contents of the folder and double-clicking on the **.exe** file.

5.6 Integrating the Generated Code into another Application

The DeepScreen generated code is architected to work with additional application code. The standard Altia API functions are used in the additional application code to send events into the DeepScreen code and listen for events from the DeepScreen code if code has been generated for stimulus, timer stimulus, and/or control.

Compiling the DeepScreen generated code typically results in libraries of object files for the DeepScreen graphics and DeepScreen version(s) of the Altia API.

For example, after generating and compiling code for the Intel x86 Software Render Windows target, the destination folder contains the new object libraries **altiaWinLib.lib**, **altiaAPIlib.lib** and **altiaAPIlibfloat.lib**. These object libraries are specific to the **.dsn** design file and target for which code was last generated.

Please see the target's separate User's Guide for instructions on how object libraries like these can be integrated into an existing project for the target.

6 Code Generation Considerations

6.1 Component Libraries That Are Compatible with Code Generation

The following component libraries (located in the Altia Design software installation **`models`** folder) contain objects that are supported by the DeepScreen code generator. For a miniGL target or a target based on miniGL, please see the target's separate User's Guide for the subset of supported objects).

<u>`Inputs:`</u>

`knobs.dsn`	(for target's with floating point support)
`inputs.dsn`	(for target's with key stimulus support)
`buttons.dsn`	
`keyboard.dsn`	
`sliders.dsn`	
`toggles.dsn`	

<u>`Outputs:`</u>

`meters.dsn`
`ledbars.dsn`
`leds.dsn`

<u>`Text:`</u>

`mlinetext.dsn`
`readouts.dsn`
`text.dsn`

<u>`Special Function:`</u>

`Blur Filter.dsn`	(on select OpenVG targets – see target User's Guide)
`Clip.dsn`	
`Decks.dsn`	
`Coverflow Horiz.dsn`	
`Coverflow Vert.dsn`	
`imageobjs.dsn`	(see later section *Image Object Handling* for details)
`Language.dsn`	
`Layer Manager.dsn`	(on select multiple display targets – see target User's Guide)
`Shapes.dsn`	
`Skin.dsn`	
`Snapshot.dsn`	(on select targets – see target User's Guide)
`Tickmarks.dsn`	
`Timers.dsn`	

<u>`Purchased:`</u>

`3D Scene.dsn`	(on select OpenGL ES 2.0 targets – see target User's Guide)
`Drawing Area.dsn`	

6.2 DeepScreen Unsupported Objects

Any model not listed in the previous section is unsupported in DeepScreen at the time of this writing. Some models may function on some targets – please refer to the target specific User's Guide for further details. The models listed below are not supported for any DeepScreen Target.

- The Sound object from **`Special Functions\sounds.dsn`** is not supported.
- The List object from **`demos\list\list.dsn`** is not supported. No support for this object is planned in the future.
- The Multi Strip object from **`Plots and Charts\Stripcharts.dsn`** is not supported.
- No objects from **`Special Functions\Views.dsn`** are supported. DeepScreen targets do not generally open multiple windows. If there is an exception, it will be documented in the target's separate User's Guide.

6.3 DeepScreen Support for Control Elements

Control is the scripting environment within Altia Design (open the Control Editor from the **Control** button in Altia Design to add or modify the Control logic associated with an object). At DeepScreen code generation time, if the **Generate code for Control** option is chosen, Control logic is converted to actual source code.

This section describes special issues to be aware of for DeepScreen code generation of Control code.

A miniGL target or a target based on miniGL may have more restrictions on the types of Control elements for which it can generate code. Please see the miniGL target's separate User's Guide for a description of its support for generation of Control code.

6.3.1 DeepScreen Code Generation for Control Variable Arrays has Restrictions

DeepScreen code generation supports using Control variables as arrays. In earlier releases of DeepScreen, Control variables (globals or routine locals) used as arrays generated code that could not compile. This release supports generating code for Control variables used as arrays if the variable is given an array size when it is declared in a **GLOBALS** statement or the **LOCALS** section of a **ROUTINE** statement as in:

```
GLOBALS     my_array[20]
```

or:

```
ROUTINE     my_routine
PARAMS:     my_param     LOCALS:   local_array[100]
```

The array size must be a number, not a Control variable name or an animation name. If an array's elements are used as strings, the array becomes an array of string elements at code generation time and the default length of each string element is 64 characters. This allows for a string of 63 printable characters followed by a nil terminator character in each element of the array. This length is adjustable from the Code Generation Options window by changing the setting for:

Size Settings if no Dynamic Memory:
Maximum number of characters in dynamic strings `100`

The setting (100 in this example) is used for the character length of each array element even if the **Use Dynamic Memory** option is selected. In other words, the character length of array elements is fixed to one size for all arrays whose elements are used as strings. Finally, if one element of an array is used as a string, all elements must be used as strings. Failure to use all elements of a specific array consistently will most likely generate code in **`altia\control.c`** that does not compile or if it compiles, it probably will fail on execution.

If array elements will contain strings and these strings are only set through indirection as in:

```
EXPR  my_array[i]  =  @temp_name
```

There must be at least one **SET** or **EXPR** statement that sets an array element directly to a string (preferably in a **WHEN** block for **`altiaInitDesign`**) as in:

```
GLOBALS    my_array[20]
WHEN  altiaInitDesign  =  {any value}
      SET  my_array[0]  =  "a string "
END
```

Otherwise, the DeepScreen code generator is not able to recognize that **`my_array[20]`** (in this example) is for string elements and it assumes the elements will hold numbers instead of strings.

6.3.2 No Support for Code Generation of Views Statements

Use of Control **Advanced > Views** statements are not supported unless a target's separate User's Guide specifically identifies support for these statements.

If these statements exist in an object's control and code is generated for the object, the code will not compile because of syntax errors. The errors will occur while compiling **`altia\control.c`**.

6.3.3 Limited Support for the EXTERN Statement

There is limited support for the Control **Advanced > Miscellaneous EXTERN** statement.

DeepScreen will simply place a call to the specified function (including parameters) in the code. It is up to the user to make sure that the function is resolved by some code (for example, by some code in **`<base_name>.c`** file if the **`altmake.bat`** script supports compiling extra application code as **`<base_name.c>`** for code generated from a design file named **`<base_name>.dsn`**).

If the function cannot be found, the code will not link because of unresolved external symbols.

6.3.4 Integer Arithmetic and the EXPR Statement

When DeepScreen generates code for the **EXPR** statement in Control, it casts or creates all numbers as type double (or integer if fixed point code is generated). Division operations are therefore performed with floating point precision by default (or integer precision if fixed point code is generated). In contrast, Altia Design and Runtime assume all numbers are integer unless they have decimal digits. Division in an **EXPR** statement is performed with integer precision by default in Altia Design and Runtime.

Because DeepScreen performs division in an **EXPR** statement with floating point precision by default (unless fixed point code is generated), the results can have decimal digits and this may yield an unexpected result (for example, `12.125` instead of just `12`).

The solution is to use the **`$int()`** function in the **EXPR** statement to force operands to be integer. For example:

```
EXPR  result
=     $int(a)   /   $int(b)
```

6.3.5 No Support for GLOBALS that Change their Types Dynamically

The code generator determines variable types for Control **GLOBALS** based on how they are used. Whereas a **GLOBALS** variable in Altia (Design or Runtime) can hold an integer at one point in time and a string at another point in time, such a variable for DeepScreen generated code must have a consistent type. The type is determined by the first use case encountered by the code generator. Altia control that uses a **GLOBALS** variable as more than one type will result in generated code that may crash at execution time.

For a GLOBALS variable being used to hold strings, a good practice is to set the variable to a string constant in a **WHEN** block for **`altiaInitDesign`**. The code generator will easily detect this usage and associate the variable with a string. For example:

```
GLOBALS my_global
WHEN  altiaInitDesign   =   {any value}
      SET my_global   =   "a string"
END
```

6.4 Do Not Use Special Characters in Animation, Stimulus, or Control Names

For DeepScreen code generation, Animation, Stimulus, and Control names can only contain lower ASCII characters (0 to 127 character values) that are regular alphabetical characters (a-z, A-Z), number characters (0-9), or the underscore (_) character. Do **not** use symbols (e.g., : ; %), spaces, tabs, high/extended ASCII characters (128 to 255 character values) or Unicode characters (256 to 65535 character values) in Animation, Stimulus or Control names. These symbol, space, tab, high/extended ASCII, and Unicode characters will nearly always cause compiler errors because Animation, Stimulus, and Control names may be used as source code variable names.

Also do not use symbols, space, tab, high/extended ASCII, or Unicode characters in the name of the design (**`.dsn`**) file because the design file name is also incorporated into variable names of the source code.

6.5 DeepScreen Supported Built-In Animation Functions

DeepScreen supports the following built-in animation functions:

`altiaCacheOutput`

`altiaColorBgObj`

`altiaColorFgObj`

`altiaDisableOnlyInput`

`altiaEnableOnlyInput`

`altiaErrorOutput`

altiaGetObjWH

altiaGetObjXY

altiaInitDesign

altiaMoveObjX

altiaMoveObjY

altiaPopToTop

altiaUnpop

altiaQuit

altiaSetBrush

altiaSetFont

altiaSetImage

altiaSetObj

altiaSetOpacity

DeepScreen does not currently support any other built-in animation functions. For a description of these and other built-in animation functions, see the ***Altia Design User's Guide***, ***Animation Editor*** chapter, ***Special Built-In Functions***. Some built-in animations may be too new to appear in the ***Altia Design User's Guide***. Please also see the ***Altia Design Enhancements Summary***.

This correctly implies that the Altia API routines that use other built-in animation functions for opening and manipulating views and designs (see Chapter 3 of the ***Altia API Reference Manual***), creating and manipulating clones (see Chapter 4 of the ***Altia API Reference Manual***) and widget integration (see Chapter 5 of the ***Altia API Reference Manual***) are not supported unless a target's separate User's Guide specifically identifies support for one or more of these capabilities.

6.5.1 Enabling Built-in Functions In DeepScreen Code

Some built-in functions are always available, regardless of code generation options. These built-in functions are:

altiaCacheOutput

altiaDisableOnlyInput

altiaEnableOnlyInput

altiaErrorOutput

altiaGetObjWH

altiaGetObjXY

altiaInitDesign

altiaQuit

altiaSetImage

altiaSetObj

These built-in functions require few resources at runtime and are therefore available regardless of code generation options.

The remaining built-in functions are only available when the **Built-in Animations** checkbox in the Code Generation Options dialog is checked. The following built-ins require extra resources as well as dynamic states for all objects (color, opacity, position, etc):

```
altiaColorBgObj
altiaColorFgObj
altiaMoveObjX
altiaMoveObjY
altiaPopToTop
altiaUnpop
altiaSetBrush
altiaSetFont
altiaSetOpacity
```

Because these built-ins require object data to reside in RAM (instead of flash), they are only enabled when the **Built-in Animations** checkbox checked. In addition, object data cannot be optimized based upon the usage in the .dsn file because the code must protect for any usage via the built-in functions above. For the smallest RAM consumption on the target hardware, Altia recommends **not** using these built-in functions in the DeepScreen generated code (i.e. do not check the **Built-in Animations** checkbox).

6.6 DeepScreen Version of C/C++ API Limitations and Additions

6.6.1 DeepScreen Version of C/C++ API Does Not Support View and Design File Manipulation Routines, Clones Routines and Widget Integration Routines

The DeepScreen version of the API does NOT support routines for opening and manipulating views and designs (see Chapter 3 of the ***Altia API Reference Manual***), creating and manipulating clones (see Chapter 4 of the ***Altia API Reference Manual***) and widget integration (see Chapter 5 of the ***Altia API Reference Manual***) unless a target's separate User's Guide specifically identifies support for one or more of these capabilities.

6.6.2 Calling Altia API from Multiple Threads is Supported for Windows and Possibly other Targets

Generated code for Windows and some other popular operating systems may support calling Altia API functions from multiple threads. The generated code has hooks to protect critical sections of the DeepScreen code during calls to any Altia API functions. For example, CriticalSection API calls are used on Windows. On VxWorks, Semaphore API calls are used. This allows multiple threads (or they are referred to as tasks in some operating systems) to call Altia API functions. If one thread is already calling an Altia API function, other threads will block on any Altia API function call until the first thread completes its call. Support for calling Altia API functions from multiple threads is target dependent. Please see the target's separate User's Guide for any restrictions.

Despite this wonderful addition, there are still issues with calling Altia API functions from multiple threads on Windows (such as for code generated from the Intel x86 Software Render Windows target).

The Windows thread that starts DeepScreen (that is, causes it to open its window) is the only thread that can get new events from DeepScreen. It should be the only thread that calls **AtNextEvent()** or **altiaNextEvent()**. If other threads call this function, they will block forever waiting for events because events from the DeepScreen window only go to the thread that opened the DeepScreen window. If a thread blocks in this manner, any other thread that makes an Altia API call will also block forever.

On Windows, the Altia API utility function **altiaSleep()** protects itself with the same critical section calls as the rest of the Altia API. This is necessary because it must process messages from Windows and these messages can potentially activate the DeepScreen code. This makes **altiaSleep()** less useful because no other threads can use the Altia API while an **altiaSleep()** is active. To allow a thread to sleep a long time (say, 1 second), still catch messages and not block other Altia API calls, code like the following is required:

```
for(i = 0; i < 25; i++)
{
      altiaSleep(20);
      Sleep(20);
}
```

The **Sleep()** call is from Windows and requires a **#include <windows.h>. Sleep(20)** does a hard sleep for 20 milliseconds, which lets other threads execute. The **altiaSleep(20)** call, on the other hand, takes the critical section (thereby blocking out all other threads that want to use Altia API calls), creates a **WM_TIMER** message and uses the Windows **GetMessage()** function to process incoming events and simultaneously wait for the timer to expire.

To use the Altia API from multiple threads to communicate with Altia Design or Altia Runtime on Windows, application code can now link with a float or integer version of the socket API library that supports multi-threaded applications. The multi-threaded float Altia API library is located in **c:\usr\altia\libfloat\ms32\liblanmt.lib** and the integer version is in **c:\usr\altia\lib\ms32\liblanmt.lib**.

DDE versions of the Altia API libraries that support multiple threads are not available due to technical problems with communicating over a single DDE from multiple threads. A DDE uses Windows messages. Similar to the restrictions with getting messages from a DeepScreen window, only the thread that creates the DDE can send/receive messages to/from the DDE.

6.6.3 Using Animation IDs

Animation IDs are enabled by default for Altia DeepScreen. For DeepScreen target releases before August 2013, the pre-processor macro **AltiaUseAnimationIds** had to be defined to use these functions (that is, it had to be defined when compiling DeepScreen generated code and application code that did a **#include** of **altia.h** and **AltiaAnimationIdTable.h**) For DeepScreen target releases after August 2013, defining **AltiaUseAnimationIds** is unnecessary because Animation IDs are enabled by default. This allows application code to use numerical IDs in place of text strings when specifying animations for the event and text APIs. New ID based APIs exist for this purpose:

- **altiaSendEventId**
- **AtSendEventId**

- **altiaSendTextId**
- **AtSendTextId**

These APIs are exactly the same as the text-based APIs except that the event parameter is a numerical (int) ID.

The ID for an animation can be obtained using the generated header file **AltiaAnimationIdTable.h** located in the **altia** folder where code was generated.

The ID for an animation can also be obtained at runtime using the **altiaGetAnimationID()** API available only in Altia DeepScreen. This API takes a text-based animation name and returns the associated numerical ID. A return code of -1 indicates that the specified animation name does not exist in the generated code.

These functions are only supported in the DeepScreen version of the Altia API. These functions are not supported in the client (socket or DDE) version of the Altia API.

If an application is going to be compiled as an Altia client (with a socket or DDE API) sometimes and as a DeepScreen application (with the DeepScreen version of the API) other times, you can conditionally compile these functions for the DeepScreen case by defining a macro (**DEEPSCREEN** is the recommended macro) such as:

```
#ifdef DEEPSCREEN
...
#endif DEEPSCREEN
```

As an alternative for creating the best common application code to compile with either a client Altia API library or DeepScreen Altia API library, try something like the following in application source code or an application common header file:

```
#include "altia.h"
#ifdef DEEPSCREEN
    #include "altia/altiaAnimationIdTable.h"
    #define altiaSendEvent altiaSendEventId
    #define altiaSendText altiaSendTextId
    #define AtSendEvent AtSendEventId
    #define AtSendText AtSendTextId

#else
    /* Not including altiaAnimationIdTable.h because this
     * is not DeepScreen.  Define the ALT_ANIM() macro.
     */
    #define ALT_ANIM(x) API_TEXT(#x)
#endif
```

After the above macros are defined or a header file is included that defines them, use calls like the following to send events or strings to the client or DeepScreen version of the Altia API.

This 1st example sends value 50 to a "speed" animation.

```
altiaSendEvent(ALT_ANIM(speed), (AltiaEventType) 50);
```

This 2nd example sends string "Hello World" to a "text" animation.

```
altiaSendText(ALT_ANIM(text), API_TEXT("Hello World"));
```

6.7 Objects Share Static Transforms and Bitmap Data

To reduce code size, DeepScreen shares object transforms that are the same between different objects. It is only possible to share transforms that are constant throughout execution. Objects that are animated have transforms that change unpredictably during execution and these cannot be shared between objects.

To reduce graphics library memory usage at execution time, DeepScreen shares graphics library bitmap data between multiple Altia Raster objects that are showing identical bitmaps.

Similarly, DeepScreen shares graphics library bitmap data for scaled/stretched/rotated Altia Raster objects if they are showing identical bitmaps and they are identically scaled/stretched/rotated. It is still considered good practice to avoid scaling/stretching/rotating Altia Raster objects because it still increases graphics library memory usage and hurts bitmap rendering performance. However, for situations where it is impossible to avoid, DeepScreen makes its best effort to reduce graphics library memory usage by sharing the bitmap data whenever possible.

DeepScreen also shares the raw bitmap data defined in the **`altia\data.c`** file for Altia Raster objects. This reduces code size when a design contains multiple Altia Raster objects that are showing identical bitmaps.

6.8 Opacity (Alpha Channels) and DeepScreen

The opacity feature of the Altia Editor allows objects or groups of objects to be made semi-transparent (from completely solid to completely invisible). For more on using opacity in the Altia Editor, please see the ***Tools and Commands*** chapter of the ***Altia Design User's Guide***.

Keep in mind that drawing semi-transparent objects may be a very CPU-intensive task and therefore should be reserved for sufficiently powerful target hardware.

6.9 Image Object Handling

There is optional support for Image objects to read image data from PNG files for some targets. Enable the **Image Object use files** code generation option for this feature. This typically requires a file system and file open and read facilities. If a target supports this feature, it will be described in the target's separate User's Guide.

If this option is **not** selected, the code generator only uses the existing image data showing in the Image object at code generation time and no code is generated to open, read or process files (which is very important for devices with no file system and/or no **`fopen()`** and **`fread()`** functions). The current image showing in the Image object is generated to code in the same way as a Raster object.

If the **Image Object use files** option is selected, it is left as an exercise for the user to copy the necessary PNG files to the device. Only PNG files are supported. The DeepScreen version of the Image object will not read BMP or JPEG files.

If Image objects refer to PNG file names using relative paths (for example, just **`image.png`** or **`images/image.png`**), the PNG files must be placed on the target in a similar relative location to the working directory from which the DeepScreen generated code will execute.

The handling of Image objects in DeepScreen generated code is enhanced for designs containing Image objects that have identical **`"image_name"`** animations and the **Image object use files** code generation option is chosen. At DeepScreen code execution time when Image objects are initializing, the image file name held by the **`"image_name"`** animation is only read one time (by the first Image object with the given **`"image_name"`** that gets initialized). Other Image objects with the same **`"image_name"`** animation do not read the file again when they initialize. Instead, they will detect that the internal raster structure is already referencing valid image data and there is no reason to read the file. This enhancement improves initialization time and uses less heap memory (just one copy of the raster data instead of a copy for each Image object that has an identical **`"image_name"`** animation). For DeepScreen targets released after July 30, 2013, this enhancement also improves performance when loading a new image file name after initialization. If a shared **`"image_name"`** animation is given a new image file name (such as from application code via the Altia API or from Altia Control code), only the first Image object with that same **`"image_name"`** animation opens the image file and loads the image. Any other Image objects with that same **`"image_name"`** animation will use the image data already loaded into the same internal raster structure so the final amount of memory in use will be for just one copy of the image data. Only the first Image object with the shared **`"image_name"`** animation consumes CPU and I/O time/resources to open the image file and load the image data.

Continue reading the next section *Controlling Image Data Loading at DeepScreen Initialization Time* to learn how the loading of Raster, Stencil, and Image object data can be controlled at execution time.

6.10 Controlling DeepScreen Image Data Loading with Preload Options

To improve initialization speed and/or reduce memory usage, limit the loading of Rasters (color images), Stencils (monochrome images), and Image objects (a special object that dynamically loads images from PNG files) with the **`NO_PRELOAD`**, **`PRELOAD_CNT`**, **`PRELOAD_STENCIL_CNT`**, and **`PRELOAD_IMAGE_CNT`** compile defines.

If **`NO_PRELOAD`** is defined (for example, with a **`-DNO_PRELOAD`** compile command line option), the DeepScreen initialization code will not load the device dependent bitmap (DDB) elements of Rasters, Stencils, and Image objects until a Raster, Stencil, or Image object actually needs to be drawn. The **`NO_PRELOAD`** capability has an extra enhancement so that Rasters referring to identical image data share a single DDB and Stencils referring to identical monochrome image data share a single DDB.

For Image objects with **`NO_PRELOAD`** defined, an image load is suppressed even when there is an **`"image_name"`** animation change and the Image object is currently hidden (such as in a hidden card of a Deck). In other words, the Image object only loads an image when it actually becomes visible if the image has not already been loaded for that Image object.

For the **`NO_PRELOAD`** feature to apply to Image objects, the **Image Object use files** code generation option must be chosen and the target must support loading Image objects from files. This typically requires that the target system have a file system. If **`NO_PRELOAD`** is defined, an Image object gets loaded from its image file when it first becomes visible or the file name that its **`"image_name"`** animation refers to is changed. For example, from application code using an Altia API function call like:

```
altiaSendText("image_name", "image.png");
```

`PRELOAD_CNT` is only important if **`NO_PRELOAD`** is defined. If **`NO_PRELOAD`** is defined and **`PRELOAD_CNT`** is not defined, **`PRELOAD_CNT`** defaults to **`0`**. The value of **`PRELOAD_CNT`** determines the number of Raster DDBs that can be simultaneously loaded. For the default value of **`0`**, all DDBs will be kept with the benefit of only loading them when they need to be drawn. For a value of **`1`** (for example, with a **`–DPRELOAD_CNT=1`** compile command line option), only 1 DDB is loaded at a time. This minimizes memory usage, but requires more CPU usage to load and unload DDBs as each unique Raster needs to be drawn.

As an extreme example, a **`PRELOAD_CNT`** of **`600`** (for example, with a **`–DPRELOAD_CNT=600`** compile command line option) allows 600 simultaneous Raster DDBs to be loaded. For a typical design, this might be all of the Raster DDBs in the design. The performance hit to load each DDB is not incurred until the Raster needs to be drawn. Over time, however, the software consumes more and more memory as new DDBs are loaded.

`PRELOAD_STENCIL_CNT` works like **`PRELOAD_CNT`** to control the loading of DDBs for Stencils instead of Rasters.

The **`PRELOAD_IMAGE_CNT`** feature has these characteristics:

- The feature is enabled by defining a value for the macro **`PRELOAD_IMAGEOBJ_CNT`** at compile time (for example, **`-DPRELOAD_IMAGEOBJ_CNT=50`**).
- This feature is only applicable when there are Image objects in the design and the option **Image Object use files** is enabled at DeepScreen code generation time. In this case, code is generated for Image objects to load their images from the file system at execution time of the DeepScreen generated code. If the **Image Object use files** option is not chosen, the image files currently being referenced by Image objects at the time code is generated are processed into pixel data and generated as constant Raster data. In this latter case, the DeepScreen code has no Image objects, only Raster objects.
- To use the **`PRELOAD_IMAGEOBJ_CNT`**, the existing **`NO_PRELOAD`** macro must also be defined which means that a **`PRELOAD_CNT`** for Raster images should also be defined (or let it default to 0) as well as a **`PRELOAD_STENCIL_CNT`** for Stencils (or let it default to 0).
- The **`PRELOAD_IMAGEOBJ_CNT`** macro is defined at compile time to establish a maximum number of Image objects that can be simultaneously loaded. At initialization time, no Image objects are loaded until they need to be rendered (this is the same as the current behavior when **`NO_PRELOAD`** is defined). If a new Image object must render and it is not already loaded and the maximum Image objects, as defined by **`PRELOAD_IMAGEOBJ_CNT`**, are already loaded, one of the existing Image objects will be unloaded prior to loading the new Image object. Ideally, it would be the Image object that has been loaded the longest. Unfortunately, this is hard to establish except for the first Image object that gets loaded. When an Image object is unloaded, the search for the next Image object to be unloaded begins with the Image object having the next higher index (as determined by the Image

object's index in the dobjs array of altia/data.c). Priority for unloading the next Image object is given to the next Image object that is hidden (for example, in a card of a Deck that is not showing) if one is hidden.

- **`PRELOAD_IMAGEOBJ_CNT`** for Image objects works independently from **`PRELOAD_CNT`** for Rasters and **`PRELOAD_STENCIL_CNT`** for Stencils. This is consistent with how **`PRELOAD_CNT`** for Rasters works independently from **`PRELOAD_STENCIL_CNT`** for Stencils.
- If **`NO_PRELOAD`** is defined and **`PRELOAD_IMAGEOBJ_CNT`** is **not** defined or it is defined as **`0`**, the behavior will be exactly like the behavior for Image objects when **`NO_PRELOAD`** is defined. That is to say, none of the Image objects will load until they need to be rendered and there will be no limit as to how many of them can load simultaneously.
- The image files to be loaded can be the default PNG file format, a target specific DDB format if a **`ddb_files .gen`** file is chosen at code generation time, or a target specific DDB RLE format if a **`ddb_rle_files .gen`** file is chosen at code generation time.
- The possibly frequent unloading and reloading of images can fragment the heap. In most cases, a steady state is eventually reached. If memory allocation fails even though the **`PRELOAD_IMAGEOBJ_CNT`** is known to be small enough for the specified number of Image objects to reside simultaneously in memory, the most likely reason for the failure is fragmentation of the heap. The expected solution is to lower the **`PRELOAD_IMAGEOBJ_CNT`** to keep more memory free to allow for the fragmentation to reach a steady state. This fragmentation is more likely when unloading/reloading PNG files because of the architecture of the PNG utility library. The fragmentation is expected to be less severe when using a **`ddb_files .gen`** or **`ddb_rle_files .gen`** since the image file is exactly in the target's DDB file format which allows for a single memory allocation operation.
- If memory allocation failures are an unresolvable problem, try defining the macro **`PRELOAD_IMAGEOBJ_DELETE_ALL=1`** at compile time to unload all images if a load fails (for example, **`-DPRELOAD_IMAGEOBJ_DELETE_ALL=1`**). The concept is to recover from possible memory fragmentation by unloading all images. **This macro should only be defined to 1 for extreme circumstances!** Also note that it will cause unloading of all images even for the (perhaps common) case where an image file is really missing so all image files should exist if this macro is defined to 1 at compile time.
- To unload more than just a single image when the **`PRELOAD_IMAGEOBJ_CNT`** is exceeded, define the macro **`PRELOAD_IMAGEOBJ_UNLOAD_ALL_HIDDEN=1`** at compile time (for example, **`-DPRELOAD_IMAGEOBJ_UNLOAD_ALL_HIDDEN=1`**). This enables the unloading of all hidden images, not just one hidden image, if the current number of loaded images is at the **`PRELOAD_IMAGEOBJ_CNT`** and another image needs to be loaded. The idea is to minimize the current load count to just objects that are visible. If no hidden images are available to unload, then just one visible image is unloaded instead.
- Debug messages can be enabled at compile time by defining the macro **`IMGOBJ_EXTRA_DEBUG`** to **`1`** (for example, **`-DIMGOBJ_EXTRA_DEBUG=1`**) and more debug messages are enabled by defining the macro **`IMGOBJ_EXTRA_DEBUG1`** to **`1`**. If the target has a stdout terminal, the **`IMGOBJ_EXTRA_DEBUG`** messages display to the stdout terminal when the maximum number of Image objects are loaded and an Image object must be unloaded to load another Image object that needs to be rendered. The additional **`IMGOBJ_EXTRA_DEBUG1`** messages display on each request to load an Image object that is not already loaded. The messages include image width and height values for each image that is unloaded and each image that is loaded.

Define these macros with compiler options like:

```
-DNO_PRELOAD -DPRELOAD_CNT=100 -DPRELOAD_STENCIL_CNT=100 -DPRELOAD_IMAGE_CNT=100
```

This is usually done in the target's generated Makefile script. That is, an **altmake.bat** that invokes a make utility for compiling the generated code. An **altmake.bat** script has something like a **TARGETDEFS** variable for passing command line options to the compiler. Make a copy of the generated **altmake.bat** script (for example, name the copy **altmake_preload.bat**). Edit **altmake_preload.bat** and define these macros in the **TARGETDEFS** variable. Save the changes and execute **altmake_preload.bat** instead of the generated **altmake.bat** to compile the generated code with these extra options.

Or, put the defines in a project IDE file (for example, the tool chain requires a project file instead of a make file) where other compiler options are also defined.

6.11 Manipulating Fonts in DeepScreen

Fonts can be changed for text objects (Labels, Text I/Os, Pie Charts, Tick Mark Objects, etc.) using the "altiaSetFont" built-in animation. This animation is available in both Altia Design and Altia DeepScreen. In addition, Altia DeepScreen has the additional APIs:

- **`int altiaGetFont(const AltiaCharType *name);`**
- **`int altiaGetFontId(int nameId);`**
- **`int altiaSetFont(const AltiaCharType *name, int fontId);`**
- **`int altiaSetFontId(int nameId, int fontId);`**

The new font APIs will get and set the font for a Text I/O and MultiLine Text object. These APIs are similar to the built-in animation, however they operate using the "name" animation instead of numerical Object IDs. The "name" animation is the same animation used in **altiaSendText** when sending a new text string to a text object.

When getting the font, the **altiaGetFont** API will find the font ID for the **FIRST** text object associated with the "name" animation. The return value is the enumerated font ID currently assigned to that text object (or -1 upon failure).

When setting the font, the **altiaSetFont** API will change the font for **ALL** text objects associated with the "name" animation. The return value will be zero upon success (or -1 upon failure).

Refer to **altia.h** in the generated DeepScreen code for the function declarations.

6.11.1 Adjusting Font Baseline

Both Altia Design and Altia DeepScreen use the bottom left corner of objects to determine position. When changing fonts, this can cause the text to shift vertically if the old font and new font have a different vertical descent (from the baseline).

To preserve baseline position, define **ALTIA_SET_FONT_BASELINE_ADJUST** at compile time. This will cause the altiaSetFont API to shift the vertical position of effected text so the baseline remains in the same position. NOTE: This will also perform the shift for the "altiaSetFont" built-in animation.

IMPORTANT: Baseline adjustment only changes the vertical position. This will result in erroneous behavior for transformed text (rotated, scaled, or distorted text).

IMPORTANT: Baseline adjustment only functions for Labels and Text I/O objects. The baseline shift is not applied to MultiLine Text, Pie, or TickMark Objects.

6.11.2 Enumerated Font IDs

For DeepScreen target releases after September 2014, an enumerated list of Font IDs will created during code generation in the **altiaAnimationIdTable.h** header file. A simple example from a design using three font face/size combinations:

```
#ifndef ALTIA_FONT_ID_TABLE
#define ALTIA_FONT_ID_TABLE

typedef enum
{
 _msdpi_Arial_medium_r_normal____130____________ ,
 _msdpi_Arial_medium_r_normal____180____________ ,
 _msdpi_Arial_medium_r_normal____200____________
} ALTIA_FONT_ID;
#endif /* !ALTIA_FONT_ID_TABLE */
```

The enumerated font IDs can be used in the new font APIs. The IDs are created from the Altia X11 Font ID Strings with underscores for digits that are not valid for C-syntax.

6.11.3 Obtaining Text Width in Pixels

In addition to the get/set font APIs, the following text width APIs are now available:

- **int altiaGetTextWidth(const AltiaCharType *name, AltiaCharType *text);**
- **int altiaGetTextWidthId(int nameId, AltiaCharType *text);**

These functions will calculate the pixel width of a text string using the font assigned to a text object. Like the get/set font APIs, the text width APIs use a "name" animation to identify what text object to use for width calculations. These functions do not change the text currently displayed for the text object. Instead they are used when a pixel width is required for algorithmic calculations in the application code (examples: determining when to elide a text string, or computing a cursor position).

Similar to **altiaGetFont**, the **altiaGetTextWidth** API will use the **FIRST** text object associated with the "name" animation for width calculations.

Refer to **altia.h** in the generated DeepScreen code for the function declarations.

7 Brief Description of Generated Code Files

For most targets, the generated code is written as a set of C source files (**.c** extensions), header files (**.h** extensions), a make file (**altmake.mk**) and a script (**altmake.bat**) to execute the make file.

The generated files are created in the folder containing the currently opened design (**.dsn**) file and a sub-folder named **altia**.

> **NOTE: Because the generated files have generic names (that is, the names are not unique to the particular design file), code should only be generated for one design file in any given folder. Otherwise, the generated code files for one design file will overwrite the generated code files for any other design file residing in the same folder.**

A text file named **altiaOps** is also written to the destination folder. The **altiaOps** file contains the options that were selected in the Code Generation Options dialog. This file is read and its contents are used to configure the default settings in the Code Generation Options dialog each time code generation is performed for a design file in the same folder. Any changes to the generation options are written back to the **altiaOps** file when code is generated.

If you open the destination folder (where the **.dsn** design file resides), you will see something like the following new files after code generation:

altmake.bat	A script that executes to make a target executable program from the generated code sources. This script is target specific. It sets up necessary environment variables for the make and then executes the target-specific Make program with altmake.mk as the Make file. To change the script's template for special target environments, see *Customizing the Compile for a Target*.
altmake.mk	A Make file that is target specific.
altiaOps	The settings for the Code Generation Options dialog from the last time code was generated for a design file in the destination folder.
altia\altia*.[ch]	Generic sources for managing and drawing objects.
altia\<target>*.[ch]	A layer that contains files with the actual calls to the target specific render engine functions. These files are isolated for easier support of multiple targets.
altia*Animate.c	Special code for managing and drawing special objects such as plots and text I/Os. These files may contain no code or limited code based on the types of special objects selected for code generation.
altia\animate.c	Contains the **AltiaAnimate()** function which is called to change the state for an animation.

altia\stimulus.c	If code was generated for stimulus and one or more objects actually have stimulus, this file contains the code to process input events into **AltiaAnimate()** calls. Else, this file is empty.
altia\draw.c	Contains the **AltiaUpdate()** function which is called when a redraw is desired after one or more **AltiaAnimate()** calls.
altia\data.c	Contains data structures that describe the specific attributes of the objects for which code was generated. These data structures are used to draw the objects with the correct sizes, locations, colors, animations, etc.
altia\altiaAPI.c **altia\altiaAPICbk.c**	DeepScreen version of Altia C/C++ API. Fully compatible with existing Altia API Chapter 2 functions (see ***Altia API Reference Manual*** for general API information and function descriptions).
altia\altiaSimpleMain.c	A default **main()** routine for the case where no custom application **main()** is present. This **main()** initializes the DeepScreen code and executes an event processing loop to handle mouse, keyboard, window refresh and window close events.
altia\<target>\WinMain.c	For code generated to run on Windows, this file contains a **WinMain()** for the case where the program is linked as a true Windows application and no custom application **WinMain()** is present. This **WinMain()** simply calls **main()** which may be the main() from **altia\altiaSimpleMain.c** or it is a custom application **main()** if one is present.
altia\common\altiaUtils.c	Contains global functions for initiating the DeepScreen Graphics code, controlling it after initialization, and closing the graphics code. These functions are listed below.
void TargetAltiaInitialize()	Does the work of creating a window (if the target has a windowing system) or initializing a screen area for the DeepScreen graphics and initializing the DeepScreen code. It determines whether initialization has already been done and does nothing in such a case. This allows it to be called more than once by unrelated sub-systems that may want to insure that everything is initialized. It is not necessary for a program to call this function if the program is linking with the DeepScreen version of the Altia API. It gets called automatically on API initialization via **AtOpenConnection()**, **altiaConnect()** or **AtStartInterface()**.
int TargetAltiaAnimate (AltiaNativeChar *funcName, double value)	Performs an Animate of the DeepScreen graphics associated with a particular animation

	function name. A non-zero value is returned if the animation requires an update and **`TargetAltiaUpdate()`** can be called to perform the update. A zero is returned if the animate request does not result in any graphics changes. NOTE THAT THIS FUNCTION TAKES AN 8-BIT STRING ON NON-UNICODE SYSTEMS OR A WIDE STRING ON UNICODE. THIS IS INDEPENDENT OF THE REST OF THE API FUNCTIONS. THE **`value`** MUST BE THE VALUE TYPE EXPECTED BY DEEPSCREEN CODE (double for floating point code generation, integer for fixed point) AND NOT JUST AN **`int`** EVEN IF AN INTEGER API IS BEING USED. It is not necessary for a program to call this function if the program is linking with the DeepScreen version of the Altia API. It gets called automatically by **`AtSendEvent()`**, **`altiaSendEvent()`**, etc. as needed.
`void TargetAltiaUpdate()`	Performs an update of the DeepScreen graphics. An update redraws any graphics that need changing since the last call to **`TargetAltiaUpdate()`**. Drawing is only performed during a **`TargetAltiaUpdate()`** call which gives application developers full control of the redraw schedule. It is not necessary for a program to call this function if the program is linking with the DeepScreen version of the Altia API. If caching is disabled (the default), it gets called automatically by **`AtSendEvent()`**, **`altiaSendEvent()`**, etc. If caching is enabled, it is called by **`AtFlushOutput()`**, **`AtPending()`**, etc.
`int TargetAltiaLoop(AltiaNativeChar *portName)`	This function provides a simple event-processing loop that can be used if there is no custom processing required. It takes a port name as an option if a full API is being used that needs to connect to an existing Altia session. Otherwise, just pass NULL. It returns a suggested program return value. NOTE THAT THIS FUNCTION TAKES AN 8-BIT STRING ON NON-UNICODE SYSTEMS OR A WIDE STRING ON UNICODE. THIS IS INDEPENDENT OF THE REST OF THE API FUNCTIONS.
`void TargetAltiaClose()`	This function "closes" the DeepScreen code and cleans up the window. It determines whether the window has already been closed and does nothing in such a case. This allows it to be called more than once by unrelated sub-

systems that may want to insure that everything is cleaned up.

It is not necessary for a program to call this function if the program is linking with the DeepScreen version of the Altia API. It gets called automatically by **`AtCloseConnection()`**, **`altiaDisconnect()`** or **`AtStopInterface()`**.

8 DeepScreen Altia API Server

8.1 DeepScreen Altia API Server Overview

This is a new capability added to the DeepScreen 5.0 or newer code generator as of the Altia Design 8.04 DeepScreen 5.0 product release.

The DeepScreen Altia API Server is software that compiles and links with DeepScreen generated code for Windows GDI32, Intel x86 Software Render Windows, and X11 targets. The DeepScreen Altia API Server software is provided as-is for no additional cost. To date, customers have successfully deployed it on Windows GDI32, Linux and Solaris running X11, and even with DeepScreen X11 generated code running on VxWorks!

When the resulting executable runs, it opens a TCP/IP socket for serving programs that are using the TCP/IP socket (i.e., **lan**) version of the Altia API. In this way, application programs can connect to, send, and receive events with the DeepScreen executable in the same way that they connect to, send, and receive events with an Altia Design editor or Altia Runtime.

An application program linked with the latest version of the Altia API **lan** library for Windows, Linux or Solaris can also start a DeepScreen executable using AtStartInterface() in a similar way that an existing application program starts Altia Runtime with AtStartInterface().

The DeepScreen Altia API Server software also allows starting the DeepScreen executable so that it does not initially show its window by passing **`-nowin`** as a command line argument at execution time (this works for Windows GDI32 and X11 targets, **not** the Intel x86 Software Render Windows target). This is the same argument already supported by Altia Runtime.

Finally, the DeepScreen Altia API Server software supports a subset of the Altia API view manipulation functions to close and open the DeepScreen window (for Windows GDI32 and X11 targets, **not** the Intel x86 Software Render Windows target). The DeepScreen Altia API Server software also supports a new built-in animation altiaSetViewStyle to allow setting the style for the DeepScreen window prior to opening it (for Windows GDI32 and X11 targets, **not** the Intel x86 Software Render Windows target).

8.2 Making DeepScreen Generated Code with the Altia API Server Software

Generate DeepScreen code as usual for Windows GDI32, Intel x86 Software Render Windows, or X11.

Generating DeepScreen code creates a folder named `altia` with generated source files (or updates the folder if it already exists). The `altia` folder is created in the same folder that contains the design (`.dsn`) file for which code was generated. For example, if code is generated for `c:\projects\ds\testing\testing.dsn`, then generating code either creates or updates the folder `c:\projects\ds\testing\altia` with source files.

One of the generated source files in the altia folder is `altiaAPIServer.c`.

To make a DeepScreen executable with DeepScreen Altia API Server functionality, manually copy `altia\altiaAPIServer.c` to the folder containing the design (`.dsn`) file, rename the copy so that it has the same name as the design (`.dsn`) file with a `.c` extension instead of a `.dsn` extension, and make the DeepScreen generated code as usually.

If DeepScreen Windows GDI32 or Intel x86 Software Render Windows code is generated for `c:\projects\ds\testing\testing.dsn`, the steps might look like the following if performed from a Command Prompt window on a Windows XP or Windows 7 machine:

```
c:
```

```
cd  \projects\ds\testing\altia
copy  altiaAPIServer.c  ..\testing.c
cd  ..
altmake.bat
```

Alternatively, perform the copying and renaming from a My Computer or Windows Explorer window and make the generated code into an executable using the **Make Standalone** option from the **Code Generation** menu in Altia Design.

For X11 generated code copied to `/projects/ds/testing` on a Linux system, the steps might look like the following if performed from a command prompt window on Linux using a GNU compiler:

```
cd  /projects/ds/testing/altia
cp  altiaAPIServer.c  ../testing.c
cd  ..
altmake.bat
```

The result should be an executable (in this example, `testing.exe` on Windows or just `testing` on Linux) that supports other applications connecting via the TCP/IP socket (i.e., **lan**) version of the Altia API.

8.3 Running a DeepScreen Generated Executable Made with the Altia API Server Software

On Windows, the executable opens TCP/IP socket 5100 by default (the same socket that Altia Runtime opens by default). By no coincidence, it is the same socket that the Altia API functions `AtOpenConnection()` and `altiaConnect()` attempt to open on Windows if no extra arguments are given.

On Linux, the executable opens domain socket `/usr/tmp/vSe.<hostname>` by default (just like Altia Runtime). This is the same socket that the Altia API functions `AtOpenConnection()` and `altiaConnect()` attempt to open on Linux if no extra arguments are given. This is a domain socket which only supports communications with programs running on the same computer. Use the **-port** argument (as described in the next paragraph) to open a network socket for communicating with programs on other computers.

Like Altia Runtime, a DeepScreen executable running the Altia API Server code takes a **-port** option to specify a different socket if desired. On Windows, the option can look like:

```
-port :NUMBER
```

where `NUMBER` is a socket number. On Linux, the option can look like:

```
-port :NUMBER
```

or:

```
-port FILE_NAME
```

where `NUMBER` is a socket number or `FILE_NAME` is a path to a file that does not already exist. Some Linux systems restrict user specified socket numbers to values greater than 10000. If a file name path is specified, a domain socket is created based on the given file name.

For more information regarding the `-port` option, please see the *Altia API Reference Manual, Section 1 - Commands and Programs, altiart.out (UNIX), altiart.exe (PC)*.

A DeepScreen executable compiled with the Altia API Server software also takes the **`-nowin`** argument (for Windows GDI32 and X11 targets, **not** the Intel x86 Software Render Windows target). Just like Altia Runtime, **`-nowin`** starts execution without opening a window. The window is usually opened later by an application with one of the Altia API functions

`AtOpenSimpleView()/altiaOpenSimpleView()`,
`AtOpenView()/altiaOpenView()`, or
`AtOpenViewPlaced()/altiaOpenViewPlaced()`.

8.4 Starting a DeepScreen Executable Made with the Altia API Server Software

Use `AtStartInterface()` to start a DeepScreen executable made with the Altia API Server Software.

The transition from Altia Runtime to a DeepScreen executable made with the Altia API Server software would not be complete without support for starting the executable with `AtStartInterface()`.

This support is available from newer versions of the Altia API TCP/IP socket (**lan**) libraries for Windows or Linux. These new versions install as part of this latest version of Altia Design.

If this latest version of Altia Design is properly installed, the new versions of the libraries are:

```
<INSTALLDIR>/lib/ms32/liblan.lib
<INSTALLDIR>/lib/ms32/liblanMT.lib
<INSTALLDIR>/lib/linux/liblan.a
<INSTALLDIR>/lib/linux/liblanMT.a
<INSTALLDIR>/lib/solaris/liblan.a
<INSTALLDIR>/lib/solaris/liblanMT.a
<INSTALLDIR>/libfloat/ms32/liblan.lib
<INSTALLDIR>/libfloat/ms32/liblanMT.lib
<INSTALLDIR>/libfloat/linux/liblan.a
<INSTALLDIR>/libfloat/linux/liblanMT.a
<INSTALLDIR>/libfloat/solaris/liblan.a
<INSTALLDIR>/libfloat/solaris/liblanMT.a
```

Where `<INSTALLDIR>` is the Altia Design software installation directory. For example, `c:\usr\altia8075` by default for Altia Design 8.075 DeepScreen 5.0.

For these new versions of the Altia API library, `AtStartInterface()` still has the same arguments:

```
AtConnectId AtStartInterface(char *designFile, char *rtmFile,
                             int editMode,
                             int argc, char *argv[]);
```

The valid values for `editMode` are now:

0	attempt to find and start Altia Runtime	(unchanged)
1	attempt to find and start Altia Design	(unchanged)
2	attempt to find and start Altia FacePlate	(unchanged)
3	assume `designFile` is a DeepScreen executable program made with the Altia API Server software, attempt to start it, and connect to it	

Here is an example for using `AtStartInterface()` in a Windows console application to start `testing.exe` assuming that `testing.exe` is in the current working directory and it is a DeepScreen executable made with the Altia API Server software:

```
#include <stdio.h>
#include <altia.h>
int main(int argc, char *argv[])
{
    AtConnectId connectId;

    connectId
        = AtStartInterface("testing.exe", NULL, 3, argc, argv);
    if (connectId < 0)
    {
        printf("AtStartInterface failed!  Exiting.\n");
        return 0;
    }
}
```

8.5 Opening and Closing the Window of a DeepScreen Executable

As previously mentioned, a DeepScreen Windows GDI32 or X11 executable (not Intel x86 Software Render Windows) compiled with the Altia API Server software recognizes the command line option **`-nowin`** to start without opening its window. For DeepScreen Windows GDI32 or X11 generated code, the Altia API Server software supports a subset of the Altia API view manipulation functions (*Altia API Reference Manual, Section 3*) for opening its windows as described below:

```
Int AtSendEvent(AtConnectId connectId, "altiaSetViewStyle",
                    AltiaEventType style)
```

Make this Altia API call from an application program to set the style for the DeepScreen window. The style does not change until the DeepScreen window is opened with one of the view open routines (describe below) after the style is set. If the DeepScreen window is already opened, its style does not change until it is closed with `AtCloseView()/altiaCloseView()` and opened again with one of the view open routines (described below).

Valid values for `style` are:

0	full window decoration as determined by the windowing system and the close option will stop the executable.
1	no window decoration (i.e., no borders or caption)
2	borders and a caption, no minimize or maximize options and the close option, if visible, is ignored.
4	borders and a caption, no minimize or maximize options, and the close option will stop the executable.

```
int AtOpenSimpleView(AtConnectId connectId, int newViewId,
                          char *viewName)
```

Opens the DeepScreen window (if it is not already opened).

The `newViewId` argument should always be 1 to open the DeepScreen window. A value less than 1 is illegal and `AtOpenSimpleView()` returns -1 without opening the DeepScreen window. A value greater than 1 opens an additional view instead of the main DeepScreen window.

If `viewName` is not `NULL`, the string it points to becomes the caption for the DeepScreen window.

```
int AtOpenView(AtConnectId connectId, int newViewId,
               char *viewName, int universeX, int universeY,
               int width, int height, double magnification)
```

Opens the DeepScreen window if it is not already opened.

The `newViewId` argument should always be 1 to open the DeepScreen window. A value less than 1 is illegal and `AtOpenView()` returns -1 without opening the DeepScreen window. A value greater than 1 opens an additional view instead of the main DeepScreen window.

If `viewName` is not `NULL`, the string it points to becomes the caption for the DeepScreen window.

`universeX` is ignored. The DeepScreen window does not support repositioning the graphics within the window unless it is performed using traditional Altia animation techniques.

`universeY` is ignored. The DeepScreen window does not support repositioning the graphics within the window unless it is performed using traditional Altia animation techniques.

`width` is ignored.
`height` is ignored.

`magnification` is ignored. The DeepScreen window does not support scaling the graphics within the window unless it is performed using traditional Altia animation techniques.

```
int AtOpenViewPlaced(AtConnectId connectId, int newViewId,
                     char *viewName, int referenceViewId,
                     int referenceX, int referenceY,
                     int universeX, int universeY,
                     int width, int height,
                     double magnification)
```

Opens the DeepScreen window if it is not already opened.

The `newViewId` argument should always be 1 to open the DeepScreen window. A value less than 1 is illegal and `AtOpenViewPlaced()` returns -1 without opening the DeepScreen window. A value greater than 1 opens an additional view instead of the main DeepScreen window.

If `viewName` is not `NULL`, the string it points to becomes the caption for the DeepScreen window.

`referenceViewId` should be the constant `RootWindowPlace`. The DeepScreen window is always placed relative to the root window (i.e., relative to the full display screen).

`referenceX` and `referenceY` specify the pixel position of the DeepScreen window's lower left corner relative to the root windows lower left corner. For instance, `referenceX` of 20 and `referenceY` of 20 positions the DeepScreen window's lower left corner 20 pixels to the right and 20 pixels above the lower left corner of the root window. To center the DeepScreen window in the root window, use the constants `CenterViewX` and `CenterViewY` for `referenceX` and `referenceY`.

Note: Windowing system borders around the window may affect the exact position of the window depending on the thickness of the borders.

`universeX` is ignored. The DeepScreen window does not support repositioning the graphics within the window unless it is performed using traditional Altia animation techniques.

`universeY` is ignored. The DeepScreen window does not support repositioning the graphics within the window unless it is performed using traditional Altia animation techniques.

`width` is ignored.

`height` is ignored.

`magnification` is ignored. The DeepScreen window does not support scaling the graphics within the window unless it is performed using traditional Altia animation techniques.

`int AtCloseView(AtConnectId connectId, int viewId)`

Closes the DeepScreen window. The `viewId` value should always be 1. The DeepScreen window always closes if it is not already closed. Note that this does not stop the DeepScreen executable from running. It continues to run and the Altia API Server software continues to respond to application programs using the Altia API. If the DeepScreen window is opened again with one of the view open calls, graphics appear as if the window was never closed (i.e., all objects are in the correct current states).

8.6 Example Application Code for Controlling a DeepScreen Executable

Here is an example of an application program. If it is linked with a new version of the Altia API library, it starts a DeepScreen executable named `testing.exe` initially showing no window on TCP/IP socket 12345. After starting the DeepScreen executable, it sleeps 5 seconds before opening the DeepScreen window in style #2 (borders and a caption, no minimize or maximize options and the close option, if visible, is ignored). The program processes events from the DeepScreen executable to close or open the DeepScreen window and finally to stop everything. Please note that the `-nowin` option, `AtOpenSimpleView()` call, and `AtCloseView()` call are only supported with Windows GDI32 or X11 DeepScreen generated code (**not** Intel X86 Software Render Windows DeepScreen generated code).

```
#include <stdio.h>
#include <altia.h>
int main(int argc, char *argv[])
{
    AtConnectId altia_id;
    char *local_argv[20];
    int local_argc = 0;
    char *event_name;
    AltiaEventType event_value;

    local_argv[local_argc++] = "-nowin";
    local_argv[local_argc++] = "-port";
    local_argv[local_argc++] = ":12345";
    altia_id = AtStartInterface("testing.exe", NULL, 3,
                                   local_argc, local_argv);
        if (altia_id < 0)
        {
            fprintf(stderr, "AtStartInterface failed!  Exiting.\n");
            return 0;
        }

        /* For this simple program, no event caching is necessary */
        AtCacheOutput(altia_id, 0);

        altiaSleep(5000);
        AtSendEvent(altia_id, "altiaSetViewStyle", (AltiaEventType) 2);
        AtOpenSimpleView(altia_id, 1, "Testing DeepScreen");

        AtSelectEvent(altia_id, "doOpen");
        AtSelectEvent(altia_id, "doClose");
        AtSelectEvent(altia_id, "doStop");

        while (AtNextEvent(altia_id, &event_name, &event_value) == 0)
        {
            if (strcmp(event_name, "doOpen") == 0)
                AtOpenSimpleView(altia_id, 1, "More Testing");
            else if (strcmp(event_name, "doClose") == 0)
                AtCloseView(altia_id, 1);
            else if (strcmp(event_name, "doStop") == 0)
                break;
        }

        AtStopInterface(altia_id);
}
```

9 DeepScreen Layers

9.1 Overview

Layers provide a way in Altia DeepScreen to take advantage of a graphical hardware display's use of multiple overlay planes or multiple screens for targets that support such features. Please see the target's separate User's Guide to learn if a specific target supports layers.

Overlay planes are where the screens are stacked one on top of another and objects that are drawn on an overlay plane will appear above objects on the primary display. Overlay planes usually have a transparent or alpha component such that the background color on the overlay plane can be transparent so objects underneath can be seen. The altiaGL directxfb layer demo sets the transparent color to the background color of the design so that whenever that color is drawn on the overlay plane, that color will be transparent to the primary surface. There is no requirement that the layers be stacked on top of one another. They could be multiple displays as well. Each layer in DeepScreen can be a different color depth and size.

9.2 Assigning a Layer Number

The layer number of an object is specified in Altia Design by assigning an **Object Layer** property to the object and setting the property to a number. To do this, select the object and double click it to bring up the Altia Property Dialog. Under the **Edit** menu in the Altia Property Dialog, select **Add Document Property** and then select **Object Layer**. This creates an **Object Layer** property for the object. Set this to the layer number for the object. Layer number 0 is the primary surface. The first overlay surface should be layer number 1, the next overlay surface should be layer number 2, and so on until the maximum supported by the target hardware. A layer number of –1 indicates that the layer is a don't care. It defaults to the primary surface unless a parent or a child object overrides it. All objects default to a layer number of –1.

Altia Design's objects are arranged in a hierarchical structure with parents and child objects. Just like with colors, layers follow this hierarchical structure. So if a parent object has a layer number of 1, all its children will be on layer 1 regardless of their layer numbers. The parent object overrides the children's layer numbers. If the parent has a layer number of –1, each child is free to set its own layer number and it will not be overridden by the parent's layer number. If a child object also has a –1 layer number, a layer number is not defined for that object and it will default to the primary layer when drawn in DeepScreen generated code.

9.3 Implementing Layers in an altiaGL Target

In releases of Altia Design that support generating code for the altiaGL target, there is a demonstration available for showing layer support on Windows. A demonstration on Windows is made possible with the **`directxfb_layers.gen`** additional target file for altiaGL code generation. If this file is selected in the **Additional Targets files list:** field of the Code Generation Options dialog, the user can try out layers. This demo only supports 2 layers. Layer 0 is the primary display and layer 1 is an overlay plane that is the same size as the primary display. The overlay plane is set to 16 bit color while the primary display is whatever the current display is set to. The overlay plane sets its transparent color to the background color of the design (this is RGB 153 153 153 by default). Whenever this color is drawn on the overlay plane, the overlay plane will be transparent to the primary display. Look at driver.c in the Altia software installation usercode/altiaGL/directxfb directory to see how a driver should be configured to support layers.

Below are the steps required to turn an altiaGL driver into one that supports layers:

1. Add the flag %LAYERFLAG% to the driver's .gen file. This tells the code generator to generate code that is defined between $STARTOPTION layers and $ENDOPTION layers directives. This includes the layer numbers of objects and other data and code required for layers.
2. Add the following to your egl_md.h file

   ```
   $STARTOPTION layers
   #define MAXLAYERS X
   #define ALTIA_LAYERS
   $ENDOPTION layers
   ```

   where X is the maximum layer number the target hardware will support. Or, change X to $MAX layers and the code generator will replace $MAX layers with the maximum layer number used in the design. See egl_md.h in the Altia software installation usercode/altiaGL/directxfb directory for more information.
3. Add the function driver_initLayer(int layer, ScreenPtr screen) to your driver.c file. This function will be called once for each layer other than the primary layer up to MAXLAYERS. The function should initialize any hardware required for that layer and the screen parameter should be initialized by calling miScreenInit() in the same manner as driver_open() initializes the screen of the primary layer.
4. Add the function driver_SetLayer(int layer) to your driver.c file. This function will be called before drawing the given layer. It will be called before driver_StartGraphics(). It should set up the hardware for drawing to the layer. DeepScreen will completely draw to each layer starting with layer 0 before calling driver_SetLayer() for the next layer. It is possible for a layer to get a driver_SetLayer() and driver_StartGraphics(), but no actually drawing of the layer will be done before the driver_StopGraphics() function is called.
5. Modify the function driver_ColorAlloc(). This function can now be called with a different ScreenPtr parameter for each of the different layers. Since it is possible for each of the different layers to have different pixel depths and/or different color formats, this function must be modified to handle these different depths and formats. If all layers have the same depth and format as the primary display, nothing needs to be done. If, however, this is not the case, this function must be modified to support generating device dependent colors for each of the different layers. This function can use the current layer as defined by driver_SetLayer() to determine which layer is current.

10 Tasking Feature

10.1 Overview

The tasking feature provides a way in Altia DeepScreen to operate in non-preemptive operating systems where it's important to limit the amount of time spent in Altia API calls (i.e. altiaFlushOutput). All targets can support tasking if released since tasking was introduced in July 2013.

The tasking feature enables new API functions and creates two queues for managing the tasks in Altia DeepScreen. To use the taking feature, it's important to understand the new APIs and how the queues function.

10.2 Enabling Tasking

To enable the tasking feature, build the Altia DeepScreen generated code with the ALTIA_TASKING compile time definition. A typical way to do this is to set the TARGETDEFS variable before performing the build:

```
set TARGETDEFS=-DALTIA_TASKING
altmake.bat clean
altmake.bat standalone
```

Most targets will support the use of TARGETDEFS as a way to add compile time options to the compiler settings. Optionally you can create a port of your DeepScreen Target by customizing the code generation configuration (.gen file) and make script (.bat file).

NOTE: It's important to perform a clean after setting a new compile option like –DALTIA_TASKING so all the Altia DeepScreen source code files will be rebuilt.

10.3 How to Use Tasking

When tasking is enabled, the Altia API functions are not executed synchronously. Instead the APIs will push commands into a Task Queue. The commands will be pulled from the Task Queue by the new Altia Task function (altiaTaskRun). As the commands are processed, the execution time is monitored. If the specified time limit for altiaTaskRun() is exceeded, the function will return.

To use the tasking feature, modify your software project as follows:

- Ensure all code is built with ALTIA_TASKING defined at compile time
- Invoke altiaTaskRun() periodically from your application code
- If desired specify a non-zero time limit in the call to altiaTaskRun()
- Use the altiaTaskStatus() API to monitor queue usage and check for overflow conditions

NOTE: If commands are pushed into the Task Queue faster than they can be executed in altiaTaskRun() the queue will overflow and data will be lost. It is important to monitor the Task Queue using the altiaTaskStatus() API in order to "tune" your Application Code and its execution of the Altia APIs.

10.3.1 Example with main() function

The following example shows a simple main function which uses the tasking feature. This example demonstrates a self-contained Altia .dsn project. Such a project could be a touch-screen HMI with the application logic contained in the .dsn in the form of Altia Control Code.

```
#define TASK_LIMIT_MS       10

int main(int argc, char *argv[])
{
    int event_count;
    AltiaCharType * event_name;
    AltiaEventType event_value;
    AtConnectId altia_id;

    /* Connect to GUI.  For DeepScreen, this initializes and
     * displays the graphics.
     */
    altia_id = AtOpenConnection(NULL, NULL, argc, argv);

    /* If opening connection fails, return.  Will never fail for
     * DeepScreen, but could fail for Altia Design or Runtime.
     */
    if (altia_id < 0)
        return 0;

    /* Turn on caching so screen draws only occur during calls
     * to draw APIs such as AtPending() or AtFlushOutput().
     */
    AtCacheOutput(altia_id, 1);

    /* Now would be the time to use the Altia API function
     * AtSelectEvent(altia_id, string_for_event_name) to register
     * for events if appropriate.
     */

    /* Get events from GUI while GUI is active */
    event_count = 0;
    while (event_count >= 0)
    {
        event_count = AtPending(altia_id);
        while (event_count > 0)
        {
            event_count--;
            AtNextEvent(altia_id, (AltiaCharType **)(&event_name), &event_value);

            /* Do something here with new events from the GUI */
        }

        /* Run the Altia task */
        altiaTaskRun(TASK_LIMIT_MS);

        /* Give up the CPU for other tasks -- no effect if the operating
        ** system does not support tasking with pre-emption.
        */
        altiaSleep(1);
    }

    /* Close connection to GUI.  For DeepScreen this frees resources */
    AtCloseConnection(altia_id);

    /* Return if we lose contact with GUI. */
    return 0;
}
```

10.3.1 Example for embedded application with application logic

The following example shows a simple HMI task function which executes application logic followed by the Altia task. It is assumed the application logic is making API calls such as altiaSendEvent() and altiaSendText(). The runMyHMI() call is a place holder for pumping the HMI application logic (typically a state machine). In this example, the runMyHMI() function will return TRUE if the screen needs to be drawn.

An example of how to initialize Altia is also provided in the hmiInit() function.

```
#define TASK_LIMIT_MS       5

void hmiTask(void)
{
    /* Pump application state machine logic */
    if (runMyHMI())
    {
        /* The application logic wants to update the display */
        altiaFlushOutput();
    }

    /* Run the Altia task (always) */
    altiaTaskRun(TASK_LIMIT_MS);

    /* Done with this cycle */
    return;
}

int hmiInit(void)
{
    /* Connect to GUI.  For DeepScreen, this initializes and
     * displays the graphics.
     */
    if (altiaConnect(NULL) < 0)
        return 0;

    /* Turn on caching so screen draws only occur during calls
     * to draw APIs such as altiaPending() or AltiaFlushOutput().
     */
    altiaCacheOutput(1);

    /* Now would be the time to use the Altia API function
     * altiaSelectEvent(string_for_event_name) to register
     * for events if appropriate.
     */

    /* Success */
    return 1;
}
```

10.4 New Tasking APIs

The Tasking feature introduces two new APIs:

10.4.1 int altiaTaskRun(unsigned long milliSeconds);

This function processes the task queue. The 'milliSeconds' parameter specifics a time limit (in milliseconds). If the time limit is exceeded the task function will return, leaving the current task in the queue unfinished. When invoked again, the task will resume where it left off previously.

A time limit of zero means "run until the queue is empty".

This function returns -1 upon error, 0 upon success. Errors include not having ALTIA_TASKING defined at compile time.

NOTE: The time limit for altiaTaskRun() is not a deterministic limit. It is a threshold used to stop processing of the task queue. When the limit is exceeded, the altiaTaskRun() function will stop processing commands and return.

NOTE: The task will skip an altiaFlushOutput command in the Task Queue if there is another flush farther down in the queue. This optimization can be disabled if the definition ALTIA_TASK_NO_FLUSH_OPTIMIZATION is defined at compile time.

10.4.2 `int altiaTaskStatus(AltiaTaskStatusType * status);`

The function obtains the status of the Altia task. The 'status' parameter must point to a valid and existing AltiaTaskStatusType structure:

```
typedef struct
{
    int task_size;
    int task_current_used;
    int task_peak_used;
    int task_overflow;
    int object_size;
    int object_current_used;
    int object_peak_used;
    int object_overflow;
    int object_max_time;
    int object_max_id;
} AltiaTaskStatusType;
```

The '_size' is the size of the associated queue in elements. The sizes are automatically set at code generation time based upon the contents of the .dsn used.

The '_current_used' is the number of queue elements currently used at the time of the altiaTaskStatus() function call.

The '_peak_used' is the maximum number of queue elements used since the last call to altiaTaskStatus(). Calling altiaTaskStatus() will reset the peak value to zero so a new peak can be measured.

The '_overflow' will be '1' if a queue overflow has occurred since the last call to altiaTaskStatus(). A value of '0' indicates that no overflow has occurred. Calling altiaTaskStatus() will reset the overflow flag so new overflows can be detected.

The 'object_max_time' and 'object_max_id' are used for debugging purposes. When the tasking feature interrupts a draw operation, these parameters indicate how much time was spent before the interruption and which object ID (in the .dsn file) was the last to be processed. This is useful because the last object ID may indicate an object in the .dsn which is especially time consuming to draw. This provides feedback on areas in the .dsn that may require some modification in order to achieve the desired performance requirements for your project. Calling altiaTaskStatus() will reset these parameters so new metrics can be obtained.

This function returns -1 upon error, 0 upon success. Errors including a NULL status pointer or not having ALTIA_TASKING defined at compile time.

10.5 Configuring the Queues

The tasking feature uses two queues:

- Task Queue – holds a list of pending tasks (i.e. sendEvent, sendText, flushOutput)
    - Default Size: 100 + (number of animations in the .dsn) + (64 * number of text objects in the .dsn)
    - Size can be overridden by setting ALTIA_TASK_QUEUE_SIZE at compile time.
    - Element Size: 8 bytes
- Object Queue – holds a list of objects to draw (i.e. lines, images, text, etc)

- Default Size: (total number of all objects in the .dsn)
- Size can be overridden by setting ALTIA_OBJECT_QUEUE_SIZE at compile time.
- Element Size: 38 bytes

In general it is not necessary to manually specify the size of the queues. The default size should be adequate for most situations. Monitoring the queue usage with the altiaTaskStatus() function will be helpful in customizing the queues for your specific .dsn project.

10.6 Troubleshooting

This section describes common problems that may occur when using the tasking feature. When troubleshooting Altia DeepScreen, it is always useful to monitor the error message functions in altiaLibError.c using a debugger breakpoint or a RS232 terminal connection to the embedded hardware.

10.6.1 Nothing appears on the embedded display

If the screen remains blank, make sure the altiaTaskRun() function is being called periodically. If this function is not called then the task queue will never be processed. Screen updates occur when flush commands are pulled from the queue for processing.

10.6.1 Display does not update as expected

If the display does not update as expected, it's possible that the time limit for altiaTaskRun() is set too low and flush commands are filling the queue faster than they can be processed.

By default the tasking feature will skip a flush command in the queue when there are other flush commands later in the queue. This is an optimization to improve performance because the screen will appear the same regardless if all flush commands are process versus only the final flush in the queue is processed.

However a small time limit combined with periodic altiaFlushOutput() calls will keep the queue filled with more flush commands than can be processed within the time limit. The result is a display that does not update. Increase the time limit for the altiaTaskRun() API or reduce the number of altiaFlushOutput() invocations by the application code.

10.6.1 Missing graphics on the display

If the queues overflow, commands will be dropped. This includes sendEvent and sendText commands in the Task Queue, as well as object draw commands in the Object Queue.

Use the altiaTaskStatus() API to verify that the queues have not overflowed.

Overflow can be caused by the following problems:

- Queue sizes are too small to handle peak loads for you HMI
  - This is a simple problem to fix by increasing queue sizes
- Application code is filling the queues faster than the queues can be processed
  - Queue size increase will ***not*** fix this problem.
  - Increase the time limit for altiaTaskRun()
  - Call altiaTaskRun() more frequently

- Modify the application logic to ensure altiaSendEvent() and altiaSendText() API calls are not being spuriously called (i.e. called when data has not changed).

10.6.2 Animations appear choppy on the display

The tasking feature is optimized to skip altiaFlushOutput commands in the queue if more than one flush command is present. This is a performance optimization because the last flush in the queue is all that is required to maintain proper screen appearance. Processing a flush command is time intensive because it often involves hardware synchronization (i.e. flipping frame buffers during the vertical blanking period).

Skipping flush commands may be undesirable if the application code pushes in multiple animation/flush sequences in order to achieve an animation effect (i.e. sliding graphics, fade-out operations, etc). To force all flush command in the queue to be processed, set the following compile time definition when building the Altia DeepScreen source code:

```
ALTIA_TASK_NO_FLUSH_OPTIMIZATION
```

NOTE: It is very easy to overflow the task queue when this definition is set. To prevent overflow you must ensure that altiaFlushOutput() is not called more often that altiaTaskRun() is called. In addition, altiaTaskRun() should have a sufficient limit to allow processing of the pending flush commands.

10.6.3 Graphical objects appear in wrong place (Windows)

When running on a Windows target using multiple threads, it's important that the altiaTaskRun() API be called from the same thread that initializes the Altia connection. Using a different thread can cause redraw issues because the window for the display was created in the thread that opens the first Altia connection.

10.6.4 The altiaGetText and AtGetText API functions Fail

The altiaGetText() and AtGetText() API functions are disabled when using tasking due to the queuing nature of the tasking feature. These API functions will return -1 if used with the tasking feature.

To obtain the text for an animation, an allowable work around is to register a callback on the animation. The callback will receive the new text one character at a time. The callback will be invoked any time next text is sent to the registered animation.

11 Integrating DeepScreen Code with Simulink Generated Code

11.1 Overview

Integrating DeepScreen generated code with Simulink generated code may be possible for some DeepScreen targets.

The first step is to generate code from Simulink's Real-Time Workshop (RTW) and build it to interface to Altia Runtime on the Windows desktop. To learn how to do this, open the **Tutorial - Simulink Connection** icon in the **Altia Design** program group or open the document from the Altia Design **Help** menu.

After you have an RTW application working with Altia Runtime, the next step is to transition to linking the RTW generated code with DeepScreen generated code to replace Altia Runtime. To learn how this might be accomplished, open the **Tutorial - Simulink DeepScreen Connection** icon in the **Altia Design** program group or open the document from the Altia Design **Help** menu.

12 Integrating DeepScreen Code with Statemate generated Code

12.1 Overview

Integrating DeepScreen generated code with Statemate generated code may be possible for some DeepScreen targets.

The first step is developing a Statemate generated code application that interfaces with Altia Runtime. To learn how this might be accomplished, please open the **Tutorial - Statemate Code Connection** icon in your **Altia Design** program group or open the document from the Altia Design **Help** menu.

> **NOTE: You MUST use Altia's** `user_activities.c` **from the 05.10.2001 or newer version of the Altia Statemate connection to integrate DeepScreen code with Statemate generated code. This connection is supplied on the Altia Design DeepScreen CD-ROM. See your Altia software installation instructions for details.**

If you've followed the steps in Altia's Statemate code connection documentation, you should have a folder, typically under your Statemate work area **`prt`** folder, that contains:

- Statemate generated code and makefiles
- Altia **`bindings.h`** file that you created specifically for your project
- Altia version of **`user_activities.c`** file from the Altia Statemate connection software.
- An executable that uses Altia Runtime (**`altiart.exe`**, **`fonts.ali`** and **`colors.ali`**) and your Altia **`.dsn`** and **`.rtm`** files to display Altia graphics and communicate with it.

If you have these items, you are ready to attempt an executable that contains DeepScreen graphics code and does not require Altia Runtime or the Altia **`.dsn`** design file at execution time.

To learn how this might be accomplished, please open the **Tutorial - Statemate DeepScreen Connection** item in your **Altia Design** program group or open the document from the Altia Design **Help** menu.

13 Optimizing for DeepScreen

During code generation, DeepScreen will attempt to optimize your design in multiple ways:

- Removing superfluous groups
- Removing unused data for dynamic objects
- Packing high volume data into structures using bit fields
- Optimizing Text I/O objects

Each of these optimizations is impacted by the options in the Code Generation Options dialog. The following sections detail these optimizations further.

13.1 Removing Superfluous Groups

Groups are very convenient in the Altia Editor when organizing your design. In DeepScreen, groups that are only organizational (i.e. do not effect child objects) are unnecessary. The DeepScreen Code Generator will remove superfluous groups under the following conditions:

1. The **Built-in Animations** option is not enabled in the Code Generation Options dialog

Built-in animations often reference an object by ID using the **`altiaSetObj`** built-in. Superfluous groups cannot be removed when using built-ins because the application code may reference the object ID for the group with the **`altiaSetObj`** built-in.

A group is considered superfluous under the following conditions (all must be true):

- The group does not have any assigned attributes (opacity, color, font, fill style, outline style)
- The group is visible (hidden groups are not superfluous)
- The group is not transformed (stretched, scaled, rotated, mirrored, or distorted)
- The group does not have any Connections
- The group does not have any Animations
- The group does not have any Stimulus definitions
- The group does not have any Control Code
- The group does not contain any Text IO Objects which meet the following criteria:
    - Text IO uses the clip feature ("clip_on" animation set to non-zero)
    - Text IO uses sibling justification with maximum character count set to zero and there is no sibling beneath the Text IO.
    - Text IO uses "right" or "center" justification with a maximum character count set to zero.
- The group does not contain any MultiLine Text Objects which meet the following criteria:
    - MultiLine Text uses sibling justification and there is not sibling beneath the MultiLine Text.

IMPORTANT: If a one pixel shift is observed for an object in DeepScreen compared to Altia Design or compared to code generation from a previous release of a DeepScreen product, try disabling the Remove Superfluous Groups code generation option. This one pixel shift occurs in some unique situations, with or without superfluous group removal. The following steps can be performed to help isolate any issues with Superfluous Group Removal:

- Generate code, build, run, and observe if the issue is gone.
- If the change in appearance is isolated to just a few groups, it may be possible to disable optimizations on those groups rather than all groups in the design. To disable superfluous group removal on a specific group, either of the following can be performed:
    - Use the "DeepScreen Optimize" property described in Section 13.1.2.
    - Break one of the conditions which makes the group superfluous. A simple trick is to add a one line comment in control code for the group.

13.1.1 Code Generation Status Message

The Code Generation Status dialog will show the following message when a group has been removed:

Removing Superfluous Group #<ID>

Where <ID> is the Object ID of the group removed.

At the end of the code generation process, the Code Generation Status dialog will show a summary message detailing the total number of superfluous groups removed:

Removed <X> Superfluous Groups out of <Y> total groups

Where <X> and <Y> are the number of groups removed and the total number of groups, respectively.

13.1.2 Disabling This Optimization

This optimization can be disabled entirely by adding the following line to your target-specific .gen file:

```
%KEEPSUPERGROUPSFLAG%
```

If using version 11.2.1 or newer of the Altia Design Editor, then this option can be disabled entirely by unchecking the **Remove Superfluous Groups** option in the Code Generation Options dialog.

To disable this optimization on an object-by-object basis, use the **DeepScreen Optimize** property in the Property Editor. Add this property to each group that should not be optimized and set the property's value to "Do Not Optimize".

13.2 Removing Unused Data for Dynamic Objects

A Dynamic Object is an object that contains animations or is a container (like a Group, Deck, Clip, etc.). These objects contain a lot of variable state data (RAM) that may not be used if the group does not contain any custom animations (animations created by the User in the Animation Editor when the **Define** button is pressed). The DeepScreen Code Generator will remove the unused state data from RAM under the following conditions:

1. The **Built-in Animations** option is not enabled in the Code Generation Options dialog

Built-in animations can change the graphical state of an object such as its opacity (**`altiaSetOpacity`**), color (**`altiaColorFgObj`**), or position (**`altiaMoveObjX, altiaMoveObjY`**). Unused state data cannot be removed when using built-ins because the application code may modify the states using the built-in animations so the state data must remain in RAM.

13.2.1 Code Generation Status Message

The Code Generation Status dialog will show the following message when an object has been optimized:

Optimizing Dynamic Object RAM size.

At the end of the code generation process, the Code Generation Status dialog will show a summary message detailing the total number of objects optimized:

Optimized <X> Dynamic Objects out of <Y> total objects

Where <X> and <Y> are the number of optimized objects and the total number of objects, respectively.

13.2.2 Disabling This Optimization

This optimization can be disabled generating code with **Built-in Animations** enabled.

13.3 Packing Data Using Bit Fields

High volume data (data which occurs frequently in generated code) will be packed into bit fields by the DeepScreen Code Generator. This occurs regardless of code generation options enabled in the Code Generation Options dialog. However some options reduce the effectiveness of the bit field optimization:

- **Built-in Animations**
  - When enabled, this option will force some data from CONST structures to variable (RAM) structures. Variable data cannot be packed into bit fields because the range of the data is unknown.
- **Image Objects use File System**
  - When enabled, this option will force image metric data from CONST structures to variable (RAM) structures. Variable data cannot be packed into bit fields because the range of the data is unknown.
- Using Snapshot Objects or 3D Scene Objects in your .dsn
  - These objects force image metric data from CONST structures to variable (RAM) structures. Variable data cannot be packed into bit fields because the range of the data is unknown.

For maximum optimization, do not use the code generation options and objects listed above.

13.3.1 Disabling This Optimization

This optimization can be disabled by adding the following line to your target-specific .gen file:

```
%NOBITFIELDSFLAG%
```

If using version 11.2.1 or newer of the Altia Design Editor, then this option can be disabled entirely by unchecking the **Pack Data Into Bit Fields** option in the Code Generation Options dialog.

13.4 Optimizing Text I/O Objects

Text I/O objects contain many features that may not be used in each occurrence of the object. The DeepScreen Code Generator will remove unused features on an object-by-object basis for each Text I/O used in your .dsn file. The following optimizations are performed:

- Removal of *Input Data* and associated animations. This data and the associated animations are typically only used to create an interactive text entry field such as you can find in the standard **Inputs** library of Altia Design.
  - Occurs when the "**clip_on**", "**cursor_mode**", "**scroll_on**", "**jump_on**", "**scroll**", "**select_now**", "**select_on**", "**shortcut_on**" animations are all zero. The "**hilight_color**" animation must also be empty ("") or black.
  - Eliminates the following animations from the generated code for this object:
    - "**clip_on**"
    - "**cursor_mode**"
    - "**hilight_color**"
    - "**jump_on**"
    - "**output_cursor**"
    - "**scroll**"
    - "**scroll_on**"
    - "**select_now**"
    - "**select_on**"
    - "**shortcut_on**"
  - Removes the variable data for the animations above
- Removal of *Length* animations
  - Occurs when the "**length_mode**" animation is zero
  - Eliminates the following animations from the generated code for this object:
    - "**length**"
    - "**length_mode**"
- Removal of *Base Mode* animations
  - Occurs when the "**base_mode**" and "**decimal_pts**" animations are both zero
  - Eliminates the following animations from the generated code for this object:
    - "**base_mode**"
    - "**decimal_pts**"
- Removal of *Maximum Character Count* animation
  - Occurs when the "**max_char_count**" animation is zero
  - Eliminates the following animations from the generated code for this object:
    - "**max_char_count**"
- Removal of *Maximum Pixel Count* animation
  - Occurs when the "**max_pixel_count**" animation is zero
  - Eliminates the following animations from the generated code for this object:
    - "**max_pixel_count**"

IMPORTANT: When an animation is removed from a specific Text I/O object, the animation will not be functional in the DeepScreen generated code. To preserve one of the removed animations above, make sure that its removal criteria is not met (or force the optimization off on an object-by-object basis per Section 13.4.3 below)

13.4.1 Code Generation Status Message

The Code Generation Status dialog will show one or more of the following messages when a Text I/O object has been optimized:

Optimizing Text I/O input data.

Optimizing Text I/O length mode.

Optimizing Text I/O base mode.

Optimizing Text I/O max character count.

Optimizing Text I/O max pixel width.

At the end of the code generation process, the Code Generation Status dialog will show a summary message detailing the total number of objects optimized:

Optimized <X> Text-IO Objects out of <Y> total objects

Where <X> and <Y> are the number of optimized objects and the total number of objects, respectively. For this message, the value <X> counts only the Text I/O objects that received the "input data" optimization.

13.4.2 Disabling This Optimization

This optimization can be disabled by adding the following line to your target-specific .gen file:

```
%NOTINYTEXTIOFLAG%
```

If using version 11.2.1 or newer of the Altia Design Editor, then this option can be disabled entirely by unchecking the **Remove Unused Text I/O Animations** option in the Code Generation Options dialog.

To disable this optimization on an object-by-object basis, use the **DeepScreen Optimize** property in the Property Editor. Add this property to each group that should not be optimized and set the property's value to "Do Not Optimize".

13.4.3 Forcing This Optimization

To force this optimization on an object-by-object basis, add the **DeepScreen Optimize** property to a Text I/O Object and set the property's value to "True". This is useful when a Text I/O does not meet the criteria to quality for optimization, but optimization is still desired.

When forced in this fashion, all the possible Text I/O optimizations will be applied.

Forcing the optimization is only possible when the overall optimization is enabled. This means that the **Remove Unused Text I/O Animations** option in the Code Generation Options dialog is enabled. It also means that `%NOTINYTEXTIOFLAG%` must not be present in the .gen file.

IMPORTANT: This should only be done when the removed animations will not be used at runtime on the target hardware.

14 DeepScreen Enhancements and Fixes for Altia Design 10.2

This chapter describes general enhancements/fixes of the DeepScreen code generator for the Altia Design 10.2 release. Please see each target's separate Enhancements Summary and User's Guide for enhancements/fixes/features specific to a target. The later chapter *DeepScreen History for Altia Design 10.1 and Earlier Releases* provides details on enhancements/fixes for earlier releases of DeepScreen.

The general enhancements/fixes described below are ordered by incident number which is approximately the chronological order that enhancements were requested and issues to be fixed were reported.

- Incident #0635 & #0977: DeepScreen – The Text I/O object is not correctly calculating its width for the purpose of truncating its content when the "max_pixel_count" animation is greater than 0. It is calculating the width of individual characters and adding these widths together. It must calculate the width of the whole string to take into account the difference in the advance from one character to the next for internal characters of the string. This was fixed for the 10.2 release.

- Incident #0646: Altia Design/Runtime and DeepScreen – Enhancement: In previous releases of Altia Design, an empty Group had no visible handles if it was selected. Its position was 0,0 in the drawing universe (but this was difficult to determine since it did not show any visible handles when selected). Because it was empty, Altia Design and Runtime did not take its position into account if its parent object was a Clip. This behavior was different from the behavior in DeepScreen where its position was taken into account if it was in a Clip. In Altia Design/Runtime 10.2, an empty Group now has a visible size of 20 x 20 pixels, it will have visible handles if it is selected, and its position is taken into account by the editor if it is in a Clip. This results in consistent behavior between Altia Design/Runtime and DeepScreen, but will result in the Clip object changing its position in Altia Design/Runtime. Please note that an empty Group may not have a position of 0,0 in the drawing universe. Its position will be offset from 0,0 by any amounts that the Group was moved after it was originally created.

  Note: Empty Groups are not desirable. Please avoid creating a Group, moving or deleting all of the objects from the Group, and leaving the Group empty. Instead, move or delete the Group.

- Incident #0704 & #0705: Altia Design/Runtime and DeepScreen – Enhancement: The Multi-line Text object (MLTO) has a new "vert_style" animation to adjust the style of the vertical padding and the "vert_pad" animation now accepts negative values. Please see the ***Altia Design 10.2 Enhancements Summary*** for a detailed description of these new capabilities.

- Incident #0743: DeepScreen – If the "Language Object use files" and/or "Skin Object use files" options are NOT selected at code generation time, the DeepScreen generated code for the Language object and/or Skin object should NOT have calls to fopen(), fread(), etc. to read data from files because the data was generated in the source code. This was fixed for the 10.2 release.

- Incident #0800: DeepScreen – Enhancement: Optimize the Multi-line Text object (MLTO) drawing code so that it does not attempt to render any lines that are entirely out of the visible screen area. If the MLTO is transformed (rotated/scaled/stretched/distorted), this optimization is disabled.

- Incident #0804: DeepScreen – Enhancement: Vector Path objects that have identical path commands, path parameters, vector paints, dash strokes, linear gradients, radial gradients, and/or gradient stops now share the data for these elements in the DeepScreen generated code. This reduces the memory footprint of the constant data portion of the compiled code. The Vector Path object is not supported on every DeepScreen target. Please see the separate User's Guide for a specific target to determine if it supports Vector Path objects.

- Incident #0840: DeepScreen – Enhancement: The Altia API Server code's main() routine was using a 50 msec pause to give other tasks/processes time to execute. This pause was too long for smooth keyboard/mouse input response and smooth timer stimulus response. The pause is changed to 5 msec. This improves keyboard/mouse response time, provides smoother timer stimulus response, and even improves response to client applications sending many animation events. It is still a long enough pause to give other tasks/processes time to execute.

- Incident #0871 & #0997: DeepScreen – Enhancement: Provide a callback facility to user application code if an event is detected to an undefined animation name. Here is the prototype for the callback function and extern declaration for the variable in DeepScreen that should be set to register the callback function:

```
typedef void (*AltiaReportFuncType) (AltiaCharType  *name, AltiaEventType value);
#if AltiaUseAnimationIds
typedef void (*AltiaReportFuncIdType)( int nameId, AltiaEventType value);
#endif
extern AltiaReportFuncType AltiaInvalidAnimationNamePtr;
```

User application code would define a function of the appropriate type and assign the address of the function to the extern AltiaInvalidAnimationNamePtr as in:

```
static void missingAnimationFunctionCB(AltiaCharType * name, AltiaEventType value)
{
    printf("Missing Animation Function [%s], value = %d\n", name, value);
}
. . .
/*
 * register callback with the Altia engine
 */
AltiaInvalidAnimationNamePtr = missingAnimationFunctionCB;
```

- Incident #0917: DeepScreen – Enhancement: When generating code for a design containing Image objects, the Image objects must be referencing 24-bit BMP images (**not** 1-bit monochrome, 4-bit 16 color, or 8-bit 256 color BMPs) or they can be referencing JPEG images or PNG images for the Image objects to be properly processed. If Image objects are referencing 1-bit monochrome, 4-bit 16 color, or 8-bit 256 color BMPs, this causes a code generation error in the Altia Code Generation Output Dialog similar to:

```
Processing Image Obj id #7
Failed to process object/group #7
```

If the Altia Image Generator window is not hidden, it displays an error such as ERROR: This image doesn't exist (no file or no data). The code generation will fail. To suppress the failures (but still **not** process the Image objects showing 1-bit monochrome, 4-bit 16 color, or 8-bit 256 color BMPs), choose the "Treat missing images as warnings" code gen option. For code generation without errors or warnings, Image objects should be referencing PNG files, JPEG, or 24-bit BMP. If the "Image object use files" option is enabled, Image objects should only be referencing PNG files since this is the only format supported by all targets that support loading Image objects from Image files at execution time.

- Incident #0936: DeepScreen – When generating code for Snapshot object, some targets have errors for Altia_Raster_Type private[] elements. This was fixed for the 10.2 release.

- Incident #0967: DeepScreen miniGL – Font string pixel width improperly calculated. This was fixed for the 10.2 release.

- Incident #0970 & #1283: DeepScreen – Enhancement: Vector Path object enhancements reduce memory usage and improve render performance. The Vector Path object is not supported on every DeepScreen target. Please see the separate User's Guide for a specific target to determine if it supports Vector Path objects.

- Incident #0975: DeepScreen – Enhancement: Each supported DeepScreen target is installed separately from the Altia Design software. In the Windows program group for each separately installed target, there is a User's Guide item and Enhancement Summary item. These documents are unique to the target. These documents should be studied to understand the capabilities of the target, any restrictions it might have, and how to take the generated code down to a real target device.

- Incident #0976: DeepScreen – When code is generated for a Vector Path object, it is missing information if the object is in a hidden card of a Deck. This was fixed for the 10.2 release. The Vector Path object is not supported on every DeepScreen target. Please see the separate User's Guide for a specific target to determine if it supports Vector Path objects.

- Incident #0980: DeepScreen miniGL – When multiple Text I/O objects share the same "text" animation, the optimization to determine if the string content of the Text I/O has actually changed (when a new string is passed to the "text" animation) is failing. It can fail to detect a real change in the string content. This was fixed for the 10.2 release.

- Incident #0994: DeepScreen – When a design contains a complicated Vector Path object or many Vector Path objects, the code generator is not setting the size for the ALTIA_INDEX variable type large enough. This causes some of the Vector Path data to be misunderstood at execution time and one or more Vector Path objects are incompletely rendered. This was fixed for the 10.2 release. The Vector Path object is not supported on every DeepScreen target. Please see the separate User's Guide for a specific target to determine if it supports Vector Path objects.

- Incident #1029: DeepScreen – Image Formatter heap corruption can cause a code generation session to crash in a small percentage of cases. This was fixed for the 10.2 release.

- Incident #1038: DeepScreen – Enhancement: When the "Image Object use files" code generation option is enabled, extra code is generated for reading PNG image files into Image objects at execution time. For Altia Design 10.2, this code has additional optimizations to reduce the number of memory allocations and frees during the reading of an image file. This reduces the memory fragmentation than can occur over a long period of time from many memory allocation and free operations.

- Incident #1090: DeepScreen – Enhancement: When the NO_PRELOAD option is defined, the default value for PRELOAD_CNT and PRELOAD_STENCIL_CNT are now 0. A 0 value indicates that no images should be preloaded. Instead, each image gets loaded when it first needs to render and every image stays loaded for the entire execution of the system. Previously, the PRELOAD_CNT and PRELOAD_STENCIL_CNT were set to 1 by default. This was impractical. It allowed only one image to load at a time. In other words, each previously loaded image was unloaded to allow the next image to load and this repeated for the entire execution of the system. For more about the preload options, see the earlier section *Controlling DeepScreen Image Data Loading with Preload options*.

- Incident #1093: DeepScreen – Enhancement: Automatically create an animation ID enumeration table at code generation time and write it to the generated file altia/altiaAnimationIdTable.h. If the user's application code calls the "animation ID" versions of the Altia API functions, it can #include the altia/altiaAnimationIdTable.h file to provide a predefined enumerated type for each animation ID value. Please see the generated altia/altiaAnimationIdTable.h file for details. The DeepScreen generated code and the user's application code must be compiled with AltiaUserAnimationIds defined (for example, with the compiler command line option: -DAltiaUserAnimationIds) to use the "animation ID" versions of the Altia API functions. Finally, the "animation ID" versions of the Altia API functions are only available for the DeepScreen version of the Altia API (not for the Altia Design/Runtime versions of the Altia API libraries).

- Incident #1156: DeepScreen – Enhancement: If user application code needs to know the raw event that becomes Altia stimulus, it can assign a function to the global function pointer variable AltiaReportEventPtr during user application code initialization. The function must take a byte argument. When the function is called, the byte argument will contain the event type value (e.g., ALTIA_DOWN_EVENT_TYPE as defined in altiaTypes.h). The function is called when it is known that a raw event will cause animations to execute, but before the animations actually execute. Here is the prototype for the callback function and extern declaration for the variable in DeepScreen that is set to register the callback function:

```
typedef void (*AltiaReportEventType)(signed char eventType);
extern AltiaReportEventType AltiaReportEventPtr;
/* These are the different event types as defined in altia/altiaTypes.h */
#define ALTIA_MOTION_EVENT_TYPE 0
#define ALTIA_DOWN_EVENT_TYPE 1
#define ALTIA_UP_EVENT_TYPE 2
#define ALTIA_KEY_EVENT_TYPE 3
#define ALTIA_KEY_UP_EVENT_TYPE 4
#define ALTIA_ENTER_EVENT_TYPE 5
#define ALTIA_LEAVE_EVENT_TYPE 6
```

User application code would define a function of the appropriate type and assign the address of the function to the extern AltiaReportEventType variable as in:

```
static void rawEventTypeDB(signed char eventType)
```

```
{
    printf("Altia Received raw event [%d]\n", (int) eventType);
}
. . .
/*
 * register callback with the Altia engine
 */
AltiaReportEventPtr = rawEventTypeCB;
```

- Incident #1158: DeepScreen – Enhancement: Provide a PRELOAD_IMAGEOBJ_CNT option in the spirit of the PRELOAD_CNT and PRELOAD_STENCIL_CNT options already available. The value defined for the PRELOAD_IMAGEOBJ_CNT option controls the preloading of image files into Image objects when the "Image Object use files" code generation option is enabled. For more details, see the earlier section *Controlling DeepScreen Image Data Loading with Preload options*.

- Incident #1185: DeepScreen – On Windows 7, pressing the Browse… buttons in the Code Generation options dialog can direct the user to the wrong folder for selecting a .bat Makefile script or an additional .gen file. This was fixed for the 10.2 release.

- Incident #1187: DeepScreen – Code generation can fail to process all Image objects if the current working directory is not the same as the directory containing the design (.dsn) file for which code is being generated. For example, the current working directory was changed as a result of importing an image from a different directory. This was fixed for the 10.2 release.

- Incident #1214: DeepScreen – Enhancement: For instructions and help with Unicode development, open the Unicode User's Guide from the Altia Design Help menu.

- Incident #1215: DeepScreen – Enhancement: For Altia Design 10.2, the Unicode version of the Altia Design editor is now the only available editor. To enable generation of Unicode (wide character) DeepScreen code, select the "Generate Unicode Font Characters" option in the Code Generation Options dialog. To generate 8-bit character DeepScreen code, do **not** select the "Generate Unicode Font Characters" option. By default, this option is **not** selected. Designs containing high/extended ASCII text characters or Unicode text characters should not be opened and saved in a non-Unicode editor. The high/extended ASCII text characters and Unicode text characters will be permanently garbled (i.e., will display as unexpected multiple characters for each high/extended ASCII or Unicode character).

- Incident #1235: DeepScreen – Use of the Language object requires wcsstr() in the generated altia/wCharStrings.c and altia/wCharString.h files, but it is missing. This was fixed for the 10.2 release.

- Incident #1281: DeepScreen – Enhancement: The Multi-line Text Object (MLTO) now supports the ability to set colors for individual characters. Individual words or letter colors can be specified as part of the MLTO text using the "ml_text" or "ml_character" animation. The color is specified in an html-like style tag such as:

  <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.

- Incident #1308: DeepScreen miniGL – Enhancement: miniGL direct memory drivers now have a chance to render raster images even with a 1-bit transparency mask (whereas previously such images were just ignored and not passed to direct memory drivers for rendering). It is up to the direct memory driver to provide the code to render an image with a transparency mask (most do not).

15 DeepScreen Known Issues for Altia Design 10.2

Individual targets may have specific known issues. Please see the target's separate Enhancements Summary and User's Guide for known issues. The issues described in this section are generally for all targets.

- Incident #0819: DeepScreen – Sound objects are not supported by most targets, but they exist in the button components of the standard models\buttons.dsn library. They are "turned off" by default so they do not make any sounds, but the code generator still attempts to generate data structures for Sound objects whether they are on or off at code generation time. A Sound object has a string for its sound file name. In rare cases, this string can be mistakenly processed at execution time as a string that needs to be rendered and can result in a message to stderr/stdout like:

  ```
  char 0x20 missing from font ...
  ```

  For a target that runs on Windows, the message appears in a Windows message dialog. A workaround is to create a simple Label that uses the same font (as shown in the message) and has the character that is shown to be missing (in the message above, character 0x20 is the space character). Another workaround is to remove all Sound objects from the design.

- Incident #1181: Altia Design/Runtime and DeepScreen – Language object content does not support double-quote (") characters and it does not support multiple lines of content for MLTO.

- Incident #1234: DeepScreen – A %FONTRANGEFLAG% entry in a .gen file can support up to 1024 ranges, but most .gen files have comments saying that only 16 ranges are supported. The comments in the .gen files should say that up to 1024 ranges are supported.

- Incident #1280: Altia Design – After opening a design file in a Unicode version of Altia Design and saving it, do not open the design file into a non-Unicode editor and save it if the design contains high/extended ASCII characters (character values 128 to 255) or Unicode characters (character values 256 to 6535) in Label objects, Text I/O objects, or MLTOs. The high/extended ASCII characters and Unicode characters will be changed to unexpected characters and cannot be recovered without manually changing each Label, Text I/O, or MLTO that contains the characters.

- Incident #1337: DeepScreen – The Multi-line Text object (MLTO) only generates its "ml_set_line" animation with a state of 0 even if it has a different current state at the time code is generated. The workaround is for application code or Control code to set the state of the animation to a value other than 0 at execution time. For a "ml_set_line" value of 0, new text entirely replaces the current text in the MLTO. For a non-zero value, new text only replaces text on the line specified by the value of "ml_set_line". For example, if "ml_set_line" has a value of 5, new text replaces just the text on line 5.

- Incident #1344: DeepScreen – If the character 'i' is not a character for which code has been generated (because a .gen file has a custom %FONTRANGEFLAG% that does not include it) and the Text I/O object "clear" animation is sent an event, a message will appear saying that character 'i' is missing from the font. This message goes to stdout/stderr for most targets, it displays in a Windows message dialog for targets that run on Windows. The workaround is to include the letter 'i' (character value 105, 0x69) in the custom %FONTRANGEFLAG% option of the .gen file.

- Incident #1363: DeepScreen – Changing the .bat file in the Makefile script: field of the Code Generation Options dialog does not necessarily force a recompile of all files. The intermediate object files should be removed before recompiling (such as with the "Code Generation > Make Clean" option in Altia Design if the target generates a working altmake.bat script).

- Incident #1376: Altia Design/Runtime and DeepScreen – Right justified Text I/O with just a single character string has a different right side alignment than a string with multiple characters. The workaround is to put a space character (" ") in front of the single character to make it multiple characters, but visually it still looks like a single character.

16 DeepScreen History for Altia Design 10.1 and Earlier Releases

This section describes new features/changes of the DeepScreen 6.0 code generator for the Altia Design 10.1, 10.0, and 9.2 software releases. Where applicable, the Altia Design release in which the feature/change was introduced or updated is noted as in ***(as of 10.1)***. There is also a later section *Enhancements and Fixes by Release and Incident Number* that summarizes DeepScreen changes back to the Altia Design 8.06 MR9 DeepScreen 5.0 release (September, 2008).

16.1 DeepScreen Intel x86 Software Render Windows Target (as of 10.1)

The **Windows** target (DeepScreen license 1001) in previous releases is replaced by the **Intel x86 Software Render Windows** target (also DeepScreen license 1001) in this release. This new target uses the graphics software rendering technology which more closely matches the rendering technology for other DeepScreen embedded targets. The Windows target in previous releases used the Windows GDI32 API. The new Intel x86 Software Render Windows target does not support multiple views (the **Generate code for multiple views** code generation option) or clones (the **Generate code for clones** code generation option). If these capabilities are required, please continue to use the **Windows** target in Altia Design 9.2 DeepScreen 5.0 or an earlier release. For more details about this target, please see the Intel x86 Software Render target's separate User's Guide.

16.2 DeepScreen OpenGL ES 1.1 Targets (as of 10.1)

DeepScreen code generation is possible for the OpenGL ES 1.1 standard for Freescale i.MX31 running Linux, Freescale i.MX51 running Linux, and Renesas V850E2/DR4-3D not running an OS. OpenGL ES 1.1 is a 3D graphics interface specification from the Khronos Group (http://www.khronos.org/opengles/). All relevant graphics acceleration features are utilized for superior drawing performance. OpenGL ES 1.1 targets do **not** support the Snapshot object.

Each OpenGL ES 1.1 target (e.g., i.MX31, i.MX51, V850E2/DR4-3D) is an optional purchase requiring a unique DeepScreen license.

Altia Design 10.1 and 10.0 OpenGL ES 1.1 Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on OpenGL for details.

Altia Design 9.2 OpenGL ES 1.1 Improvements:

- Support was added for the Freescale i.MX51 running Linux. To date, this port does **not** include Altia OpenGL (3D) object support.

Altia Design 9.1 and 8.08 MR3 OpenGL ES 1.1 Improvements:

- Support for the Altia OpenGL (3D) object was added to the Freescale i.MX31 running Linux when generating code for floating point (not fixed point).
- If generating code for native Windows OpenGL, define WIN32_OPENGL_SCREEN_REDRAW as 0 at compile time or in egl_md.h to do faster dirty region redrawing. The default behavior is for WIN32_OPENGL_SCREEN_REDRAW to be defined as 1 in egl_md.h to do slower full window redrawing because this yields the most consistent results on most Windows desktop computers.

Each OpenGL ES 1.1 hardware target has its own target User's Guide. Please see the target User's Guide for more details (such as how to configure the development environment, supported code generation options, support for special objects, how to compile generated code, and how to run it on a target device).

16.3 DeepScreen OpenGL ES 2.0 Targets (as of 10.1)

DeepScreen code generation is possible for the OpenGL ES 2.0 standard for Freescale i.MX51 running Linux and Freescale i.MX53 running Linux. OpenGL ES 2.0 is a 3D graphics interface specification from the Khronos Group (http://www.khronos.org/opengles/). All relevant graphics acceleration features are utilized for superior drawing performance. Please note that OpenGL ES 2.0 targets do **not** support the Altia OpenGL (3D) object. OpenGL ES 2.0 targets can support the Snapshot object.

Each OpenGL ES 2.0 target (e.g., i.MX51, i.MX53) is an optional purchase requiring a unique DeepScreen license.

Altia Design 10.1 and 10.0 OpenGL ES 2.0 Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on OpenGL for details.

Each OpenGL ES 2.0 hardware target has its own target User's Guide. Please see the target User's Guide for more details (such as how to configure the development environment, supported code generation options, support for special objects, how to compile generated code, and how to run it on a target device).

16.4 DeepScreen OpenVG 1.1 Targets (as of 10.1)

DeepScreen code generation is possible for the OpenVG 1.1 standard for Freescale i.MX51 running Linux, Freescale MPC5645S (Rainbow), and Renesas SH7268. OpenVG is a vector graphics interface specification defined by the Khronos Group (http://www.khronos.org/openvg/). All relevant graphics acceleration features are utilized for superior drawing performance.

Each OpenVG 1.1 target (e.g., i.MX51, MPC5645S) is an optional purchase requiring a unique DeepScreen license.

Altia Design 10.1 and 10.0 OpenGL VG 1.1 Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on OpenVG for details.

Each OpenVG 1.1 hardware target has its own target User's Guide. Please see the target User's Guide for more details (such as how to configure the development environment, supported code generation options, support for special objects, how to compile generated code, and how to run it on a target device).

The OpenVG specification does not support fixed point graphics data so a target device must have floating point support.

Certain gradients in images are not converted to RGB565 effectively and this can cause a banding effect on some target devices. This might also be the cause for anti-aliasing not working for some images.

An OpenVG targets may have support for the Snapshot object. See the target's separate User's Guide for details.

16.5 DeepScreen Freescale Spectrum DCU Target (as of 10.1)

For Freescale Qorivva MPC560xS family of 32-bit Power

Architecture microcontrollers (MCUs) that address TFT displays in automotive instrument cluster applications.

This target requires the purchase of DeepScreen license 1028.

The Altia objects and dynamic behaviors supported by this target are similar in scope to those supported by the miniGL target.

For documentation, please see the Freescale Spectrum DCU target User's Guide.

Altia Design 10.1 and 10.0 Freescale Spectrum Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on Spectrum for details.

16.6 DeepScreen Fujitsu Jade Custom2D Target (as of 10.1)

For Fujitsu MB86R01 'Jade' device. This is an integrated ARM926EJ-S CPU and 'Coral PA' graphics display controller (GDC). All relevant features of the GDC are utilized for superior drawing performance. QNX OS 6.3.0 is supported as well as Linux (tested with Fujitsu Linux SDK V0.8.3). The target is portable to other operating systems.

This target requires the purchase of DeepScreen license 1022.

For documentation, please see the Fujitsu Jade Custom2D target User's Guide.

Altia Design 10.1 and 10.0 Fujitsu Jade Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on Fujitsu Jade for details.

Altia Design 9.2 Fujitsu Jade Improvements:

- The generic (no OS) port has these improvements: **(1)** Now it defines AltiaUseAnimationIds to 1 by default. This enables the version of the DeepScreen Altia API that uses id numbers for animation names (instead of strings) for improved performance. **(2)** Support was added to the generic port for 480x272 display resolution. **(3)** There is a change to the generic port to resolve a register offset issue when applying the patch code for Jade Errata E5 (Memory Access Not Coherent). **(4)** A syntax error in the generic port driver function driver_pixmapAlloc() is resolved.
- This target now supports the **Generate external resource files** code generation option.
- Snapshot object support is added.
- Generated code was modified to resolve Fujitsu Jade Errata V1.4 issue #E5 so that missing pixels do not appear in polygons.
- The Linux port now supports the "DDB" uncompressed image data code generation mode. See one of the Fujitsu Jade DDB .gen files for an example usage.
- Now there is no dynamic memory allocation for the "DDB" uncompressed image data code generation mode when static memory code generation is chosen (that is, when **not** selecting the **Use Dynamic Memory** code gen option).
- There is improved rendering for distorted images by breaking up the raster surface into more triangles.
- This target now rejects compiling generated code for the **Full Screen Mode** code generation option. Fujitsu Jade always compiles code for full screen rendering and selecting this option is not compatible.
- The color buffer could be incomplete if alpha-map color was unchanged from one draw to the next, but the first draw used less of the color buffer because it had a smaller stride. This is fixed.

Altia Design 9.1 and 8.08 MR3 Fujitsu Jade Improvements:

- Added Hardware APIs altiaHardwareBlocked() (returns true if hardware would block because it is out of render lists) and altiaHardwareActive() (returns true if hardware is rendering). See the Fujitsu Jade target document Overview.doc file for more details.
- Multiple display controller support was added including support for cursors and a special mode for a single display controller to drive two displays simultaneously. See the Fujitsu Jade

target document Overview.doc file for more details.
- A uties folder was added to the usercode/fujitsu target directory. It contains example code for loading rasters in the background.
- Added a "generic" port including Green Hills MULTI project files. This can be used as a starting point for porting to a custom operating system. See the Fujitsu Jade target document Overview.doc file for more details.
- Using a transparent color on layer 0 may show the color layer which is blended with the Alpha layer. This is just how the Jade behaves. To eliminate the problem, the transparent color is removed from layer 0 altogether (since layer 0 is on the bottom, this is OK to do).
- If ALTIA_TOUCH_WIDTH and ALTIA_TOUCH_HEIGHT are not defined, they get defined as ALTIA_SCREEN_0_WIDTH and ALTIA_SCREEN_0_HEIGHT to force input code to use display 0 as the default touch screen input device.
- Added FreeType font support. See the Fujitsu Jade target document Overview.doc file for details. The FreeType software is copyright © 2006-2008 The FreeType Project (www.freetype.org). All rights reserved. This notice satisfies The FreeType Project LICENSE agreement requirement to acknowledge The FreeType Project when its software is included in a commercial product.
- Added variable point touch-screen calibration routine to Linux driver. See the Fujitsu Jade target document Overview.doc file for details.
- Removed DDB Alpha Layer check in egl_BitmapBlt() so some blits will work with the Alpha Layer and DDB.
- Force drawable width to a multiple of 8 as required by the graphics device.
- Improved Linux driver synchronization between mapped and physical memory.
- For Linux, compile Unicode generated source files with -finput-charset=ISO-8859-1 to support upper ANSI character range for strings in data.c and control.c.

Altia Design 8.08 MR2 Fujitsu Jade Improvements:

- Changes to resolve missing pixels for rectangle objects with an opacity less than 100%.
- A stride calculation error is resolved that would previously make the blend bitmap too small and generate the error message "Insufficient scratch space for monochrome bitmap!".
- A layer update synchronization issue is resolved because sometimes the VSync signal can have a high latency and this may be the VSync that occurred just prior to the last layer update.
- Improved image delete logic.
- Unicode support added. Select altmake_unicode.bat or altmake_linux_unicode.bat as the Makefile script at code generation time.
- Support for input (mouse/touchscreen) stimulus especially for targets running Linux or QNX.
- For Linux, a VSync issue is resolved that would cause the screen to flicker.
- For Linux, a memory manger was added to keep all memory used by DeepScreen generated code in a single 64 MByte page as required by the graphics controller.
- For Linux and QNX, libpng and zlib object libraries are provided for out-of-the-box support of the "Image object use files" option at code generation time if the target has a file system.

Altia Design 8.075 Fujitsu Jade Improvements:

- For Linux code generation, the pixel data for images is split into 4194304 byte arrays to improve GNU compiler performance.

Altia Design 8.07 MR4 Fujitsu Jade Improvements:

- Added a Linux port tested with Fujitsu Linux SDK V0.8.3. See the Fujitsu Jade target documentation for general instructions and see usercode\fujitsu\linux\readme.txt for instructions on building and installing the required Altia Kernel Module into the Linux kernel.
- Blits to frame buffer are suppressed and an error message is printed if the source address is not in the same 64 Mbyte segment as the frame buffer as required by the device.
- Transformed image blits are suppressed and an error message is printed when transforming an image that does not have power-of-two dimensions.
- Changes to always blit on 64-byte boundaries as required by the device.

- Changes for when the display width in bytes is not a multiple of 64.
- Changes for 2D color register having incorrect value after using the 3D color register to draw wide lines.
- Disabled bilinear filtering for blitting an untransformed image that has transparent pixels to suppress blurring of the image especially for opacities less than 100%.
- Corrected width/height off-by-one for blitting an image if it has transparent pixels.
- Improved Stencil object rendering.
- Minor changes to comply with Jade GA spec.
- Driver does not initialize DDR FIFO register because it is not graphics hardware related.
- Driver now uses generic PC monitor timings for display resolutions of 320x200 through 1280x1024 instead of the previous single resolution 1280x480 display timings and reference clock value is 658.7MHz to match Jade development hardware.

Altia Design 8.07 MR3 Fujitsu Jade Improvements:

- Gracefully ignore drawing text characters for which there is no character bitmap.
- Improved rendering for very large polygons.
- Improved rendering for wide lines.

Altia Design 8.07 MR2.1 Fujitsu Jade Improvements:

- A lockup issue was resolved when trying to delete an image that is in the current render list. **To get this very important change, upgrade to 8.07 MR2.1 or newer**.

Altia Design 8.07 MR1 Fujitsu Jade Improvements:

- Improved dirty region copy to not use transparent color.
- Improved response to zero extent list.
- Added layer API for overriding default layer parameters. The driver_layerPosition() function supports moving layers off the screen.
- Transparency masks are allowed for rotated/scaled Raster objects. Previously, the masks were not processed when rendering rotated/scaled Raster objects.
- Transparency mask fix for non-transformed Raster objects.
- Concave polygon support.
- Render list optimizations.
- Inline fixed point math optimizations for 10-15% better performance for fixed point generated code.
- Relinquishes driver resources on AtStopInterface(), AtCloseConnection(), or altiaDisconnect() Altia API calls.

16.7 DeepScreen Fujitsu Carmine Custom2D Target (as of 10.1)

For Fujitsu MB86297 'Carmine' graphics display controller (GDC).

This target is portable to a variety of host processors with an example port to PowerPC 603e running ThreadX. All relevant features of the GDC are utilized for superior drawing performance.

For documentation, please see the Fujitsu Carmine Custom2D target User's Guide.

Altia Design 10.1 and 10.0 Fujitsu Carmine Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on Fujitsu Carmine for details.

Altia Design 9.1 and 8.08 MR3 Fujitsu Carmine Improvements:

- Added Hardware APIs altiaHardwareBlocked() (returns true if hardware would block because it is out of render lists) and altiaHardwareActive() (returns true if hardware is rendering). See the Fujitsu Jade target document Overview.doc file for more details.
- Multiple display controller support was added including support for cursors and a special

mode for a single display controller to drive two displays simultaneously. See the Fujitsu Jade target document Overview.doc file for more details.

- A uties folder was added to the usercode/fujitsu_carmine target directory. It contains example code for loading rasters in the background.

Altia Design 8.07 MR4 Fujitsu Carmine Improvements:

- Added maximum memory check for pixmap allocations to avoid going beyond 128 Mbyte video ram capacity.

16.8 DeepScreen Intel x86 Software Render Linux Target (as of 10.1)

This target is for Linux with only a framebuffer. In releases before Altia Design 10.1 DeepScreen 6.0, this target was named Linux Framebuffer. The name change is the only difference.

All transformed drawing is performed in software.

This target requires the purchase of DeepScreen license 1032.

For documentation, please see the Intel x86 Software Render Linux target User's Guide:

Altia Design 10.1 and 10.0 Intel x86 Software Render Linux Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on Intel x86 Software Render Linux, altiaGL, or Linux Framebuffer for details.

Altia Design 9.2 Linux Framebuffer Improvements:

- Support for the Freescale i.MX51 device using the Freescale SDK 10.04.01. Select an imx51 Makefile script and Additional Target files list .gen file at code generation time.

Altia Design 9.1 Linux Framebuffer Improvements:

- The altmake_linuxfb_unicode_gcc.bat Makefile script for compiling generated code as Unicode now uses the compiler command line option -finput-charset=ISO-8859-1. This enables support for the upper ANSI character range in strings generated to data.c and control.c.

Altia Design 8.07 MR3 Linux Framebuffer Improvements:

- Support for ST SPEAr device. This is a device with an ARM926EJ-S processor core. Code compiles using the Linux SDK provided by ST (an arm-linux-gcc toolchain). Choose the altmake_linuxfb_SPEAr.bat Makefile script at code generation time.

Altia Design 8.07 MR1 Linux Framebuffer Improvements:

- Support for Freescale i.MX31 running Linux is expanded to include the **(1)** Freescale MCIMX31ADSE/C development board with keyboard and MXC ¼ VGA touch screen display and **(2)** Freescale i.MX31 "3-stack" Product Development Kit (PDK) development board with keyboard and full VGA (480 x 640) touch screen display. At code gen time, choose Makefile script altmake_linuxfb_imx31ads.bat for (1) or altmake_linuxfb_imx31pdk.bat for (2).

16.9 DeepScreen miniGL (as of 10.1)

This target generates C source code to deploy on very simple devices with pixel-based displays. The generated C source is deployable to any system independent of its OS (especially systems that have no formal OS). The code produced by the miniGL target is extremely compact and this imposes some unique restrictions on the design. For documentation, please see the miniGL target User's Guide.

VERY IMPORTANT MINIGL CHANGE FOR THIS RELEASE:
Previous releases of miniGL will pack empty Deck objects. In this release, the packDynamic.exe program will throw an error, show a message like **Unsupported animation type [0] on object #**

33301, and the empty Deck object will get selected in Altia Design. This Deck object must be deleted and then code can be generated again and packed. When all empty Decks have been deleted, the packing can complete successfully.

For documentation relevant to the features of the latest releases, please see the miniGL target User's Guide.

Altia Design 10.1 and 10.0 miniGL Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on miniGL for details.

Altia Design 9.2 miniGL Improvements:

- Support is added to miniGL target devices for custom animation states on Text I/O and Raster objects. Animation states to change foreground color (for Text I/O), opacity, X/Y position, and hide/show (map/unmap) can now be assigned from the Altia Design Animation Editor and the miniGL generated code will support these animations. As is expected, opacity and position are interpolated between states.
- The lcdWin32 simulator target is updated to support the **Make with Editor API** compile option and to also support the new custom animation and language object enhancements.
- Language object support is added.
- FreeType font integration is now possible.
- There is now support for the Drawing Area object (DAO) with limitations.
- Improved merging of damaged areas for clipped drawing (especially when a new damage area completely encapsulates an existing damage area).
- The dobjsc array is now pushed into .const data rather than the .data segment (RAM) for a significant improvement in RAM memory usage.
- Support in paged memory devices to split stencil/raster/mask bits data across defined ROM page boundaries when there are too many of these objects to fit the data in a single ROM page.
- If a Text I/O object changes, but it is hidden (such as in a card of a Deck that is not showing), now the area of the Text I/O is not redrawn.
- A fix was made for a "dancing text bug" when more than one Text I/O has the same base name.
- Changes were made to isolate negative values from bit fields when data structures are being packed.
- There was a fix for a dynamic raster index bug. Previously for a design with a mix of static rasters (imported images) and dynamic rasters (images showing via Image objects or an imported image with a custom animation), the raster index could be incorrectly computed.
- True layer support is added. If objects in the design have an **Object Layer** property, the generated code will have layer support.
- MINIGL_LAYER_REDRAW compiler directive (#define) specifies that all objects within a layer will be redrawn when one or more have changed. Similar to clipping but based upon layers instead of dirty regions. Only layers that experience one or more object changes will be redrawn. This is required for a miniGL MPC560x target since it uses multiple layers and all objects within a layer must be processed when any one object changes.
- MINIGL_LAYER_MASK compiler directive (#define) masks a layer ID used in Altia Design. For the purposes of processing layers, the Altia Design layer ID will be masked with this value. This allows for special bit flags to be used in the layer IDs. Example: a layer mask of 0xff would treat the following layers all as layer 1: 0x0101, 0x0001, 0x0201. This is required for a miniGL MPC560x target since bit flags are used for tiling and GRAM copy.
- Layer identifiers are now supported by miniGL. This is defined by the miniGL .ali file .layerids token.
- The .align token in the miniGL .ali file is enhanced to support different alignments for raster, stencil, alphamask, monomask, alphafont, and monofont data. Previously, all such resources could only be aligned the same (on 8, 16, or 32 bits). See one of the updated miniGL .ali files for a description of this enhancement..
- Raster or alpha masks can now be externalized as a whole by leaving off the mask value in the

miniGL .ali file .external rasters token or .external alphamasks token.

- Pixel mode support is added (in contrast to address mode). This is required by some custom targets, mainly hardware accelerated solutions. This is enabled by the miniGL .ali file .gcmode token.
- Added the ability to turn off bit counts and report width or height information. This is required by some custom targets, mainly hardware accelerated solutions. This is defined by the miniGL .ali file .nobitcounts token.
- Custom encoder plug-in DLL hook added to allow flexibility during the code gen packing process. This is defined by the miniGL .ali file .custom encoder token.
- Images can now be padded, both width and/or height. This is required by some custom targets, mainly hardware accelerated solutions. This is defined by the miniGL .ali file .pixelpad token.
- The opaque stencils optimization can be turned off whereby they will be stored as normal stencil images. This is required by some custom targets that cannot dynamically draw data such as sprite based targets. This is defined by the miniGL .ali file .noopaquestencils token.
- The checksum value generation can be turned off. Some custom targets cannot accommodate a checksum value due to alignment restrictions. This is defined by the miniGL .ali file .external nochecksums token.
- Chromakey color definition is added (this is a color to make transparent). Used for padding images and replacing RGB pixel data that has an alpha value of 0 (transparent). This is defined by the miniGL .ali file .chromakey token.
- Anti-aliased fonts can now be configured up to 8 bits per pixel. Previously only 8-bpp was supported. This is defined by the miniGL .ali file .alphafonts token.
- Alpha values for fonts and rasters can now be inverted. Some custom targets require this. This is defined by the miniGL .ali file .alphavalues inverted token.
- Raster alpha values can now be interlaced (inlined) into the RGB data. When interlacing is turned on, the alpha mask (if any) directly follows the RGB data instead of being stored in a separate data block. This is defined by the miniGL .ali file .interlaced token.
- Static colors are now optimized into a common table to save ROM. Optionally the index value can be preserved and provided to the miniGL driver layer via the graphics context. This is defined by the miniGL .ali file .colors keepindex token.
- Indirect color moding support has been added. miniGL automatically determines the best fit for a particular raster image and will encode it accordingly. Optionally, images can be constrained to an indirect color mode. Likewise, color lookup tables can be constrained. A color saver option has also been introduced to reduce the size of encodings. These features are enabled by the various miniGL .ali file .indirect tokens.
- A pre-flush function, miniGLDriverFlushing(), has been added. Useful for hardware accelerated targets.
- Support for a MINIGL_TEXT_CHARACTER_FLAGS compile option is added for more flexible processing of static text strings. Some custom miniGL targets require this. Enabling this option identifies first and last characters of a text string. This is necessary when special consideration is required for processing whole strings of text.
- Special operations are now allowed for dealing with text for custom miniGL targets. The maximum Text I/O and Label character width is now made available at the driver level. The value is defined by MAX_TEXTIO_ENCODING_LENGTH in the miniGLDriver.h header file.
- For the DAO simulator target, support for defining DAOWIN32_DEBUG_EXTENTS at compile time (as in the command line option -DDAOWIN32_DEBUG_EXTENTS) was added to draw a bounding box around the dirty rectangles for internal debugging/development.
- For the DAO simulator target, a new event "miniGLUpdateFBDone" is generated after a frame buffer update for external application code to monitor the frame buffer (perhaps to capture it for testing purposes).

Altia Design 8.08 MR2 miniGL Improvements:

- The 16-bit horizontal framebuffer driver is changed to correctly draw overlapped objects when the stride is 0.

- Text I/O objects with the same "text" animation now draw when the text changes.
- The DAOWin32 simulator window now always opens in front of the Altia Design window when doing "Run on PC" from the Altia Design "Code Generation" menu.
- The DAOWin32 simulator now supports the "Make with Editor API" option from the Altia Design "Code Generation" menu.
- For the RenesasH8SX target, the default framebuffer style is now non-inverted.
- Resolved a benign compiler warning when the clear frame operation is not used.
- Added NULL Text I/O string check to satisfy static analysis.
- Resolved benign packing phase compiler warnings to support Microsoft compiler maximum warning level /W3.
- Resolved a variety of benign compiler warnings during the "Pack miniGL Static/Dynamic Data" phase when the Windows host compiler is Visual Studio 2005 or newer.
- The %DATECHECKFILE% option is now used in .gen files to copy binary/object/DLL files so that newer versions overwrite older versions.
- Added CRC16 code generation to validate external flash files.
- Images now always draw correctly at execution time when the device dependent raster image is externalized and the corresponding alphamask data is kept internal.
- Cortex M3 target optimized driver pipeline with generic DHAL implementation, alpha blending (read back) corrected, and minor bug fixes.
- Cortex M3 target changes to setRasterResourceType() and setMaskResourceType() to accommodate efficient serial flash caching.
- Cortex M3 target provisions for inlining display HAL API functions as well as serial flash HAL API functions.
- Cortex M3 WIN32 simulator added that uses existing DAOWIN32 simulator with the Cortex M3 driver pipeline.
- Cortex M3 target change to build code with -Ohs (speed) optimization level.

<u>Altia Design 8.075 miniGL Improvements:</u>

- For 16-bit horizontal driver, changes to support Renesas H8SX and Renesas HEW compiler. In miniGLClearFrame(), force 16-bit calculations and do full screen update by scanline instead of a linear progression.
- A packing phase improvement to handle animations that are only used in stimulus definitions.
- DAO Win32 simulator now updates the frame buffer at startup so the main() routine does not need an explicit altiaFlushOutput() call to force an update.

<u>Altia Design 8.07 MR4 miniGL Improvements:</u>

- For encoded bits, swapping on a single byte boundary is fixed.

<u>Altia Design 8.07 MR1 miniGL Improvements:</u>

- New DAO WIN32 simulator for simulating direct color displays (i.e. 16-bit) and stimulus support.
- HC12 Cosmic support continues.
- LCD WIN32 simulator support continues (no stimulus support, use the DAO WIN32 simulator for stimulus support).
- New Renesas H8SX support using HEW tool chain.
- New STMicroelectronics ARM Cortex-M3 support using IAR tool chain.
- Compression of raster images, fonts and stencils. This is enabled with the .compress option in the miniGL target's .ali file.
- Object clipping improves drawing performance. This is enabled with the .clipping option in the miniGL target's .ali file.
- Direct memory model (non-paged architecture for exceptional speed increase) and still supports a paged memory model as in previous releases. A paged memory model is enabled with .pagesize options in the miniGL target's .ali file (no .pagesize options implies a direct memory model).
- Limited stimulus support for touch screen displays. This is left mouse down or up stimulus on

any object except a static text Label or Text I/O object. The stimulus can be Rectangle or Whole area; however, on a Deck, the stimulus must be Whole area. The stimulus can execute any animation name and state (for example, it can be an animation name that is only processed in external application code using altiaNextEvent()). The execute must be a State value, not an Inc or Dec value. An enable condition is not yet supported (but hopefully will be in the near future). If a non-supported stimulus is detected during packing, it is indicated with a warning message. The packing process will not fail, but unsupported definitions are ignored. **For generating designs from PhotoProto that have stimulus, use PhotoProto 1.05**. The stimulus generated by PhotoProto 1.06 or newer is currently beyond the stimulus capability of this miniGL release.

- Additional API functions for stimulus support including altiaPending(), altiaNextEvent(), altiaSelectEvent(), and altiaSelectAllEvents().
- Optional anti-aliased text support. This is enabled by choosing the **Generate anti-aliased fonts** option from the Code Generation Options dialog at code generation time.
- Controllable bit field packing (0, 8, 16, and 32) using the .maxbitfield option in the miniGL target's .ali file.
- Specify a default character with the .defaultfontchar option in the .ali file. This character is used when a character value in a string from altiaSendText() is not recognized. This can happen if the character was not encoded at code generation time (perhaps because the font does not contain the character).
- Very fast binary search mechanism for font character lookup and fixed issue when index becomes negative.
- Image dithering for nicer looking graphics. This is enabled with the .dithering option in the miniGL target's .ali file.
- Driver layer can now use function pointers for added flexibility. This is enabled with the .usefunctionpointers option in the miniGL target's .ali file.
- Support for external (i/o based) flash technologies (stencils, fonts, rasters, and alphamasks). This is enabled with .external options in a miniGL target's .ali file.
- Generic 16-bit direct color "relative" frame buffer driver now available. The existing "absolute" type is also still supported. The type is determined by the style of the color table in the miniGL target's .ali file.

16.10 DeepScreen Toshiba Capricorn M Custom2D Target (as of 10.1)

For Toshiba TX4961 with CAPM graphics acceleration or Toshiba TX4964 with CapA graphics acceleration. All relevant graphics acceleration features are utilized for superior drawing performance. This target requires the purchase of DeepScreen license 1021.

This target also offers support for using the FreeType font engine for text rendering at execution time.

The FreeType software is copyright © 2006-2008 The FreeType Project (www.freetype.org). All rights reserved.

The above notice satisfies The FreeType Project LICENSE agreement requirement to acknowledge The FreeType Project when its software is included in a commercial product.

For documentation, please see the Fujitsu Carmine Custom2D target User's Guide.

IMPORTANT NOTE: If the message "ERROR - Command List Overflow (possible memory corruption)" displays on the target (a working stdout is needed to see this message) and no graphics draw, the render command list size is too small for the HMI. The default size is 4000. To increase this, set the ALTIA_RENDER_LIST_SIZE compiler flag in the build project and rebuild all object files. For example, try: -DALTIA_RENDER_LIST_SIZE=15000

<u>Altia Design 10.1 and 10.0 Toshiba Capricorn Improvements:</u>

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on Toshiba Capricorn for details.

Altia Design 9.2 Toshiba Capricorn Improvements:

- The image dithering pseudo-noise function was made in-line to improve performance.
- Image decompression code now sets transparent pixels to black (and skips a color calculation entirely) for improved performance.
- Display timings were modified to add support for a 1280x800 PC monitor.
- There is now a Green Hills Probe RBTX4961_AOB.mbs target initialization script for the Capricorn-AOB (Application Oriented Board) development board. This file is generated when a "tx4961" .gen file is chosen.

Altia Design 8.07 MR3 Toshiba Capricorn Improvements:

- Gracefully ignore drawing text characters for which there is no character bitmap.

Altia Design 8.07 MR1 Toshiba Capricorn Improvements:

- Clip region fixes for bitmap, line, polygon, and rectangle.

16.11 DeepScreen text rendering speed improvements (as of 10.1)

When text characters (e.g., for Labels, Text I/Os, or MLTOs) are generated, they are now ordered exactly by their character code values. This allows for a very fast binary search at execution time. In addition, a missing character code is detected just as quickly as a valid character code.

As part of this improvement, code generated from the Altia Design 8-bit character editor (i.e., not Unicode) must be compiled for 8-bit character encoding (i.e., the UNICODE macros cannot be defined). And code generated from the Altia Design Unicode editor must be compiled for Unicode character encoding (i.e., the UNICODE macros must be defined). If generated code from the Altia Design 8-bit character editor is compiled for Unicode, the compile will abort with a message: `UNICODE compile time definition not allowed because this code was generate using the ASCII Altia Editor`. If generated code from the Altia Design Unicode editor is compiled for 8-bit character encoding (i.e., UNICODE macros are **not** defined), the compile will abort with a message: `UNICODE compile time definition required because this code was generated using the Unicode Altia Editor.`

16.12 Generate External Files for Loading at Execution Time (updated for 10.1)

Use the new **Generate external resource files** option at code generation time to generate all image and font data to binary files instead of the source code files such as `altia/data.c` and `altia/<TARGET>/egl_font.c`.

Raster object pixel values, font character images, and Stencil object and mask bits are generated to a series of `altiaImageDataBank[0-N].bin` files. Typically, each `.bin` file has a maximum size of 4 Mbytes unless a single image is greater than 4 Mbytes because it must reside in a single `.bin` file. The maximum size is configured in the `.gen` files for each target and some targets use a different maximum size (like the Renesas V850E2/DX4-H uses 64 Mbytes).

If the **Generate external resource files** option is chosen at code generation time, the default behavior of the generated code at execution time is to open each file with `fopen()`, allocate a block of memory large enough to hold the data from the file (the data size is determined at code generation time and included in the generated `altia/data.c` file), read the data from the file into an allocated block of memory using a single call to `fread()`, and close the file with `fclose()`.

The `.bin` files contain data exactly in the format the data would have taken if it had been generated to the `data.c` and `egl_font.c` files. For example, if a DDB `.gen` file is chosen at code generation time, the data is in the defined device dependent format. Otherwise, the data is in

Altia's custom compressed format. If the **Image object use files** option is also enable at code generation time, then images are still dynamically loaded from the individual image `.png` files (or image `.ddb` files for a DDB_FILES or DDB_RLE_FILES `.gen` file).

Using this **Generate external resource files** feature generates a `data.c` file that is much smaller (it compiles faster and takes up much less program space on the target).

The use of `fopen()`, `fread()`, and `fclose()` requires a target environment that supports these functions. The generated `.bin` files must be in the working directory at execution time. In other words, only a file name (such as `altiaImageDataBank0.bin` without any path information) gets passed to `fopen()`.

The file processing can be fully customized by defining `ALTIA_DRV_LOADASSET` at compile time (as in a `-DALTIA_DRV_LOADASSET` option to the compiler). When `ALTIA_DRV_LOADASSET` is defined, the DeepScreen generated code calls these functions to do the work instead of calling `fopen()`, `fread()`, and `fclose()`:

```
        void *driverLoadAsset(char *filename, int size)
        /* To load "size" bytes from file named "filename" and return
         * the pointer to the buffer holding the data.  Please note
         * if the data is in device dependent format, some targets
         * may require it to be in contiguous physical memory (versus
         * memory managed process memory that may not be physically
         * contiguous) and properly aligned (such as on a 64-byte
         * address boundary).  For example, Fujitsu Jade has these
         * requirements.
         */

        void *driverFinishAsset(void *asset, int size)
        /* To finish any required driver operations after the buffer
         * "asset" has been decompressed by the DeepScreen code.
         * The "asset" parameter will be the same address returned
         * by the earlier call to driver_loadAsset().  This
         * driverFinishAsset() function can optionally free that
         * memory if it is no longer needed.
         *
         * NOTE:  The memory is still needed when:
         * 1. Using uncompressed image storage
         * 2. Using compressed images with NO_PRELOAD
         * 3. Using software based scaling/rotation
         *
         * Return NULL if the asset was freed.  Otherwise, return
         * the same asset pointer.
         */
        void driverFreeAsset(void *asset, int size)
        /* To free the allocated data buffer when the DeepScreen code
         * is cleaning up such as during a call to one of the Altia
         * API functions altiaDisconnect(), AtCloseConnection(),
         * or AtStopInterface().
         */
```

Open the generated `altia/altiaImageAccess.c` file to see the actual implementation for this feature. All targets with a file system implement this feature. Other targets can have custom implementations on an as-needed basis.

16.13 Support for Distort, Language, Skin, and Snapshot Objects (as of 10.1)

The new Distort feature, Language object and Skin object are supported for a variety of targets. For details about this new feature and these new objects, please see the ***Altia Design User's Guide***. To determine if a specific DeepScreen target supports one or more of these features, please see the target's separate User's Guide. The Snapshot object has DeepScreen limitations as described below.

As in previous releases, the miniGL target does not support transformed objects and it does not support the new Distort feature because it is a type of transform. It supports a limited set of special objects such as the Deck and Text I/O, but not other complicated special objects and not the new Skin and Snapshot objects. These miniGL limitations are necessary to minimize the code size for execution on very simple embedded targets.

For the 10.1 release, Snapshot object support was expanded to include Intel x86 Software Render Windows, altiaGL, Altera D/AVE, Fujitsu Jade, Intel x86 Software Render Linux, some OpenGL ES 2.0 targets, and some OpenVG 1.1 DeepScreen targets. Support for other DeepScreen targets may be arriving in future releases. As of the 9.2 release, the Snapshot object can capture the OpenGL object on targets that support this special object and it can capture the Drawing Area object (DA) on targets that support this special object.

There are also the following restrictions for the DeepScreen implementation of the Snapshot object:

- For DeepScreen, capture by "Location" (capture animation = 2) is not implemented.
- If a "DDB" .gen file is chosen at DeepScreen code generation time, the Snapshot object will not render if it is transformed (scaled/stretched/rotated/distorted) especially for altiaGL targets (and targets based off of altiaGL such as Intel x86 Software Render Windows/Linux).
- DeepScreen altiaGL and targets based off of altiaGL (such as Intel x86 Software Render Windows/Linux) only support the "Software" alpha mode (alphaMode animation = 2).

16.14 Image storage for all DeepScreen Targets (as of 10.0)

Image data is now partitioned into multiple image data files instead of the single `data.c` source code file. This does not change the functionality of the DeepScreen generated code. However it does make the code friendlier for compilers when code is generated for very large HMI designs.

16.15 DDB image file option for some Software Render and Fujitsu Jade targets (as of 10.0)

Some software render targets and the Fujitsu Jade targets have additional `.gen` file options when generating code. The additional .gen files end with `_ddb_files.gen` or `_ddb_rle_files.gen`.

This feature makes device dependent bitmap files (`.ddb`) out of all images used in Image objects of the `.dsn` file when the **Image object use files** code generation option is enabled. This provides a faster image load mechanism than the conventional `.png` file. The `.ddb` image files are created during code generation for each image file that is referenced by an Image object at the time of code generation. If there is a directory path included in the image file name, the `.ddb` file is written to the same directory. It is important that the `.ddb` files get moved to the embedded target while preserving the same directory structure as on the Windows PC.

If the **Image object use files** code generation option is **not** enabled, generating code with a `_ddb_files.gen` or a `_ddb_rle_files.gen` is the same as generating code with a `_ddb.gen`. That is, Image object data is generated into the code itself in DDB format, not as external files.

At execution time of the generated code, when an Image object needs to load its image or its `image_name` animation is changed, the generated code first tries to open/load the `.ddb` version of the image file. If it does not exist, it tries to open/load the `.png` version of the image file.

The performance comes at the cost of file system space. The `.ddb` files are larger than their `.png` equivalent files. An additional option of RLE compression is provided on the Fujitsu Jade target (using a `_ddb_rle_files.gen` at code generation time) to help reduce the file system size requirements with minimal impact to `.ddb` file load time. For a `_ddb_rle_files.gen`, only the `.ddb` files are RLE compressed. Any static images (from Raster or Stencil objects) or font characters compiled into the generated code are still in an uncompressed DDB format.

For software render targets (such as Intel x86 Software Render Windows/Linux), the **Image object use files** option and a `ddb.gen` or `ddb_files.gen` are not compatible for any Image object that is transformed or distorted. This will display an error message at DeepScreen code execution time when it is time to render the Image object. Also, a transformed or distorted Snapshot object is not compatible with a `ddb.gen` or `ddb_files.gen` for software render targets. This will also display an error message at execution time when the Snapshot needs to be rendered.

16.16 DeepScreen code generator improved image processing (as of 10.0)

DeepScreen has been updated with a new External Image Processing engine. This engine processes all image data including rasters, image objects, stencils, patterns, and fonts.

A new dialog is available during code generation showing the details of image processing. This dialog is shown by enabling the **Show Image Formatting Dialog** option in the Code Generation Options dialog. The dialog shows the total image processing progress as well as details for every image in the design file.

NOTE: The new Image Formatting Engine will enforce error checking of the `.dsn` file during code generation. Code Generation will stop if invalid image references are in the `.dsn` file (i.e. an Image object with an invalid image file path). This prevents the unintentional generation of code with missing images. To disable treating missing images as an error, enable the **Treat missing images as warnings** option in the Code Generation Options dialog.

16.17 DeepScreen 10.1 and Earlier Miscellaneous Improvements

Altia Design 10.1 and 10.0 Miscellaneous DeepScreen Improvements:

- Please see the later section *Enhancements and Fixes by Release and Incident Number* and search on DeepScreen for details.

Altia Design 9.2 Miscellaneous DeepScreen Improvements

- There is a new code generation option **Generate external resource files** to generate Raster object pixel values, font character data, and Stencil object bits to binary files instead of source code files.
- The handling of Image objects in DeepScreen generated code is enhanced for designs containing Image objects that have identical image_name animations and the **Image object use files** code generation option is chosen. At DeepScreen code execution time when Image objects are initializing, the image file name held by the image_name animation is only read one time (by the first Image object with the given image_name that gets initialized). Other Image objects with the same image_name animation will not read the file again when they initialize. Instead, they will detect that the internal raster structure is already referencing valid image data and there is no reason to read the file. This enhancement improves initialization time and uses less heap memory (just one copy of the raster data instead of a copy for each Image object that has an identical image_name animation). This enhancement only improves performance at initialization time. If a shared image_name animation is given a new image file name (such as from application code via the Altia API or from Altia Control code), each instance of the

Image object with that same image_name animation opens the image file and loads the image. Each Image object instance will load the image data into the same internal raster structure so the final amount of memory in use will be for just one copy of the image data, but each Image object instance consumes CPU and I/O time/resources to open the image file and load the image data.

- DeepScreen accuracy of string width calculations is improved for altiaGL targets, framebuffer targets (e.g., Intel x86 Software Render Linux), and accelerated targets (e.g., Altera D/AVE, Fujitsu Jade, etc.). This improves string placement especially for the Text I/O sibling justify modes.
- Text I/O max_pixel_count animation initial value was being ignored in DeepScreen generated code. This is fixed.
- DeepScreen Text I/O was drifting with right or center sibling justify modes if no sibling actually existed. This is not a valid configuration, but it also was not the behavior prior to the 8.08 and newer releases. This is fixed.
- DeepScreen static label characters with 128-255 character values (e.g., degrees, western euro characters, etc.) were previously not displaying because the values were getting sign extended at code gen time. This is fixed.
- DeepScreen Snapshot object support is expanded to include altiaGL targets, Altera D/AVE target, Fujitsu Jade target, Intel x86 Software Render Linux target, and OpenGL ES 2.0 targets.
- The DeepScreen Skin object image path animation is now processed even if there is no use of a file system. There is a use case where it is desired to generate skin files into altia/skinData.c, but then change the image path. Essentially this is changing skins by using the same skin data, just different image folders. This is a very isolated change to allow changing of the image path when there is embedded skin data (i.e., whether or not there is a skin file providing the data).
- The "DDB" uncompressed image data code generation mode is enhanced to support a chroma key identifier. The previous algorithm converted transparency masks to alpha channels. For targets like Fujitsu Jade, this prevents rotation/scaling of the image and using an alpha channel is less efficient than using a chroma key color if the hardware supports it. See one of the Fujitsu Jade target's DDB .gen files for an example usage.
- The DeepScreen Drawing Area object (DAO) redraw region was off by one. This is fixed.
- Code gen for an empty Deck could cause another object's animation to get the Deck's card events if any card events were generated from code/control/stimulus. This is fixed.
- DeepScreen code changes were made to give application code a chance to detect raw stimulus events before they execute animations. Application code can assign a function's address to the global function ptr: void (*AltiaReportEventPtr)(ALTIA_UBYTE eventType). The eventType argument passed to the function will be a value from one of the altiaTypes.h file's Altia_InputEvent_types defines (for example, ALTIA_DOWN_EVENT_TYPE).

Altia Design 9.1 and 8.08 MR3 Miscellaneous DeepScreen Improvements (missing in 9.0)

- Added support for upper ANSI character range (character values > 127) in optional FreeType font engine for Altera D/AVE, Fujitsu Jade, OpenGL ES, and Toshiba targets. The FreeType software is copyright © 2006-2008 The FreeType Project (www.freetype.org). All rights reserved. This notice satisfies The FreeType Project LICENSE agreement requirement to acknowledge The FreeType Project when its software is included in a commercial product.
- Previously, Image objects were failing to load from files when NO_PRELOAD was also defined. This is fixed.

Altia Design 9.0 Miscellaneous DeepScreen Improvements

- When code is generated for a distorted object, the DeepScreen transform data structures and code include information and logic to perform non-affine transformations.
- DeepScreen accelerated targets, altiaGL targets, and the X11 target might draw black pixels on the edges of rotated images (e.g., Raster objects in Altia). There are changes for the 9.0 release to draw these pixels as the background color instead of black.

- For DeepScreen generated code compiling with Visual Studio 2005 or 2008, the compile options are changed to use -O1 for improved performance.
- For some DeepScreen accelerated targets (for example, Altera D/AVE), a distort animation may not reach its maximum size. This is not a defect. Targets that use texture mapping to do distort cannot distort to the same size for large distortion factors compared to targets that distort in software (such as altiaGL).

Altia Design 8.08 MR3 Miscellaneous DeepScreen Improvements

- Previously, an image (e.g., Raster object in Altia) with alpha channels would have too much transparency on its edges as it stretched/scaled. This is fixed.
- Previously, Image objects might not draw correctly if they were transformed (scaled/rotated) and images were dynamically loaded from a file (by using the "Image Object use files" option at code generation time). This is fixed.
- Previously, animating to full opacity could change another object's opacity. This is fixed.

Altia Design 8.08 MR2 Miscellaneous DeepScreen Improvements

- The Text I/O max_pixel_count animation is supported in DeepScreen. For details on this new Text I/O object animation, please see the ***Altia Design Enhancements Summary***.
- DeepScreen altiaGL custom targets can choose two background colors for raster and text objects to support special behaviors at the driver level. Please see the Altia Design software installation file usercode/altiaGL/README.txt for more details. Search for the key word ALTIA_USE_BACKGROUND_COLOR_FLAG. Also, changes were made so that ALTIA_USE_BACKGROUND_COLOR_FLAG works with transformed Raster/Image objects (i.e., rotated/scaled/stretched color images) and Stencil objects (i.e., monochrome bitmaps). This is in addition to the existing support for untransformed Raster/Image objects.
- There is a nodither option for the .gen file %IMAGEFLAG% directive when generating images as uncompressed DDB data. The nodither option disables dithering of the uncompressed DDB data. Dithering is enabled by default to smooth the appearance of the image, but it can result in bands of colors for some color depths. For details, open a .gen file that uses the %IMAGEFLAG% (usercode/fujitsu/jade_qnx_int_ddb.gen for example).
- There is a nopat option for the .gen file %IMAGEFLAG% directive. If it is used, it keeps patterns (used to fill solid vector objects) in a monochrome format instead of an uncompressed DDB data format. For details, open a .gen file that uses the %IMAGEFLAG% (usercode/fujitsu/jade_qnx_int_ddb.gen for example).
- When generating code with the "Image Object use files" option enabled, if NO_PRELOAD is defined at compile time and an Image objects is initially hidden at execution time, it's image data is not loaded from the file system until the Image object shows (such as because of a card change for the Deck containing the Image object). This shortens initialization time of the DeepScreen generated code.
- A text shifting issue is resolved for the Text I/O object. Previously in Altia Design, Altia Runtime, and DeepScreen generated code, a Text I/O object using a sibling justify mode of 5 or higher (for right or center sibling justify) might show a slight shift in position as characters change if the leading character has a positive offset change. This is fixed. There is still the chance of a shift for sibling justify mode 4 (left sibling justify). Use justify mode of 0 as an alternative.
- There is a change for altiaGL targets to resolve a bad memory reference. There was a corner case for a bad memory reference because GetScratchGC() of altiaGL/egl_common.c did not initialize pGC->stipple to NULL. The bad memory reference would only occur if the previous pGC->stipple element did not exist anymore.
- There is an OpenGL (3D) object change for updating the camera location. Previously, the OpenGL (3D) object jumped when camera location data was changed quickly. The solution was an OpenGL (3D) object code change to only update camera x and y when camera z changes. This change requires existing designs to change so that they update the camera z whenever camera x or camera y are changed. Necessary changes are made to the

demos/openglobj/viewer.dsn design (for details about this design, open demos/openglobj/OpenGLObj.pdf or the README.txt file in the same folder).
- There is a change to support code generation without stimulus when the design contains an OpenGL (3D) object. Previously, generated code for an OpenGL (3D) object would not compile if "Generate code for stimulus" was disabled at code generation time.
- Accelerated targets have a change in the processing of Control WHEN blocks for the "altiaInitDesign" event. Previously for Altera D/AVE, Fujitsu Jade/Carmine, and Toshiba CapA/CapM targets, the driver could fail on the first screen draw if Control WHEN blocks for the "altiaInitDesign" event caused off-screen animations to change states. This is fixed.
- There is a change to the processing of timer stimulus events. Previously, timer stimulus events might execute twice on each interval if the event triggered Control WHEN blocks. This is fixed.

Altia Design 8.07 MR3 Miscellaneous DeepScreen Improvements

- If generating code to use dynamic memory and there are Control WHEN blocks, the Control WHEN queue is now gracefully resized to accommodate more queued WHEN blocks if necessary. Previously the queue could overflow if there were too many simultaneous WHEN block execution requests and this could cause an infinite loop.

Altia Design 8.07 MR2 Miscellaneous DeepScreen Improvements

- The behavior for NO_PRELOAD with PRELOAD_CNT is improved and a PRELOAD_STENCIL_CNT is added. Defining NO_PRELOAD at compile time suppresses initial loading of all Raster and Stencil device dependent bitmaps (DDBs). Also defining PRELOAD_CNT to a value limits the number of simultaneously loaded Raster DDBs to the chosen value (if PRELOAD_CNT is not defined, it defaults to 1 when NO_PRELOAD is defined). Also defining PRELOAD_STENCIL_CNT to a value limits the number of simultaneously loaded Stencil DDBs to the chosen value (if PRELOAD_STENCIL_CNT is not defined, it defaults to 1 when NO_PRELOAD is defined). Set the defines with compile options like: –DNO_PRELOAD –DPRELOAD_CNT=100 –DPRELOAD_STENCIL_CNT=100. The improved behavior is to reduce memory usage by reusing loaded DDBs for Rasters referring to the same image data and Stencils referring to the same monochrome image data.
- The behavior for YES_COMPATIBLE_LAYERS is improved. If a target uses layers, defining YES_COMPATIBLE_LAYERS at compile time indicates that the color formats of all layers are identical. When a Raster is loaded, the logic to determine a Raster's layer number is skipped. This improves performance. In addition, duplicate Rasters on different layers share device dependent bitmaps (DDBs). This reduces memory usage. YES_COMPATIBLE_LAYERS in combination with NO_PRELOAD (w/wo PRELOAD_CNT or PRELOAD_STENCIL_CNT) can significantly improve initialization speed and reduce memory usage.

Altia Design 8.07 MR1 Miscellaneous DeepScreen Improvements

- Changes were made to the Clip object rendering code to minimize extent off-by-one issues (compared to Clip object extents in Altia Design and Runtime) especially for fixed point targets.
- The new Text I/O **justify_mode** of **7** is supported in DeepScreen. It right justifies to a sibling object just like the existing justify mode of 5, but it clips text on the left side instead of the right side if **clip_on** 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).
- The built-in Altia animation **altiaSetOpacity** is supported in DeepScreen generated code if the **Generate code for Built-in Functions** option is enabled at code generation time. This built-in animation was already supported in Altia Design and Runtime. For more about this built-in animation, please see the *Altia Design 9 User's Guide, Chapter 7, Section 7.8.9 – Miscellaneous Functions*.
- The Altia Design models library decks.dsn is changed. Previously, its components contained Control code that might not compile for deeply embedded targets or if it compiled, it might generate unresolved external symbols at link time. The decks.dsn models library in this

release is simplified so that it's DeepScreen generated code will compile for any target. Press the **Libraries** button in Altia Design to show an Open dialog for selecting a models library to open.

- Now a Deck object does not redraw when its card animation state is set and the value is the same as the previous value. This is a performance optimization.
- Now the altiaGL targets gracefully ignore rendering characters for which there is no character bitmap. Previously, they would abort the rendering of the current character string and this would cause the initial valid characters to show in the wrong position. For 8.07 MR3 or newer, the accelerated targets (Altera D/AVE, Fujitsu Jade and Carmine, OpenGL ES, and Toshiba Capricorn) also gracefully ignore rendering characters for which there is no character bitmap.
- There are changes to the processing of Control $format statements. Previously the %f in Control $format statements would have extra 0 decimal digits in the result compared to Altia Design and Runtime. This is fixed so that DeepScreen code displays the number just like Altia Design and Runtime.
- If a Text I/O object is not showing any text (for example, because the clear animation was previously set), it now ignores any repeated clear animation events. This is different from the behavior by Altia Design and Runtime. It is an optimization to improve performance on embedded targets when it is not convenient for application code to minimize setting of the clear animation.
- There is a change to how a Text I/O processes a string that is a number. If a Text I/O object receives a string that is a number, it generates the number as an event on its float and integer animations. In DeepScreen generated code prior to 8.07 MR1, the integer number was not rounded like it is in Altia Design and Runtime. This is fixed.
- Previously, setting the hilight_color animation for a DeepScreen Text I/O or MLTO using a hex number string in the style "#RRGGBB" would show the wrong color. This is fixed.
- Code generated to execute on Windows XP/Vista now compiles with Visual Studio 2008. Visual Studio 2005, eMbedded Visual C 4.0, and Visual C 6.0 are also supported. See the Intel x86 Software Render target User's Guide for details. Installation requirements given for Visual Studio 2005 generally also apply to Visual Studio 2008.

Altia Design 8.06 MR9 Miscellaneous DeepScreen Improvements

- The new built-in Altia animations **altiaGetObjWH**, **altiaObjW**, and **altiaObjH** are implemented in DeepScreen generated code (excluding the Java and miniGL targets) if the **Generate code for Built-in Functions** option is enabled at code generation time. These built-in animations allow for getting the width and height, in pixels, for an object given the object's identification number. Please see the early ***Altia Design Enhancements Summary*** for more details about these built-in animations.
- Previously, line widths for the DeepScreen **altiaGL** and **X11** targets could not exceed 16 pixels. Now the limit is 100 pixels. Line widths for other targets do not have an explicit limit. Instead, it depends on the limitations of the hardware or underlying graphics library.
- Now there is code generation for dynamic multi-line (Dyna Line in Altia Design) and dynamic filled or unfilled polygon (Dyna Polygon in Altia Design) objects (excluding the Java and miniGL targets). These objects are typically copied from the Altia Design software installation more/plot.dsn models library. Press the **Libraries** button in Altia Design to show an Open dialog for selecting a models library to open. DeepScreen code generation now supports all objects in more/plot.dsn (contrary to what is written in the models library itself).

Altia Design 8.06 MR4 Miscellaneous DeepScreen Improvements

- DeepScreen altiaGL, Altera D/AVE, Fujitsu, and Toshiba targets support layers. Please see the later section *DeepScreen Layers* of this document for more information.

16.18 DeepScreen Notes and Known Issues for Altia Design 10.1

- Incident #0094: Fujitsu Jade DeepScreen Target - Clip boundaries occasionally off by 1 pixel.

- Incident #0097: Fujitsu Jade DeepScreen Target - Clip boundaries occasionally off by 1 pixel.

- Incident #0172: Fujitsu Jade DeepScreen Target - Raster image blending off by 1 pixel.

- Incident #0264: Windows and Windows CE DeepScreen Targets - These targets have been removed. Replacement for the Windows target is the Intel x86 Software Render Windows target based on the altiaGL windib target. This replacement target does not support the multiple views or cloning features. If one of these features or a Windows CE target is still required, continue to use your existing Altia Design release that supports these features and/or the required Windows CE target.

- Incident #0396: altiaGL DeepScreen Target - Additional target files list directxfb_*.gen are non-functional with Microsoft SDK June 2010.

- Incident #0408: altiaGL DeepScreen Target - Button rotation off 1 pixel.

- Incident #0414: altiaGL DeepScreen Target - Execution of design containing snapshot displays dialog "char 0x20 missing from font."

- Incident #0415: altiaGL DeepScreen Target - Execution of design containing snapshot results in artifacts and other rendering issues.

- Incident #0416: altiaGL DeepScreen Target - Scaled snapshot off by 1 pixel.

- Incident #0417: Altera D/AVE DeepScreen Target - Clip Object off by 1 pixel.

- Incident #0457: Altera D/AVE DeepScreen Target - Model, in 3 slice pie chart, "C" is distorted.

- Incident #0458: Altera D/AVE DeepScreen Target - Click and hold up button for number input, results in display artifact.

- Incident #0459: Altera D/AVE DeepScreen Target - Text spacing incorrect especially on rotated text.

- Incident #0523: miniGL DAO DeepScreen Target - Rasters are shifted 1-2 pixels.

- Incident #0525: miniGL DeepScreen Target - A design with an empty deck can display a confusing message, "Object # 34 is not a child. Must be in a group or deck object."

- Incident #0609: Fujitsu Jade DeepScreen Target - Semi-transparent texture rendering issue.

- Incident #0629: All DeepScreen Targets - Snapshot operations do not support DDB.

- Incident #0635: All DeepScreen Targets – Text I/O objects which have an animation parameter "textio_max_pixel_count" configured will observe that the text is cut off prior to reaching the max pixel count. It is off by approximately 10 pixels.

- Incident #0790: Altera Nios II D/AVE Target – For the new alpha<#> .gen files for 1bpp, 2bpp, and 4bpp alpha masks on stencils and font characters (DDB mode only), distorted/rotated/scaled text might be filled with the wrong color, such as always white.

- Incident #0833: Renesas SH7266 DeepScreen Target – Rendering spline objects is not supported.

- Incident #0834: Renesas SH7266 DeepScreen Target – Extended character values [0x80-0xFF] in text (a Label, Text I/O, or MLTO) at code generation time will not compile, but they can be sent to Text I/O or MLTO objects at execution time.

- Incident #0835: Renesas SH7266 DeepScreen Target – Generated code does not compile if the design contains a Skin object.

- Incident #0836: Renesas SH7266 DeepScreen Target – Unicode characters in text are not supported.

- Incident #0884: Vector Path Object for DeepScreen Target – The Vector Path object is only support on DeepScreen OpenVG targets.

- Incident #0886: Toshiba Capricorn DeepScreen Target – This target is not available in this release, but will return in a future release.
- Incident #0896: Renesas V850E2/DR4-3D OpenGL ES 1.1 DeepScreen Target - The Distort feature is not supported.
- Incident #0897: Freescale i.MX51 OpenVG 1.1 DeepScreen Target – The Snapshot object is not supported.

17 DeepScreen Enhancements and Fixes by Release and Incident Number for Altia Design 10.1 and Earlier Releases

The enhancements and fixes summarized here are first ordered by release starting with the Altia Design 10.1) and continuing back to the September 2008 Altia Design 8.06 MR9 release (which followed the April 2007 Altia Design 8.062 release). Within a release, DeepScreen enhancements and fixes are ordered by incident number which is approximately the chronological order that enhancements were requested and issues to be fixed were reported.

17.1 Altia Design 10.1 Enhancements and Fixes

- Incident #0264: DeepScreen Windows – Enhancement, added Intel x86 Software Render Windows target to replace the Windows target in previous releases. The Intel x86 Software Render Windows target is based on the altiaGL windib target. This replacement target does not support the multiple views or cloning features. If one of these features or a Windows CE target is still required, continue to use your existing Altia Design release that supports these features and/or the required Windows CE target.

- Incident #0516: DeepScreen OpenVG i.MX51 - Rectangle boundary off by one pixel - more_all_objects. This was fixed for the 10.1 release.

- Incident #0517: DeepScreen OpenVG i.MX51 - Objects have wrong y position - more_all_objects. This was fixed for the 10.1 release.

- Incident #0611: DeepScreen – Enhancement, added Fujitsu MB91590 (Sapphire) Custom 2D target.

- Incident #0628: DeepScreen miniGL - Control code execution order differs from Altia Design. This was fixed for the 10.1 release.

- Incident #0630: DeepScreen – Enhancement, added Renesas V850E2/DR4-3D OpenGL ES 1.1 target.

- Incident #0633: DeepScreen – Enhancement, when text characters (e.g., for Labels, Text I/Os, or MLTOs) are generated, they are now ordered exactly by their character code values. This allows for a very fast binary search at execution time. In addition, a missing character code is detected just as quickly as a valid character code. As part of this improvement, code generated from the Altia Design 8-bit character editor (i.e., not Unicode) must be compiled for 8-bit character encoding (i.e., the UNICODE macros cannot be defined). And code generated from the Altia Design Unicode editor must be compiled for Unicode character encoding (i.e., the UNICODE macros must be defined). If generated code from the Altia Design 8-bit character editor is compiled for Unicode, the compile will abort with a message: UNICODE compile time definition not allowed because this code was generate using the ASCII Altia Editor. If generated code from the Altia Design Unicode editor is compiled for 8-bit character encoding (i.e., UNICODE macros are not defined), the compile will abort with a message: UNICODE compile time definition required because this code was generated using the Unicode Altia Editor.

- Incident #0643: DeepScreen Freescale i.MX51 – Enhancement, each i.MX51 target's template Makefile scripts (for example, altmake_linux_imx51.bat) are configured for the L2.6.35_11.10.01

SDK. The template Makefile scripts also contain comments to reconfigure them for the older L2.6.31_10.04.01 SDK.

- Incident #0649: DeepScreen Toshiba Capricorn - Getting a divide by 0 when rendering an image or a text character just 1 pixel wide or tall. This was fixed for the 10.1 release.

- Incident #0653: DeepScreen OpenGL ES 2.0 - Renders 24-bit rasters incorrectly. This was fixed for the 10.1 release.

- Incident #0663: DeepScreen – Enhancement, added Freescale i.MX53 OpenGL ES 2.0 target.

- Incident #0666: DeepScreen Renesas V850E2/Dx4-H - RLE data should be cached. This was fixed for the 10.1 release.

- Incident #0668: DeepScreen – Enhancement, added Freescale i.MX51 OpenGL ES 1.1 target, Freescale i.MX51 OpenGL ES 2.0 target, and Freescale i.MX51 OpenVG 1.1 target.

- Incident #0670: DeepScreen Renesas V850E2/Dx4-H - Anti-aliased fonts anomalies. These were fixed for the 10.1 release.

- Incident #0676: DeepScreen Altera D/AVE – Enhancement, There are new .gen files ending with alpha<#> that will generate font and monochrome image (Stencil object) data using the specified alpha channel depth. Example: cyclone3_int_ddb_alpha4.gen will generate uncompressed, device dependent Stencil object and font data at 4 bits-per-pixel in alpha depth. Supported depths are 1, 2, and 4 bits per pixel. The default is 8 bits per pixel for the .gen files without an alpha<#> suffix. Please note: for 1, 2, and 4 bit per pixel settings, distorted/rotated/scaled text might be filled with the wrong color, such as always white (open incident #0790).

- Incident #0677: DeepScreen Renesas V850E2/Dx4-H - Extra lines at 90 and 180 degrees on knobs, buttons and stencils, more_all_objects_dx4_1.dsn build 4.6.1 with patch. This was fixed for the 10.1 release.

- Incident #0678: DeepScreen Renesas V850E2/Dx4-H - Anti-aliasing is off for some numbers and characters - build 4.6.1(patched) more_circle_dx4_1.dsn. This was fixed for the 10.1 release.

- Incident #0681: Fujitsu Sapphire: Discrepancy with fujitsuSapphire_packager .bat script. This was fixed for the 10.1 release.

- Incident #0686: DeepScreen Renesas V850E2/Dx4-H - Displays Palace Script font as black rectangles - fonts.dsn build 4.6.1 (patched). This was fixed for the 10.1 release.

- Incident #0687: DeepScreen Renesas V850E2/Dx4-H - Simple toggle button does not display correctly in "On" position for float_layers_32RLE.gen Build: 4.6.1(patched). This was fixed for the 10.1 release.

- Incident #0688: DeepScreen Renesas V850E2/DR4-3D – Enhancement, perform dirty region redraw instead of full screen redraw.

- Incident #0690: DeepScreen – Enhancement, added Renesas SH7266 Software Render target.

- Incident #0697: DeepScreen Renesas V850E2/Dx4-H - Scaled raster extent off by 1 pixel, Build 4.6.1(patched), raster_extent. This was fixed for the 10.1 release.

- Incident #0715: DeepScreen miniGL – Enhancement, include altiaPollEvent() API function (especially to poll animations used in Control code).

- Incident #0719: DeepScreen – Enhancement, added Renesas V850E2/Dx4-H Custom2D target.

- Incident #0726: DeepScreen miniGL – Enhancement, add typecasts to miniGL bits.c to support HC12.

- Incident #0731: DeepScreen Renesas V850E2/Dx4-H - Lines can be positioned off by a pixel due to rounding. This was fixed for the 10.1 release.

- Incident #0732: DeepScreen Freescale Spectrum – Enhancement, the Freescale Spectrum target can generate code that runs on the Freescale MPC5645S (Rainbow) device. At code generation time, use either the rainbow_internal.gen or rainbow.gen file in the Additional Target files list: of the Code Generation Options dialog. In addition, MPC5645S must be defined at compile time to conditionally compile the code for Rainbow versus Spectrum.

- Incident #0742: DeepScreen Renesas V850E2/Dx4-H – Enhancement, provide interface to control dithering (MVO registers). See target generic driver files.

- Incident #0751, #0887: DeepScreen Fujitsu Jade - egl_font.c compile errors for runtime FreeType because position of several variable declarations are illegal for C. This was fixed for Altia Design 10.1.

- Incident #0758: DeepScreen Renesas V850E2/Dx4-H - Error in extent copy. This was fixed for Altia Design 10.1.

- Incident #0757: DeepScreen X11 - "Unable to load Image Formatting Engine" code generation error. This was fixed for Altia Design 10.1.

- Incident #0766: DeepScreen Renesas SH7266 - Enhancement, to make the most of this device's limited memory, the DeepScreen target code implements a statically defined heap from which temporary memory can be allocated at execution time.

- Incident #0772: DeepScreen Freescale Spectrum - Was not processing transparency masks on images so an image with a transparency mask was not appearing at all. This was fixed for Altia Design 10.1.

- Incident #0777: DeepScreen Freescale Spectrum - Characters could wobble if the same text string was drawn on an even pixel boundary and then an odd pixel boundary and the text rendering algorithm was not considering character offsets which could result in larger than desired gaps in the spacing between characters. This was fixed for Altia Design 10.1.

- Incident #0778: DeepScreen Freescale Spectrum - Character spacing was not protecting for a negative offset on first character and it was appearing beyond the left edge of the "fragment." This was fixed for Altia Design 10.1.

- Incident #0799, #0866, #0867: DeepScreen Freescale i.MX51 and i.MX53 OpenGL ES 2.0 - Images can be distorted and shifted left because of inconsistent row alignment (between GL_UNPACK_ALIGNMENT values of 1 and 4 so just always use 1). This was fixed for Altia Design 10.1.

- Incident #0818: DeepScreen - Binary search fix for missing character on first generated character set. This was fixed for Altia Design 10.1.

- Incident #0828: DeepScreen Freescale i.MX51 and i.MX53 OpenGL ES 2.0 - Distort compile error fixed :expected specifier-qualifier-list before VOID. This was fixed for Altia Design 10.1.

- Incident #0831: DeepScreen Freescale i.MX51 and i.MX53 OpenGL ES 2.0 - Generated code does not compile when there is a Snapshot object, egl_common.c reference to altiaLibInSnapshotDraw is undefined. This was fixed for Altia Design 10.1.

- Incident #0848: DeepScreen Renesas V850E2/DR4-3D - Unfilled rectangles were being drawn out of position. This was fixed for Altia Design 10.1.

- Incident #0849: DeepScreen Renesas V850E2/DR4-3D - Dirty region extents were not getting cleared before each redraw. This was fixed for Altia Design 10.1.

- Incident #0857: DeepScreen Renesas V850E2/DR4-3D - High speed serial flash address ranges were wrong in documentation. The target documentation has been updated to show the correct address ranges. This was fixed for Altia Design 10.1.

- Incident #0862: DeepScreen Freescale i.MX51 targets - Enhancement, all target board set-up instructions converted to PDF files.

17.2 Altia Design 10.0 Enhancements and Fixes

- Incident #0074: DeepScreen - Enhancement, add Layer Synchronization

- Incident #0075: DeepScreen - Enhancement, add Texture Processing for targets that follow texture rules when drawing

- Incident #0088: DeepScreen - Enhancement, support setting a default character rather than skipping missing encoded font characters for Altera D/AVE

- Incident #0096: DeepScreen miniGL & Spectrum, resolve packing error when language objects use UNICODE characters

- Incident #0105: DeepScreen miniGL, resolve H8SX issue with altiaSelectEvent() - application is not notified of event

- Incident #0106: DeepScreen miniGL, resolve clipping cache problem with repetitive altiaSendEvent calls

- Incident #0217: DeepScreen Toshiba Capricorn A target - resolved compile failure

- Incident #0248: DeepScreen target - Addition of the Renesas SH7268 OpenVG target

- Incident #0507: DeepScreen miniGL target - modify lcdwin32 simulator such that the simulator will remain on top

- Incident #0512: DeepScreen miniGL target - Add control code support

- Incident #0513: DeepScreen miniGL target - Add timer stimulus support

17.3 Altia Design 9.2 Enhancements and Fixes

- Incident #738: If DeepScreen code is generated just for DAO, it will not compile. This was fixed for the 9.2 release.

- Incident #842: DeepScreen Windows target had droppings for transformed rasters with alpha pixels. This was fixed for the 9.2 release.

- Incident #902: For a DeepScreen altiaGL framebuffer target (e.g., directxfb or linuxfb, not windib because it is a scanline target) that has segmented memory (such as Intel x86 architectures), transformed rasters can cause a fault because a memory address just beyond the raster's area might be read and this address might not be in the current segment. This was fixed for the 9.2 release.

- Incident #941: DeepScreen Fujitsu Jade Snapshot object support is added in the 9.2 release.

- Incident #965: DeepScreen's single char draw code was not taking MLTO height into account and this code is used by OpenGLES even if the text is not transformed so MLTO was drawing in wrong position. This was fixed for the 9.2 release.

- Incident #1074: In DeepScreen miniGL, if a Text I/O object changes, but it is hidden (such as in a card of a Deck that is not showing), now the area of the Text I/O is not redrawn. This was an enhancement for the 9.2 release.

- Incident #1106: DeepScreen DAO redraw region off by one. This was fixed for the 9.2 release.

- Incident #1125: DeepScreen code might crash if no code gen for Image objects to use files, but Control or application code still sets image name animations. This was fixed for the 9.2 release.

- Incident #1141: DeepScreen memory growth when Image object loads a new image after it has been transformed. This was fixed for the 9.2 release.

- Incident #1142: DeepScreen DAO text not refreshing correctly in deepscreen/exapi/plotdraw.dsn after draw button press. Also when window becomes visible after dragging another window over it. This at least happens with altiaGL windib target. This was fixed for the 9.2 release.

- Incident #1143: For DeepScreen, fixed DAO recursive calling issue into exapi when an exapi func calls TargetAltiaUpdate. This was fixed for the 9.2 release.

- Incident #1155: DeepScreen compile error for altiaGL DDB when there is a Snapshot object. This was fixed in the 9.2 release and a Snapshot object is OK as long as there is no transforming of the Snapshot.

- Incident #1156: DeepScreen snapshotting of Tickmark was putting snapshotted Tickmark in the wrong place. Also crash is fixed when snapshotting scaled/rotated text/bitmap with Scale Bitmaps/Text disabled. And also objects are not incorrectly semi-transparent for Windows unicode gen/make. This was fixed for the 9.2 release.

- Incident #1159: DeepScreen API Server code can cause DeepScreen executable to crash on exit if code is generated with "Generate code for clones" option. This was fixed for the 9.2 release.

- Incident #1175: DeepScreen Linux Unicode compiles fail because of missing wfopen() function. This was fixed for the 9.2 release.

- Incident #1173: DeepScreen Snapshot object support is expanded in the 9.2 release to include the Windows target, altiaGL targets, Altera D/AVE target, Fujitsu Jade target, Linux Framebuffer target, OpenGL ES 2.0 target. Restrictions are (1) Capture by "Location" (capture mode 2) is not yet implemented in DeepScreen, (2) if a "DDB" .gen file is chosen at DeepScreen code generation time, the Snapshot object will not render if it is transformed, (3) Windows, altiaGL, and Linux Framebuffer targets only support the "Software" alpha mode (alphaMode animation = 2).

- Incident #1197: DeepScreen Fujitsu Jade target rejects compiling generated code for the "Full Screen Mode" option in the 9.2 release. Fujitsu Jade always compiles code for full screen rendering and selecting this option is not compatible.

- Incident #1200: DeepScreen Fujitsu Jade generated code was modified to resolve Fujitsu Jade Errata V1.4 issue #E5 so that missing pixels do not appear in polygons. This was changed for the 9.2 release.

- Incident #1202: DeepScreen Jade FreeType documentation fixes were made for the 9.2 release.

- Incident #1216: Updating more Windows Makefile scripts to save/restore LIBPATH for using autogen testing framework with VS 2008. At the same time, added comments about setting -DDEBUG_WINDOW to show a command prompt window for stdout/stderr messages. These enhancements were made for the 9.2 release.

- Incident #1229: DeepScreen Snapshot memory leak when capturing and transforming. This was fixed for the 9.2 release.

- Incident #1230: DeepScreen Linux Framebuffer target is not drawing stencils rotated 90 degrees counter-clockwise (CCW). This was fixed for the 9.2 release.

- Incident #1237: DeepScreen snapshot/transition.dsn snap4 fails for Windows with Unicode .bat script. This was fixed for the 9.2 release.

- Incident #1240: For DeepScreen, added raster transform check for mixed hw/sw draws. Consolidation of the egl_AlphaChannelSet() was part of the fix for the 9.2 release.

- Incident #1242: DeepScreen altiaGL directxfb target in windowed mode (not full screen) is not updating the graphics on JM computer. The framebuffer pointer changes after the surface is unlocked. This was fixed for the 9.2 release.

- Incident #1246: The DeepScreen Skin object image path animation is now processed even if there is no use of a file system. There is a use case where it's desired to generate skin files into skinData.c, but then change the image path. Essentially this is changing skins by using the same skin data, just different image folders. This is a very isolated change to allow changing of the image path when there is embedded skin data (i.e., whether or not there is a skin file providing the data). This was an enhancement for the 9.2 release.

- Incident #1249: DeepScreen Fujitsu Jade target now supports the Snapshot object. This was an enhancement for the 9.2 release.

- Incident #1250: OpenGL ES 2.0 target support for Snapshot object (support for OpenGL ES 1.1 is not possible). This was an enhancement for the 9.2 release.

- Incident #1261: DeepScreen TMPA900 16 bpp framebuffer fix. The issue had to do with stride calculation for 16-bit pixmaps in the hardware blit routine. This was fixed for the 9.2 release.

- Incident #1262: DeepScreen TMPA900 invalid free() operation fix. The TMPA900 target uses an internal memory manager. The xrealloc function was calling free() instead of xfree(). This was fixed for the 9.2 release.

- Incident #1266: DeepScreen Snapshot support for OpenGL object and Drawing Area object (DAO). For DAO support, application code must use the AltiaEx API to draw to the DAO. This enhancement was added for the 9.2 release.

- Incident #1267: Use the new "Generate external resource files" option at code generation time to generate all image and font data to binary files instead of the data.c and egl_font.c source code files. Raster object pixel data is generated to <DESIGN_NAME>_pixels.bin, all font character images to <DESIGN_NAME>_fonts.bin, and all Stencil object and mask data to the <DESIGN_NAME>_stencils.bin file. At execution time, the default behavior of the generated code is to open each file with fopen(), allocate a block of memory large enough to hold the data from the file (the data size was determined at code generation time and included in the generated data.c file), read the data from the file into an allocated block of memory using a single call to fread(), and close the file with fclose(). The data files contain binary data exactly in the format the data would have taken if it had been generated to the data.c and egl_font.c files. For example, if a DDB .gen file is chosen at code generation time, the data is in the defined device dependent format. Otherwise, the data is in Altia's custom compressed format. Using this external resource files feature generates a data.c file that is much smaller (it compiles faster and takes up much less program space on the target). The use of fopen(), fread(), and fclose() requires a target environment that supports these functions. The generated .bin files must be in the working directory at execution time. In other words, only a file name (such as "my_design_pixels.bin" without any path information) gets passed to fopen(). The file processing can be fully customized by defining ALTIA_DRV_CUSTOMASSET at compile time (as in a -DALTIA_DRV_CUSTOMASSET option to the compiler). Open the generated altia/<TARGET>/egl_common.c file and search for "Asset" to see the actual implementation for a specific target. This was an enhancement for the 9.2 release.

- Incident #1267: DeepScreen Fujitsu Jade target now supports the "Generate external resource files" code generation option. This was an enhancement for the 9.2 release.

- Incident #1267: DeepScreen Fujitsu Jade target Linux port now supports the "DDB" uncompressed image data code generation mode. See one of the Fujitsu Jade DDB .gen files for an example usage. This was an enhancement for the 9.2 release.

- Incident #1268: DeepScreen altiaLibFxmath.c compile fails if ALTIA_USE_DRIVER_FX_MATH is defined (support #7176). This was fixed for the 9.2 release.

- Incident #1269: DeepScreen generated code compiler warnings when a design contains a Snapshot object and an OpenGL obj. This was fixed for the 9.2 release.

- Incident #1270: DeepScreen Red HMI hang on Altera D/AVE cube because of wrong optimize level (use -O3, not -O2). This was fixed for the 9.2 release.

- Incident #1273: DeepScreen static label characters with 128-255 character values (e.g., degrees, western euro characters, etc.) were previously not displaying because the values were getting sign extended at code gen time. This is true for miniGL as well as all other targets. This was fixed for the 9.2 release.

- Incident #1277: DeepScreen Toshiba Capricorn image dithering pseudo-noise function was made in-line to improve performance. This was an enhancement for the 9.2 release.

- Incident #1277: DeepScreen Toshiba Capricorn image decompression code now sets transparent pixels to black (and skips a color calculation entirely for improved performance). This was an enhancement for the 9.2 release.

- Incident #1277: DeepScreen Toshiba Capricorn display timings were modified to add support for a 1280x800 PC monitor. This was an enhancement for the 9.2 release.

- Incident #1280: DeepScreen Fujitsu Jade target does not do any dynamic memory allocation for the "DDB" uncompressed image data code generation mode when static memory code generation is chosen (i.e., "Use Dynamic Memory" is not a selected code gen option). This was an enhancement for the 9.2 release.

- Incident #1281: For DeepScreen Toshiba Capricorn, there is now a Green Hills Probe RBTX4961_AOB.mbs target initialization script for the Capricorn-AOB (Application Oriented Board) development board. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL support for Drawing Area object (DAO) with limitations. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL fix was made for "dancing text bug" when Text I/Os have the same base name. This was fixed for the 9.2 release.

- Incident #1283: Layer identifiers are now supported by DeepScreen miniGL. This is defined by the miniGL .ali file .layerids token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL pixel mode support is added (in contrast to address mode). This is required by some custom targets, mainly hardware accelerated solutions. This is enabled by the miniGL .ali file .gcmode token. This was an enhancement for the 9.2 release.

- Incident #1283: For DeepScreen miniGL, added the ability to turn off bit counts and report width or height information. This is required by some custom targets, mainly hardware accelerated solutions. This is defined by the miniGL .ali file .nobitcounts token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL custom encoder plug-in DLL hook added to allow flexibility during the code gen packing process. This is defined by the miniGL .ali file .custom encoder token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL images can now be padded, both width and/or height. This is required by some custom targets, mainly hardware accelerated solutions. This is defined by the miniGL .ali file .pixelpad token. This was an enhancement for the 9.2 release.

- Incident #1283: For DeepScreen miniGL, the opaque stencils optimization can be turned off whereby they will be stored as normal stencil images. This is required by some custom targets that cannot dynamically draw data such as sprite based targets. This is defined by the miniGL .ali file .noopaquestencils token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL checksum value generation can be turned off. Some custom targets cannot accommodate a checksum value due to alignment restrictions. This is defined by the miniGL .ali file .external nochecksums token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL chromakey color definition is added (this is a color to make transparent). Used for padding images and replacing RGB pixel data that has an alpha value of 0 (transparent). This is defined by the miniGL .ali file .chromakey token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL anti-aliased fonts can now be configured up to 8 bits per pixel. Previously only 8-bpp was supported. This is defined by the miniGL .ali file .alphafonts token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL alpha values for fonts and rasters can now be inverted. Some custom targets require this. This is defined by the miniGL .ali file .alphavalues inverted token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL raster alpha values can now be interlaced (inlined) into the RGB data. When interlacing is turned on, the alpha mask (if any) directly follows the RGB data instead of being stored in a separate data block. This is defined by the miniGL .ali file .interlaced token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL static colors are now optimized into a common table to save ROM. Optionally the index value can be preserved and provided to the miniGL driver layer via the graphics context. This is defined by the miniGL .ali file .colors keepindex token. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL indirect color moding support has been added. miniGL automatically determines the best fit for a particular raster image and will encode it accordingly. Optionally, images can be constrained to an indirect color mode. Likewise, color lookup tables can be constrained. A color saver option has also been introduced to reduce the size of encodings. These features are enabled by the various miniGL .ali file .indirect tokens. This was an enhancement for the 9.2 release.

- Incident #1283: For DeepScreen miniGL, a pre-flush function, miniGLDriverFlushing(), has been added. Useful for hardware accelerated targets. This was an enhancement for the 9.2 release.

- Incident #1283: DeepScreen miniGL support for a MINIGL_TEXT_CHARACTER_FLAGS compile option is added for more flexible processing of static text strings. Some custom miniGL targets can require this. Enabling this option identifies first and last characters of a text string. This is necessary when special consideration is required for processing whole strings of text. This was an enhancement for the 9.2 release.

- Incident #1283: For DeepScreen miniGL DAO simulator target, support for defining DAOWIN32_DEBUG_EXTENTS at compile time (as in -DDAOWIN32_DEBUG_EXTENTS) was added to draw a bounding box around the dirty rectangles for internal debugging/development. This was an enhancement for the 9.2 release.

- Incident #1283: For DeepScreen miniGL DAO simulator target, a new event "miniGLUpdateFBDone" is generated after a frame buffer update for external app code to monitor the frame buffer (perhaps to capture it for testing purposes).

- Incident #1284: DeepScreen Fujitsu Jade has improved rendering for distorted images by breaking up the raster surface into more triangles. This was an enhancement for the 9.2 release.

- Incident #1286: DeepScreen miniGL dobjsc array is now pushed into .const data rather than the .data segment (RAM) for a significant improvement in RAM memory usage. This was an enhancement for the 9.2 release.

- Incident #1288: The .align token in the DeepScreen miniGL .ali file is enhanced to support different alignments for raster, stencil, alphamask, monomask, alphafont, and monofont data. Previously, all such resources could only be aligned the same (on 8, 16, or 32 bits). See one of the updated miniGL .ali files for a description of this feature. This was an enhancement for the 9.2 release.

- Incident #1289: In DeepScreen miniGL, special operations are now allowed for dealing with text for custom miniGL targets. The maximum label and Text I/O character width is now made available at the driver level. The value is defined by MAX_TEXTIO_ENCODING_LENGTH in the miniGLDriver.h header file.

- Incident #1291: The handling of Image objects in DeepScreen generated code is enhanced for designs containing Image objects that have identical "image_name" animations and the "Image object use files" code generation option is chosen. At DeepScreen code execution time when Image objects are initializing, the image file name associated with the "image_name" animation will only be read one time (by the first Image object with the given "image_name" that gets initialized). Other Image objects with the same "image_name" animation will not read the file again when they initialize. Instead, they will detect that the internal raster structure is already referencing valid image data and there is no need to read the file again. This enhancement improves initialization time and uses less heap memory (just one copy of the raster data instead of a copy for each Image object that has an identical "image_name" animation). This enhancement only improves performance at initialization time. If a shared "image_name" animation is given a new image name (such as from application code via the Altia API or from Altia Control code), each instance of the Image object will open the image file and load the image. Each Image object instance will load the image data into the same internal raster structure so the final amount of memory in use will be for just one copy of the image data, but each Image object instance will consume CPU and I/O time/resources to open the image file and load the image data. Also, if code gen for clones is enabled, this optimization is disabled in its entirety. These were enhancements for the 9.2 release.

- Incident #1291: Fix problems for NO_PRELOAD mode when Image objects are sharing rasters. First, software rendering targets could draw the Image object using an old transform. Second, hidden Image objects were unnecessarily deleting shared rasters. This was fixed for the 9.2 release.

- Incident #1294: DeepScreen miniGL Yamaha YGV629 VC1 1-bit raster images were not being displayed properly because the index value into the PLTI configuration was incorrect. This was fixed for the 9.2 release.

- Incident #1296: DeepScreen miniGL changes to isolate negative values from bit fields when data structures are being packed. This was an enhancement for the 9.2 release.

- Incident #1339: DeepScreen miniGL Yamaha YGV629 VC1 pattern memory binary diff utility was added (usercode/YamahaYGV629/vc1pmdiff.exe) to help distinguish the differences between two binary files. For example, to help a developer understand which blocks have changed between two 4MByte binary files. This was an enhancement for the 9.2 release.

- Incident #1341: New Toshiba TMPA900 target license 1025. This is a DeepScreen target to generate graphics code to render to a Toshiba TMPA900 Microcontroller running Linux. The TMPA900 device has an ARM926EJ-CPU core, an LCD controller with image process accelerator, and a touch screen interface. This Linux port requires a unique hardware interface driver developed and provided by Altia for use with the DeepScreen generated code. For instructions to generate code for this target, configure the development environment, and configure the kernel, see the documents usercode/tmpa900/documentation/Overview.doc (code generation instructions), usercode/tmpa900/README.txt (development environment help), and usercode/tmpa900/module/readme.txt (Linux kernel configuration instructions). This was an enhancement for the 9.2 release.

- Incident #1348: DeepScreen Fujitsu Jade generic (no OS) port now defines AltiaUseAnimationIds to 1 by default. This enables the version of the DeepScreen Altia API that uses id numbers for animation names (instead of strings) for improved performance. Support was also added to the generic port for 480x272 display resolution. There was also a change to the generic port of resolve a register offset issue when applying the patch code for Jade Errata E5 (Memory Access Not Coherent). And a syntax error in the generic port driver function driver_pixmapAlloc() was resolved. These were enhancements for the 9.2 release.

- Incident #1349: New DeepScreen Fujitsu Lime target license 1026. This is a DeepScreen target to generate graphics code to the Fujitsu MB86276 "Lime" graphics controller device for 2D applications. The Lime incorporates the display controller, rendering engine and video input unit from the Fujitsu MB86296 'Coral PA' device but the floating point co-processor and all 3D-functions have been removed. Please see the usercode/fujitsuLime/FujitsuLime.pdf file for code generation instructions. The Altia objects and dynamic behaviors supported by this target are similar in scope to miniGL targets. This was an enhancement for the 9.2 release.

- Incident #1350: DeepScreen miniGL changes for the Fujitsu Lime target to improve the raster/alphamask/font load-on-demand-from-flash feature. Loads are limited to copying a specified number of bytes, miniGL gives up control for other tasks after loading the specified number of bytes, miniGL continues loading more bytes when it is re-entered, and the sequence repeats until all bytes are finally loaded. See the usercode/fujitsuLime/FujitsuLime.pdf or FujitsuLime.rtf file for more details (search for LOAD_ON_DEMAND). This was an enhancement for the 9.2 release.

- Incident #1351: New DeepScreen Yamaha YGV629 VC1 target license 1027. This is a DeepScreen target to generate graphics code to the Yamaha YGV629 (code name VC1) device. The VC1 is a Video Display Processor (VDP) using a pattern rendering function by the sprite method. See the Yamaha YGV629 data sheet for details: http://www.yamaha.co.jp/english/product/lsi/prod/pdf/graphic/4gv629a60.pdTo generate code for this device from the Altia Design Code Generation Options dialog, select the "Yamaha YGV629 VC1" target, the Makefile script should be set to ${ALTIAHOME}\usercode\YamahaYGV629\altmake_template.bat and browser for the Additional Target files list .gen file. The Altia objects and dynamic behaviors supported by this target are similar in scope to miniGL targets. This was an enhancement for the 9.2 release.

- Incident #1352: DeepScreen code generator might fail to put external asset files in current project directory (directory containing the .dsn file). This was fixed for the 9.2 release.

- Incident #1354: DeepScreen for support ticket #9693: Text I/O drifts with right or center sibling justify if no sibling actually exists. This was not the behavior prior to the 8.08 and newer releases. This was fixed for the 9.2 release.

- Incident #1355: DeepScreen Fujitsu Jade color buffer can be incomplete if alpha-map color is unchanged from one draw to the next, but first draw used less of the color buffer because it had a smaller stride. This was fixed for the 9.2 release.

- Incident #1358: For DeepScreen miniGL, support is added to target devices for custom animation states on Text I/O and Raster objects. Animation states to change foreground color (for Text I/O), opacity, X/Y position, and hide/show (map/unmap) can now be assigned from the Altia Design Animation Editor and the miniGL generated code will support these animations. As is expected, opacity and position are interpolated between states. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL Language object support is added. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL FreeType font integration is now possible. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL true layer support is added. If objects in the design have an "Object Layer" property, the generated code will have layer support. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL MINIGL_LAYER_REDRAW compiler directive (#define) specifies that all objects within a layer will be redrawn when one or more have changed. Similar to clipping but based upon layers instead of dirty regions. Only layers that experience one or more object changes will be redrawn. This is required for a miniGL MPC560x target since it uses multiple layers and all objects within a layer must be processed when any one object changes. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL MINIGL_LAYER_MASK compiler directive (#define) masks a layer ID used in Altia Design. For the purposes of processing layers, the Altia Design layer ID will be masked with this value. This allows for special bit flags to be used in the layer IDs. Example: a layer mask of 0xff would treat the following layers all as layer 1: 0x0101, 0x0001, 0x0201. This is required for a miniGL MPC560x target since bit flags are used for tiling and GRAM copy. This was an enhancement for the 9.2 release.

- Incident #1358: DeepScreen miniGL raster or alpha masks can now be externalized as a whole by leaving off the mask value in the miniGL .ali file .external rasters token or .external alphamasks token. This was an enhancement for the 9.2 release.

- Incident #1359: DeepScreen TMPA900 target fix to support Image object using files. This was fixed for the 9.2 release.

- Incident #1360: New DeepScreen Linux Framebuffer target license 1032. Generate code for the "Linux Framebuffer" target. See the Altia software usercode/linxufb/README.txt for more information. Includes support for a Freescale i.MX51 device using the Freescale SDK 10.04.01. Select an "imx51" Makefile script and Addition Target files list .gen file. These were enhancements for the 9.2 release.

- Incident #1364: DeepScreen miniGL Yamaha YGV629 VC1 generic utility to interrogate an opened Altia Design session to get a specified part number (as defined by customer). The part number can be used in a custom script in a custom script to embed into the ROM file that is flashed onto the target. A supplier might use this to help differentiate the various clusters they are manufacturing. This was an enhancement for the 9.2 release.

- Incident #1367: DeepScreen i.MX51 support for altiaGL and OpenGL ES 1.1 targets. Also see incident #1415. These were enhancements for the 9.2 release.

- Incident #1369: Freescale i.MX51 support for OpenGL ES 1.1 target. Also see incident #1415 for changes to support the Freescale SDK 10.07.11 in OpenGL ES 1.1, 2.0, and OpenVG targets (which altiaGL linuxfb remains at SDK 10.04.01). These were enhancements for the 9.2 release.

- Incident #1370: DeepScreen miniGL Fujitsu Lime support for 16mb part. This was an enhancement for the 9.2 release.

- Incident #1371: Text I/O max_pixel_count initial value ignored in DeepScreen generated code. This was fixed for the 9.2 release.

- Incident #1375: DeepScreen miniGL fix dynamic raster index bug. Previously for a design with a mix of static rasters (imported images) and dynamic rasters (images showing via Image objects or an imported image with a custom animation), the raster index could be incorrectly computed. This was fixed for the 9.2 release.

- Incident #1376: New Freescale Spectrum target license 1028. This is a DeepScreen target to generate graphics code to the Freescale Qorivva MPC560xS family of 32-bit Power Architecture microcontrollers (MCUs) that address TFT displays in automotive instrument cluster applications. Please see the usercode/spectrum/documentation/Spectrum.rtf file for complete details. The Altia objects and dynamic behaviors supported by this target are similar in scope to miniGL targets. This was an enhancement for the 9.2 release.

- Incident #1377: The "DDB" uncompressed image data code generation mode is enhanced to support a chroma key identifier. The previous algorithm converted transparency masks to alpha channels. For targets like Fujitsu Jade, this prevents rotation/scaling of the image and using an alpha channel is less efficient than using a chroma key color if the hardware supports it. See one of the Fujitsu Jade DDB .gen files for an example usage. This was an enhancement for the 9.2 release.

- Incident #1378: DeepScreen miniGL custom animation transform calc bug fix. This was fixed for the 9.2 release.

- Incident #1386: DeepScreen accuracy of string width calculations is improved for altiaGL targets, framebuffer targets (e.g., Linux Framebuffer), and accelerated targets (e.g., Altera D/AVE, Fujitsu Jade, etc.). This improves string placement especially for the Text I/O sibling justify modes. This was an enhancement for the 9.2 release.

- Incident #1387: New OpenGL ES 2.0 target license 1029. OpenGL ES 2.0 is a 3D graphics interface specification from the Khronos Group (http://www.khronos.org/opengles/). From the code Generation Options dialog, select the OpenGL ES 2.0 target. Generate code for Windows using the PowerVR emulator by selecting the altmake_template.bat or altmake_unicode.bat Makefile script and either win32_PowerVR_float.gen or win32_PowerVR_int.gen for the Additional Target files list .gen file. This PowerVR emulator requires an OpenGL 1.5 or better graphics card. Desktop graphics cards known to be compatible are the ATI Radeon X1000 or better and the NVIDIA GeForce 6 or better. Most Intel chipsets are NOT supported. This target also supports generating code for a Freescale i.MX51 device running Linux and using the Freescale SDK 10.07.11. Select an "imx51" Makefile script and Additional Target files list .gen file. Also see the documentation/imx51_instructions.pdf file for more information about setting up an i.MX51 target device. Limitations: The Windows PowerVR emulator is known to have performance issues. As a result, it is not a great candidate for demonstration purposes. The Altia Drawing Area object (DAO) and OpenGL object are not supported. The Snapshot object IS supported. This was an enhancement for the 9.2 release.

- Incident #1387: DeepScreen OpenGL ES 2.0 reordered calls in initShaders() for Windows PowerVR emulator to work with more graphics cards. This was fixed for the 9.2 release.

- Incident #1388: New OpenVG target license 1031. OpenVG is a vector graphics interface specification defined by the Khronos Group (http://www.khronos.org/openvg/). To generate code for the Windows emulator (please note this is very slow compared to OpenVG running on an actual OpenVG graphics device), select the altmake_templat.bat Makefile script and the win32_float.gen Additional Target files list .gen file. This target also supports generating code for a Freescale i.MX51 device running Linux and using the Freescale SDK 10.07.11. Select an "imx51" Makefile script and Additional Target files list .gen file. Also see the documentation/imx51_instructions.pdf file for more information about setting up an i.MX51 target device. Limitations: At this time, the OpenVG specification does not support fixed point graphics data so a target device must have floating point support. The touch screen of the Freescale i.MX51 evaluation kit hardware has known sluggish response. Altia has implemented a workaround to improve this, but it does not solve the issue completely. Freescale is aware of the issue. Certain gradients in images are not converted to RGB565 effectively causing a banding effect to occur. This might also be the cause for anti-aliasing not working for some images. Please note that this target does not support the Snapshot object at this time. This target was an enhancement for the 9.2 release.

- Incident #1389: DeepScreen for support ticket #10599. Code gen for an empty Deck causes another object's animation to get the Deck's card events if any card events are generated from code/control/stimulus. This was fixed for the 9.2 release.

- Incident #1390: The DeepScreen miniGL lcdWin32 simulator target is updated to support the "Make with Editor API" compile option and to also support the new custom animation and language object enhancements. These were enhancements for the 9.2 release.

- Incident #1391: DeepScreen changes for "signed" types. This was an enhancement for the 9.2 release.

- Incident #1393: DeepScreen miniGL support in paged memory devices to split stencil/raster/mask bits data across defined ROM page boundaries when there are too many of these objects to fit the data in a single ROM page. This was an enhancement for the 9.2 release.

- Incident #1400: DeepScreen Spectrum internal ROM storage and even-only alpha images. Use Layer flag 0x1000 to flag images to be stored in internal ROM (bitsData.c) instead of the default external ROM (rasters.bin). Use Layer flag 0x0400 to flag an alpha image to be processed as "even only" instead of the default "odd and even" (DMA requirement). This was an enhancement for the 9.2 release.

- Incident #1405: Give application code a chance to detect raw stimulus events before they execute animations. The application code can assign a function's address to the global function pointer: void (*AltiaReportEventPtr)(ALTIA_UBYTE eventType) The eventType argument passed to the function will be a value from one of the altiaTypes.h Altia_InputEvent_types defines (for example, ALTIA_DOWN_EVENT_TYPE). This was an enhancement for the 9.2 release.

- Incident #1411: DeepScreen TMPA900 changes (1) to fix noise on display with a dedicated flush of CPU data cache before next screen operation because CPU data cache had old data from LCD Accelerated DMA, (2) to fix a missing check for an alpha pixmap within egl_RasterTransformDraw so an object with a scale animation that was already scaled at initialization time would get correctly drawn, (3) enhanced kernel module so it access functions provided by the DMA driver of the Linux kernel instead of using the DMA interrupt directly, (4) kernel module now registers/unregisters automatically within the sysfs so no more mknod is necessary. As of September 21, 2010, The port was tested on the Glyn-Tonga-Board based on the Linux-Kernel 2.6.34 from KernelConcepts. The latest kernel tree (dated August 2010) must be used to have the LCDDA as a system resource. These were fixes for the 9.2 release.

- Incident #1415: OpenGL ES 1.1 target adds support for a Freescale i.MX51 device running Linux and using the Freescale SDK 10.07.11. At code generation time, browse and select an "imx51" Makefile script and Additional Target files list .gen file in the Code Generation Options dialog. See the documentation/imx51_instructions.pdf file for more information about setting up an i.MX51 target device. Please note that this target is not able to support the Snapshot object. These were enhancements for the 9.2 release.

- Incident #1418: Minor changes to DeepScreen OpenGL ES 1.1 target for Flash object support (changes are safe even when Flash object is not present). This was an enhancement for the 9.2 release.

- Incident #1421: For DeepScreen miniGL targets, when adding new clipping rectangles, the incoming rectangle isn't merged properly. If the incoming rectangle fits inside an existing rectangle, all is fine. However, if the incoming rectangle encapsulates an existing rectangle then the rectangle is added to the list without any additional processing. This problem appears in the customer simulation model as "flicker" since, in some cases, text is painted twice which gives the appearance of darker text. This was fixed for the 9.2 release.

- Incident #1423: Fix DeepScreen X11 crash on window close when NO_PRELOAD mode is enabled. This was fixed for the 9.2 release.

- Incident #1437: The DeepScreen miniGL Yamaha YGV629 target miniGL.ali file is changed to define the ".noraminit textio" option. This clears the content of Text I/O objects at code generation time as requested by the first customer for this target. This was an enhancement for the 9.2 release..

- Incident #1447: Altia Design crash during code generation when a design has an Image object that has its "image_name" animation deleted. This defect was introduced from changes for incident #1291 that went into the limited distribution releases 8.075 MR2, 8.08 MR3, 8.08 MR4, 9.105 FT, and 9.106 FT. This was fixed for the 9.2 release.

- Incident #1448: Altia Design crash during code generation is possible when the code gen is cancelled and the design has Image objects. This defect was introduced from changes for incident #1291 that went into the limited distribution releases 8.075 MR2, 8.08 MR3, 8.08 MR4, 9.105 FT, and 9.106 FT. This was fixed for the 9.2 release.

- Incident #1464: DeepScreen miniGL generated code for lcdWin32 did not compile if the design contained static text. This was fixed for the 9.2 release.

- Incident #1466: DeepScreen Text I/O and MLTO string buffer size off-by-one fix for static memory code generation. The absolute minimum changes were made to bring us back to parity with pre 9.0 releases. These changes were just increasing the affected Text I/O and MLTO string buffers by 1 (no changes to any executing code, just the space reserved for these buffers is increased by 1). These were fixes for the 9.2 release.

17.4 Altia Design 9.1 Enhancements and Fixes

- Incident #1071: For DeepScreen Windows target, if generated code contains a Language or Skin object and there is application code using the Altia API, the link fails with an unresolved external TargetAltiaAnimateId(). This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1150: For DeepScreen, the ActiveX and Java targets are no longer available. This change was made for the 9.1 release.

- Incident #1158: For DeepScreen Fujitsu Jade, if ALTIA_TOUCH_WIDTH and ALTIA_TOUCH_HEIGHT are not defined, define them as ALTIA_SCREEN_0_WIDTH and ALTIA_SCREEN_0_HEIGHT to force input code to use display 0 as the default touch screen input device. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release)

- Incident #1176: For DeepScreen Fujitsu Jade, added FreeType font support. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1176: For DeepScreen Fujitsu Jade, added variable point touch-screen calibration routine to driver. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1176: For DeepScreen Fujitsu Jade, removed DDB Alpha Layer check in egl_BitmapBlt() so some blits will work with the Alpha Layer and DDB. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1180: For DeepScreen Altera D/AVE, need an fmod() function in os_Wrapper.c to conditionally compile into executables for successful testing with DeepScreen test suite. This change was made for the 9.1 release.

- Incident #1187: For DeepScreen Fujitsu Jade, text might draw corrupted because drawable width was not forced to a multiple of 8. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1190: For DeepScreen, Skin object change to not use strupr() because it is not available on all targets (for example, not on Fujitsu Jade on Linux). This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

- Incident #1193: For DeepScreen Fujitsu Jade on Linux, had a synchronization problem between mapped/physical memory. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1194: For DeepScreen Altera D/AVE, Fujitsu Jade, OpenGL ES, and Toshiba targets, added support for upper ASCII character range in optional FreeType font engine. This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

- Incident #1196: For DeepScreen, Image objects were failing to load from files when NO_PRELOAD was defined. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1198: For DeepScreen Fujitsu Jade on Linux, compile Unicode generated source files with -finput-charset=ISO-8859-1 to support upper ANSI character range for strings in data.c and control.c. This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

- Incident #1198: For DeepScreen altiaGL linuxfb target, compile Unicode generated source files with -finput-charset=ISO-8859-1 to support upper ANSI character range for strings in data.c and control.c. This change was made for the 9.1 releases.

- Incident #1215: For DeepScreen Altera D/AVE, update altmake_eek_flash.bat and altmake_template.bat Makefile scripts to automatically copy <DESIGN>.c (if it exists) to altiaSimpleMain.c for easier automated testing. This change was made for the 9.1 release.

- Incident #1216: For DeepScreen WinCE and OpenGLES Windows Makefile scripts, save/restore LIBPATH to support many executions of the script from a Command Prompt window. This change was made for the 9.1 release.

- Incident #1231: For DeepScreen Altera D/AVE, add support for Snapshot object. This enhancement was made for the 9.1 release.

- Incident #1232: For DeepScreen Altera D/AVE, add stubs in os_Wrapper.c for floating point functions and fixed point code gen so that the function calls are at least resolved (such as because they are called from Control code). A fixed point target does not have floating point functionality so the stubs typically just return 0. The stubbed functions are fmod(), sysconf(), floor(), sin(), cos(), and pow(). This change was made for the 9.1 release.

- Incident #1239: For DeepScreen Toshiba Capricorn, might see the message "ERROR - Command List Overflow (possible memory corruption)" on the target (if there is a working stdout) and no graphics will draw. The render command list size must be increased (4000 is the default) by setting the ALTIA_RENDER_LIST_SIZE compiler flag in the build project. An example setting would be something like: -DALTIA_RENDER_LIST_SIZE=15000

- Incident #1245: For DeepScreen, code generation for a design with a Skin object and the "Skin Object use files" code gen option is enabled can crash Altia Design if any objects have invalid properties. This was fixed for the 9.1 release.

17.5 Altia Design 9.0 Enhancements and Fixes

- Incident #994: For DeepScreen generated code compiling with Visual Studio 2005 or 2008, change compile options to -O1 for improved performance. This enhancement was made for the 9.0 release.

- Incident #1060: In DeepScreen, altiaSetBrush is ignoring setting brush to 1. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1069: In DeepScreen, altiaGL, X11, and accelerated targets might draw black pixels on the edges of rotated rasters. Changed for the 9.0 release to draw pixels outside the raster as the background color instead of black. This will make the rasters look better for most situations.

- Incident #1072: In DeepScreen, generating code for views and dynamic memory and then setting a view name might cause a crash. This was fixed for the 9.0 release.

- Incident #1129: For DeepScreen miniGL target, need to generate packing error if any objects have non-affine (distort) transforms because miniGL does not support distort. This change was made for the 9.0 release.

- Incident #1137: For DeepScreen, text is not redrawing after a Language object loads a new configuration. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1149: For DeepScreen, only Windows, altiaGL, and miniGL targets are tested for the new features of the 9.0 release (such as distort). Code generation is temporarily disabled for Altera D/AVE, Fujitsu Jade and Carmine, OpenGL ES, Toshiba Capricorn, WinCE, and X11 for the 9.0 release.

- Incident #1149: For DeepScreen, made Altera D/AVE, Fujitsu Jade, Fujitsu Carmine, Toshiba, WinCE, and X11 targets available in the 9.1 release after they were temporarily removed from the 9.0 release for testing.

- Incident #1153: For DeepScreen, Multi-Line Text object (MLTO) might crash in a corner case when appending to a line and dynamic memory is enabled. This was fixed for the 9.0 release.

- Incident #1157: For DeepScreen, Drawing Area object (DAO) might crash in a corner case when processing key/mouse events. This was fixed for the 9.0 release.

- Incident #1160: For DeepScreen, Clone initialization might crash for very simple designs that have no defined animations. This was fixed for the 9.0 release.

- Incident #1163: For DeepScreen OpenGL ES using Windows PowerVR emulation, executable might crash on exit with nVidia driver 195.62. The defect is in PowerVR. The OpenGL ES PowerVR PC Emulation software has shown to be sensitive to the version of the nVidia card

OpenGL driver on some systems. If a computer has an nVidia card and the DeepScreen executable will not run, it might be related to the version of the nVidia driver. It has been found that the nVidia driver 195.62 (Nov 2009) causes problems on some systems whereas an older nVidia driver (such as 179.24 from Dec 2008) does not.

- No Incident #: For DeepScreen, generated code compiling to integrate with an MFC app has compile error on DebugWindowquitDetected symbol. This was fixed for the 9.0 release.

- Incident #1211: For some DeepScreen accelerated targets (e.g., Altera D/AVE), a distort animation may not reach its maximum size. This is not a defect. Targets that use texture mapping to do distort cannot distort to the same size for large distortion factors compared to targets that distort in software (such as altiaGL).

- Incident #1213: For DeepScreen Altera D/AVE, code generated from Altia Design 9.0 or newer requires at least a 2009 version of the Altera "system" files (e.g., system.h and libsyslib.a) to compile a working executable. Older 2007 "system" files cause bad rendering or lock-ups. If building a .elf file for download, set the Altera Quartus II programmer to use at least a 2009 version of the "system" file altera_3c25_lcd.sof.

17.6 Altia Design 8.08 MR 3 Enhancements and Fixes

- Incident #1041: In Altia Design and DeepScreen, when "clip_on" is enabled, sibling right justify modes still need to apply an offset for backwards compatibility. This enhancement was made for the 8.08 MR3 release.

- Incident #1060: In DeepScreen, altiaSetBrush is ignoring setting brush to 1. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1071: For DeepScreen Windows target, if generated code contains a Language or Skin object and there is application code using the Altia API, the link fails with an unresolved external TargetAltiaAnimateId(). This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1077: For DeepScreen Fujitsu Jade and Carmine, added Hardware APIs altiaHardwareBlocked() (return true if hardware would block because it is out of render lists) and altiaHardwareActive() (return true if hardware is actively rendering). Target documentation/Overview.doc files were updated to describe these additions. These enhancements were made for the 8.08 MR3 release.

- Incident #1084: For DeepScreen Fujitsu Jade, Fujitsu Carmine, OpenGL ES, and Toshiba Capricorn, added #error to indicate Snapshot object is not yet supported if code is generated for it. This change was made for the 8.08 MR3 release.

- Incident #1094: For DeepScreen Fujitsu Jade and Carmine, multiple display controller support was added. See each target's documentation/Overview.doc file for details. This enhancement was made for the 8.08 MR3 release.

- Incident #1094: For DeepScreen Fujitsu Jade and Carmine, added support for cursors on multiple display controllers. See each target's documentation/Overview.doc file for details. This enhancement was made for the 8.08 MR3 release.

- Incident #1094: For DeepScreen Fujitsu Jade and Carmine, added special mode for single display controller to drive 2 displays simultaneously. See each target's documentation/Overview.doc file for details. This enhancement was made for the 8.08 MR3 release.

- Incident #1094: For DeepScreen Fujitsu Jade and Carmine, added target "uties" folder with background raster loading example. This enhancement was made for the 8.08 MR3 release.

- Incident #1094: For DeepScreen Fujitsu Jade, added "generic" port, including Green Hills Software MULTI project files, to use as a starting point for a custom operating system port. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was made for the 8.08 MR3 release.

- Incident #1101: For DeepScreen, Skin object load might fail to move children if data says to just move the parent. This was fixed in the 8.08 MR3 release.

- Incident #1103: For all DeepScreen targets, an image (e.g., Raster object in Altia) with alpha channels would have too much transparency on its edges as it stretched/scaled. This was fixed for the 8.08 MR3 release.

- Incident #1107: For DeepScreen OpenGL ES target, if generating code for native Windows OpenGL, define WIN32_OPENGL_SCREEN_REDRAW as 0 at compile time or in egl_md.h to do faster dirty region redrawing. The default behavior is for WIN32_OPENGL_SCREEN_REDRAW to be defined as 1 in egl_md.h to do slower full window redrawing because this yields the most consistent results on most Windows desktop computers. This enhancement was made for the 8.08 MR3 release.

- Incident #1109: For Altia Design and DeepScreen, changed Skin object data handling to use floating point math to fix object shifting issues. This enhancement was made for the 8.08 MR3 release.

- Incident #1110: For Altia Design and DeepScreen, added _ANCHOR attribute for exported/imported Skin data of Text I/O objects to specify which point of the object to align with the position data. This enhancement was made for the 8.08 MR3 release.

- Incident #1115: For Altia Design and DeepScreen, on importing of Language data, upper range of 8-bit characters (char values 128-255) were not importing correctly. This was fixed for the 8.08 MR3 release.

- Incident #1119: For DeepScreen Fujitsu Jade, using a transparent color on layer 0 may show the color layer which is blended with the Alpha layer. This is just how the Jade behaves. To eliminate the problem, remove transparent color from layer 0 altogether (since layer 0 is on the bottom, this is OK to do). This change was made for the 8.08 MR3 release.

- Incident #1122: For DeepScreen all targets, Image objects might not draw correctly if they were transformed (scaled/rotated) and images are dynamically loaded from a file (by using the "Image Object use files" option at code generation time). This was fixed for the 8.08 MR3 release.

- Incident #1133: For DeepScreen, Skin load may not cause a visual update. This was fixed for the 8.08 MR3 release.

- Incident #1133: For Altia Design and DeepScreen, need a "Strip path from image names" option on exporting and processing Skin data. This enhancement was added for the 8.08 MR3 release.

- Incident #1136: For DeepScreen, animating to full opacity can change another object's opacity. This was fixed for the 8.08 MR3 release.

- Incident #1137: For DeepScreen, text is not redrawing after a Language object loads a new configuration. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1158: For DeepScreen Fujitsu Jade, if ALTIA_TOUCH_WIDTH and ALTIA_TOUCH_HEIGHT are not defined, define them as ALTIA_SCREEN_0_WIDTH and ALTIA_SCREEN_0_HEIGHT to force input code to use display 0 as the default touch screen input device. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release)

- Incident #1176: For DeepScreen Fujitsu Jade, added FreeType font support. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1176: For DeepScreen Fujitsu Jade, added variable point touch-screen calibration routine to driver. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1176: For DeepScreen Fujitsu Jade, removed DDB Alpha Layer check in egl_BitmapBlt() so some blits will work with the Alpha Layer and DDB. See the Fujitsu Jade target's documentation/Overview.doc file for details. This enhancement was added in the 8.08 MR3 release and 9.1 release (the enhancement was not available in the 9.0 release).

- Incident #1187: For DeepScreen Fujitsu Jade, text might draw corrupted because drawable width was not forced to a multiple of 8. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1190: For DeepScreen, Skin object change to not use strupr() because it is not available on all targets (for example, not on Fujitsu Jade on Linux). This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

- Incident #1193: For DeepScreen Fujitsu Jade on Linux, had a synchronization problem between mapped/physical memory. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1194: For DeepScreen Altera D/AVE, Fujitsu Jade, OpenGL ES, and Toshiba targets, added support for upper ASCII character range in optional FreeType font engine. This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

- Incident #1196: For DeepScreen, Image objects were failing to load from files when NO_PRELOAD was defined. This was fixed in the 8.08 MR3 release and 9.1 release (the fix was not available in the 9.0 release).

- Incident #1198: For DeepScreen Fujitsu Jade on Linux, compile Unicode generated source files with -finput-charset=ISO-8859-1 to support upper ANSI character range for strings in data.c and control.c. This change was made in the 8.08 MR3 release and 9.1 release (this change was not available in the 9.0 release).

17.7 Altia Design 8.08 MR 2 Enhancements and Fixes

- Incident #996: In Altia Design, Altia Runtime, and DeepScreen generated code, there is a new Text I/O object "max_pixel_count" animation to limit the visible characters of text to a certain number of pixels. Set this new animation to 0 to disable it. This enhancement was made for the 8.08 MR2 release.

- Incident #997: In Altia Design, Altia Runtime, and DeepScreen generated code, support the new Language object. This enhancement was made for the 8.08 MR2 release.

- Incident #998: In Altia Design, Altia Runtime, and DeepScreen generated code, support the new Skinning object. This enhancement was made for the 8.08 MR2 release.

- Incident #998: In DeepScreen generated code with the "Image Object use files" option enabled, if NO_PRELOAD is defined at compile time and an Image objects is initially hidden at execution time, it's image data is not loaded from the file system until the Image object shows (such as because of a card change for the Deck containing the Image object). This enhancement was made for the 8.08 MR2 release.

- Incident #1015: The DeepScreen code generator now automatically disables irrelevant options so they do not cause unexpected errors during code generation. This way, an option does not remain enabled for the next code generation session if it is inappropriate. For example, "Image Object use files" is automatically disabled when there are no Image objects and then disabling "Use Dynamic Memory" does not cause a code generation error. This change was made for the 8.08 MR2 release.

Incident #1019: For DeepScreen miniGL target, the 16-bit horizontal framebuffer driver might incorrectly draw overlapped objects when the stride is 0. This was fixed for the 8.08 MR2 release.

- Incident #1021: For DeepScreen miniGL target, Text I/O objects with the same text animation were not drawing when the text changed. This defect was introduced when the clipping enhancement was added. This fix was fixed for the 8.08 MR2 release.

- Incident #1022: For DeepScreen altiaGL custom targets, changes were made so that ALTIA_USE_BACKGROUND_COLOR_FLAG works with transformed Raster/Image objects (i.e., rotated/scaled/stretched color images) and Stencil objects (i.e., monochrome bitmaps). This is in addition to the existing support for untransformed Raster/Image objects. See incident #942 for more details about ALTIA_USE_BACKGROUND_COLOR_FLAG. This change was made for the 8.08 MR2 release.

- Incident #1023: For DeepScreen miniGL target, the DAOWin32 simulator window might open behind the Altia Design window when doing "Run on PC" from the Altia Design "Code Generation" menu. This was fixed for the 8.08 MR2 release.

- Incident #1025: In DeepScreen, there is now a "nodither" option for the .gen file %IMAGEFLAG% directive when generating images as uncompressed DDB data. The "nodither" option disables dithering of the uncompressed DDB data. Dithering is enabled by default to smooth the appearance of the image, but it can result in bands of colors for some color depths. For more details, see a sample .gen file that uses the %IMAGEFLAG% (for example, usercode/fujitsu/jade_qnx_int_ddb.gen). This change was made for the 8.08 MR2 release.

- Incident #1025: In DeepScreen, there is a "nopat" option for the .gen file %IMAGEFLAG% directive that was previously undocumented. If it is used, it keeps patterns (used to fill solid vector objects) in a monochrome format instead of an uncompressed DDB data format. For more details, see a sample .gen file that uses the %IMAGEFLAG% (for example, usercode/fujitsu/jade_qnx_int_ddb.gen). The "nopat" option was documented for the 8.08 MR2 release.

- Incident #1026: For DeepScreen miniGL target, support for "Make with Editor API" would be nice for the DAOWin32 simulator target. This enhancement was made for the 8.08 MR2 release.

- No Incident #: For DeepScreen miniGL RenesasH8SX target, the default framebuffer style should be non-inverted. This change was made for the 8.08 MR2 release.

- Incident #1028: For DeepScreen miniGL target, there is a benign compiler warning fix when the clear frame operation is not used. This change was made for the 8.08 MR2 release.

- Incident #1029: For DeepScreen miniGL target, added NULL Text I/O string check to satisfy static analysis. This change was made for the 8.08 MR2 release.

- Incident #1030: For DeepScreen miniGL target, fix benign packing phase compiler warnings to support Microsoft compiler maximum warning level /W3. These changes were made for the 8.08 MR2 release.

- Incident #1031: In DeepScreen, a bad value 0.0 for the Altia_DynamicObject_type structure currentValue element was generated for fixed point code generation. This was fixed for the 8.08 MR2 release.

- Incident #1032: For DeepScreen miniGL target, the %DATECHECKFILE% option should be used in .gen files to copy binary/object/DLL files so that newer versions overwrite older versions. These changes were made for the 8.08 MR2 release.

- Incident #1033: For DeepScreen miniGL target, there were a variety of compiler warnings during the "Pack miniGL Static/Dynamic Data" phase when the Windows host compiler was Visual Studio 2005 or newer. These warnings were fixed for the 8.08 MR2 release.

- Incident #1034: In DeepScreen, timer stimulus triggering Control WHEN blocks might execute twice for each interval instead of one time. This was fixed for the 8.08 MR2 release.

- Incident #1036: For DeepScreen miniGL target, (1) added CRC16 code generation to validate external flash files; (2) Cortex M3 target optimized driver pipeline with generic DHAL implementation, alpha blending (read back) corrected, and minor bug fixes; (3) Cortex M3 target changes to setRasterResourceType() and setMaskResourceType() to accommodate efficient serial flash caching; (4) Cortex M3 target provisions for inlining display HAL API functions as well as serial flash HAL API functions; (5) Cortex M3 WIN32 simulator (uses existing DAOWIN32 simulator but uses same driver pipeline); (6) Cortex M3 target change to build code with -Ohs (speed) optimization level. These enhancements were made for the 8.08 MR2 release.

- Incident #1037: For DeepScreen miniGL target, an image might not draw correctly at execution time when the device dependent raster image was externalized and the corresponding alphamask data was kept internal. This was fixed for the 8.08 MR2 release.

- Incident #1039: The DeepScreen Fujitsu Jade target could have missing pixels for rectangle objects with an opacity less than 100%. This was fixed for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target now supports Unicode. Select the altmake_unicode.bat or altmake_linux_unicode.bat Makefile script at code generation time. This enhancement was made for the 8.08 MR2 release.

- Incident #1040: The DeepScreen Fujitsu Jade target can now support input (mouse/touchscreen) stimulus especially for targets running Linux or QNX. This enhancement was made for the 8.08 MR2 release.

- Incident #1041: Altia Design, Altia Runtime, and DeepScreen generated code, a Text I/O object using a sibling justify mode of 5 or higher (for right or center sibling justify) might show a slight shift in position as characters change if the leading character has a positive offset change. This is fixed for the 8.08 MR2 release. There is still the chance of a shift for sibling justify mode 4 (left sibling justify). Use justify mode of 0 as an alternative.

- Incident #1043: The DeepScreen Fujitsu Jade target had a stride calculation error that could make the blend bitmap too small and generate the error message "Insufficient scratch space for monochrome bitmap!" This was fixed for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target has improved image delete logic. This enhancement was made for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target for Linux had a VSync issue that could cause the screen to flicker. This was fixed for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target had a layer update synchronization issue because sometimes the VSync signal can have a high latency and this may be the VSync that occurred just prior to the last layer update. This issue was resolved for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target for Linux needs a memory manager to keep all memory used by DeepScreen generated code in a single 64 MByte page as required by the graphics controller. This change was made for the 8.08 MR2 release.

- No Incident #: The DeepScreen Fujitsu Jade target for Linux or QNX provide libpng and zlib object libraries for out-of-the-box support of the "Image object use files" option at code gen time. This requires that the target have a file system. This enhancement was made for the 8.08 MR2 release.

- Incident #1045: For DeepScreen, the OpenGL (3D) object jumps when camera location data is changed quickly. Solution is to only update camera x and y when camera z changes. This change requires previous designs to change so that they update the camera z whenever camera x or camera y are changed. This change is made for the demonstration demos/openglobj/viewer.dsn design (open demos/openglobj/OpenGLObj.pdf or the README.txt file for details). This change was made for the 8.08 MR2 release.

- Incident #1045: For DeepScreen OpenGL ES i.MX31 Linux target, OpenGL (3D) object textures only showing their back sides. This was fixed for the 8.08 MR2 release.

- Incident #1046: For DeepScreen OpenGL ES i.MX31 Linux target, does not correctly display the z order of a model in OpenGL (3D) object. This was fixed for the 8.08 MR2 release.

- Incident #1049: For DeepScreen, generated code for OpenGL (3D) object will not compile if "Generate code for stimulus" is disable at code generation time. This was fixed for the 8.08 MR2 release.

- Incident #1050: For DeepScreen Altera D/AVE, Fujitsu Jade/Carmine, and Toshiba CapA/CapM targets, the driver can fail on the first screen draw if Control WHEN blocks for the "altiaInitDesign" event cause off-screen animations to change states. This was fixed for the 8.08 MR2 release.

- Incident #1052: For DeepScreen OpenGL ES i.MX31 Linux target, add OpenGL (3D) object support. This enhancement was made for the 8.08 MR2 release.

- Incident #1055: For DeepScreen altiaGL targets, there was a corner case for a bad memory reference because GetScratchGC() of egl_common.c did not initialize pGC->stipple to NULL. The bad memory reference would only occur if the previous pGC->stipple element did not exist anymore. This was fixed for the 8.08 MR2 release.

17.8 Altia Design 8.075 Enhancements and Fixes

- Incident #1010. For DeepScreen Altera D/AVE target, added support for multiple layers and updated usercode/altera_DAVE/documentation/Overview.doc accordingly. The specific changes were: (1) There is a new Driver API function driver_layerGetConfig(int layer, void * config). It allows the driver layer to reconfigure any layer used in Altia Design. DeepScreen will call this function for each layer used in the design when DeepScreen code is initializing. This function can be used to set application specific layer formats (such as forcing a layer to be d2_mode_alpha8). (2) There is a new compiler flag ALTIA_USE_BACKGROUND_COLOR_FLAG that is enabled (1) by default. It disables/enables the detection of a background color assignment to bitmaps and fonts. The background color is used only when drawing to alpha layers. The background color will determine if the object is drawn in “overwrite” or “blend” mode. By default, the “overwrite” mode is used. The color defined in egl_md.h for BACKGROUND_COLOR_DEFINITION (cyan by default) specifies the color used to select “blend” mode. (3) The default number of layers for the cyclone 2 target is now 3 instead of 1. These enhancements were made for release 8.075.

- Incident #1011. For DeepScreen code generation, if there is an OpenGL object in the design, but not a Drawing Area object (DAO) in the design, dsDrawEx.c generates warnings for undefined drawing functions. This was fixed for the 8.075 release.

- No Incident #. For DeepScreen targets with EGL_USE_CURSOR_HANDLING defined (such as altiaGL linuxfb target), resolved warnings about extra tokens at end of #endif when altiaUtils.c compiles. This change was made for the 8.075 release.

- Incident #1013. For DeepScreen miniGL 16-bit horizontal driver, changes to support Renesas H8SX and Renesas HEW compiler. In miniGLClearFrame(), force 16-bit calculations and do full screen update by scanline instead of a linear progression. These enhancements were made for the 8.075 release.

- Incident #1014. For DeepScreen miniGL code generation, the packing phase would sometimes fail with an assertion when there were animations only used in stimulus definitions. This was fixed for the release 8.075.

- No Incident #. For DeepScreen Fujitsu Jade Linux target, split generated pixel data for images into 4194304 byte arrays to improve GNU compiler performance. This enhancement was made for release 8.075.

- No Incident #. The models libraries imageobjs.dsn, sounds.dsn, more/imageobj.dsn, more/plot.dsn, and more/sound.dsn are updated with improved comments regarding DeepScreen support. The Image object can now read PNG images from the file system for some DeepScreen targets. The Sound object is now supported for some DeepScreen targets. The Dynamic Line/Polygon objects (in more/plot.dsn) are now supported in DeepScreen. These changes were made for the 8.075 release.

- Incident #1018. For DeepScreen miniGL DAO Win32 simulator, the frame buffer is now updated automatically at startup so that the main() routine does not need to call altiaFlushOutput() to force an update the first time. This was changed for the 8.075 release.

17.9 Altia Design 8.07 MR 4 Enhancements and Fixes

- Incident #918. For DeepScreen X11 target, recognize point size fonts and try to load them (which will work if the TTF file is properly installed for the X11 font server to use). The details are described in the Altia Design software installation usercode/X11/README.txt file. This enhancement was added for 8.07 MR4.

- Incident #995. For DeepScreen OpenGL ES code generation with the PowerVR OpenGL ES Windows emulation software, there are changes to support nVidia graphics cards if the design contains an OpenGL (3D) object. These changes were made for 8.07 MR4.

- Incident #999. For the Altia Design Code Generation Options dialog, the target names are improved to better represent each device. This enhancement was added for 8.07 MR4.

- Incident #1000. For DeepScreen altiaGL Windows CE targets, there are fixes for the Image object to use files on the target when EXPANDED_VIRTUAL_MEMORY is defined at compile time. These fixes were added for 8.07 MR4.

- Incident #1002. For DeepScreen targets compiling on Windows, removed all includes of fstream.h. It is unnecessary and does not exist in Visual Studio 2008. These changes were made for 8.07 MR4.

- No Incident #. For DeepScreen miniGL target, swapping on a single byte boundary for encoded bits is fixed. This was fixed for 8.07 MR4.

- No Incident #. For DeepScreen OpenGL ES target and Windows code generation (e.g., specifying win32_PowerVR_float.gen), font character generation is open to all printable characters instead of just characters in the range of 32 (space) to 126 (~). This change was made for 8.07 MR4.

- No Incident #. For DeepScreen Fujitsu Jade target, a lockup issue was resolved when trying to delete an image that is in the current render list. To get this very important change, upgrade to 8.07 MR2.1 or newer.

- No Incident #. For DeepScreen Fujitsu Jade target, (1) added a port to Linux tested with Fujitsu Linux SDK V0.8.3, (2) blits to the frame buffer are suppressed and an error message is printed if the source address is not in the same 64 Mbyte segment as the frame buffer as required by the device, (3) transformed image blits are suppressed and an error message is printed when transforming an image that does not have power-of-two dimensions, (4) changes for blitting on 64-byte boundaries as required by the device, (5) minor changes to comply with Jade GA spec, (6) driver does not initialize DDR FIFO register because it is not graphics hardware related, (7) driver now uses generic PC monitor timings for display resolutions of 320x200 through 1280x1024 instead of the previous single resolution 1280x480 display timings and reference clock value is 658.7MHz to match Jade development hardware. All of these changes were made for 8.07 MR4.

- Incident #1003. For DeepScreen Fujitsu Jade and Carmine targets, made changes for when the display width in bytes is not a multiple of 64. These changes were made for 8.07 MR4.

- Incident #1004. For DeepScreen Fujitsu Jade target, fixed issue with 2D color register having incorrect value after using the 3D color register to draw wide lines. This change was made for 8.07 MR4.

- Incident #1006. For DeepScreen Fujitsu Jade target, disable bilinear filtering for blitting an untransformed image that has transparent pixels to suppress blurring of the image especially for opacities less than 100%. This change was made for 8.07 MR4.

- Incident #1007. For DeepScreen Fujitsu Jade target, corrected width/height off-by-one for blitting an image if it has transparent pixels. This change was made for 8.07 MR4.

- Incident #1008. For DeepScreen Fujitsu Jade target, changes to improve Stencil object rendering. These changes were made for 8.07 MR4.

- Incident #1009. For DeepScreen Fujitsu Carmine target, added maximum memory check for pixmap allocations to avoid going beyond 128 Mbyte video ram capacity. This change was made for 8.07 MR4.

17.10 Altia Design 8.07 MR 3 Enhancements and Fixes

- Incident #979. In DeepScreen, the built-in animation altiaGetObjXY was failing because it was out of order with altiaGetObjWH. It started failing for 8.06 MR9. It was fixed for 8.07 MR3.
- Incident #980. In DeepScreen, if a design contained only a Strip Chart object, the generated code for altiaGL, accelerated, or X11 targets would not compile . This was fixed for 8.07 MR3.
- Incident #987. In DeepScreen altiaGL Windows CE targets, increase default maximum size of expanded memory to 128 Mbytes when EXPANDED_VIRTUAL_MEMORY is defined at compile time and also support the defining of ALTIA_MEM_POOL_MB_PAGES at compile time to change the number of Mbytes for the maximum size (for example -DALTIA_MEM_POOL_MB_PAGES=32 to use 32 Mbytes as the maximum size instead of 128 Mbytes). This enhancement was added for 8.07 MR3.
- No Incident #. For DeepScreen, add ST SPEAr support to the altiaGL linuxfb target (by using altmake_linuxfb_SPEAr.bat as the Makefile Script). This is a device with an ARM926EJ-S processor core. Code compiles using the Linux SDK provided by ST (an arm-linux-gcc toolchain). This enhancement was added for 8.07 MR3.
- No Incident #. For the DeepScreen Fujitsu Jade and Carmine targets, improve the rendering of very large polygons and wide lines. These enhancements were added for 8.07 MR3.

17.11 Altia Design 8.07 MR 2 Enhancements and Fixes

- Incident #847. Data alignment for variables declared as simple types (arrays of bytes or double-bytes) and then accessed as wider types or structures is controlled at compile time by the definition of ALTIA_ALIGN. Please see the Altia Design software installation usercode/altiaGL/README.txt file for more information about customizing ALTIA_ALIGN. This enhancement was added for 8.07 MR2.1.
- Incident #969. In DeepScreen, fix NO_PRELOAD with a PRELOAD_CNT greater than 1 to reuse a loaded DDB if more than one Raster refers to the same image data. Also add PRELOAD_STENCIL_CNT to support limiting the number of simultaneously loaded DDBs for Stencil objects. YES_COMPATIBLE_LAYERS behavior is also improved. Please see the Altia Design software installation usercode/altiaGL/README.txt file for a full description of these capabilities. These changes were added for 8.07 MR2.
- Incident #975. In DeepScreen, Raster load performance is better when YES_COMPATIBLE_LAYERS is defined. Please see the Altia Design software installation usercode/altiaGL/README.txt file for more information about using YES_COMPATIBLE_LAYERS. This enhancement was added for 8.07 MR2.1.

17.12 Altia Design 8.07 MR 1 Enhancements and Fixes

- Incident #736 and #836. Fixed DeepScreen altiaGL Windows CE targets unable to draw multi-segment or vertical lines. This was first observed on Windows CE with a text entry area. The DeepScreen executable would crash as soon as the vertical cursor of the text entry area needed to be drawn. Or, there would be a crash drawing the multi-segment lines of a Tickmark object. The fix is a workaround to an apparent eVC and VS 2005 Windows CE compiler defect. This was fixed for 8.07 MR1.

- Incident #835. For DeepScreen code generation, when the chosen target required specifying a .gen file in the "Additional Target files list:" field of the Code Generation Options dialog, entering a file name that did not exist could crash the Altia Design editor. This was fixed for 8.07 MR1.

- Incident #883. A DeepScreen altiaGL or accelerated target (e.g., OpenGL ES, Altera D/AVE, Fujitsu, Toshiba) extra .gen file can now truly specify as many as 1024 font ranges in a %FONTRANGEFLAG% line because the maximum line length for a .gen was increased to 17600 characters (that is 17 characters per range specification x 1024 ranges = 17408 characters with 190 characters remaining for %FONTRANGEFLAG% and a font name). This was added for 8.07 MR1.

- Incident #891. For DeepScreen, added Unicode support for compiling the Altia API Server code altiaAPIServer.c as the <design>.c file. If the macro UNICODE is defined at compile time, the code uses wide character strings. This was added for 8.07 MR1.
  - Incident #893. Fixed DeepScreen Deck object to not redraw when its card animation state is set and the value is the same as the previous value. This was fixed for 8.07 MR1.

- Incident #894. Fixed DeepScreen altiaGL targets to gracefully ignore drawing characters for which there is no character bitmap. Previously, it would abort drawing the current string and this would cause the initial valid characters to render in the wrong position because the dimensions of the string were miscalculated. This was fixed for 8.07 MR1. For 8.07 MR3 release, added the fix for all accelerated targets (Altera D/AVE, Fujitsu Jade and Carmine, OpenGL ES, and Toshiba). Windows, Windows CE and X11 targets never had a problem.

- Incident #895. For DeepScreen, the %f in Control $format statements would have extra 0 decimal digits in the result compared to Altia Design and Runtime. This was fixed for 8.07 MR1.

- Incident #897. For DeepScreen code generation, the buffer for processing the Control ROUTINE statement was too small to handle very many PARAMS and LOCALS. If the buffer overflowed, any remaining ROUTINES would not get generated into altia/control.c. This was fixed for 8.07 MR1.

- Incident #900. DeepScreen generated code running on Windows could generate extra mouse move events on button presses and this might cause stimulus to behave differently from Altia Design or Runtime. This was fixed for 8.07 MR1.

- Incident #901. Fixed DeepScreen Windows CE executable compiled with VS 2005 or newer crashing when it has to draw an image that contains pixels with alpha components. This was fixed for 8.07 MR1.

- Incident #905. DeepScreen Text I/O "clear" animation does not cause a render if the text is already cleared. This was added for 8.07 MR1.

- Incident #908. DeepScreen Clip object changes to fix extent off-by-one especially for fixed point targets. These were added for 8.07 MR1.

- Incident #920. The DeepScreen altiaGL "genericfb" target is updated for this release to better serve as a starting point for a new custom framebuffer target. Please see the Altia Design software installation file usercode/altiaGL/README.txt for more details. This was added for 8.07 MR1.

- Incident #926. In DeepScreen, if code was generated to use dynamic memory, Control WHEN queue could overflow (because of many Control WHEN block executions for a single event) instead of allocating additional space and DeepScreen execution could get into an infinite loop. This was fixed for 8.07 MR3.

- Incident #927. If a DeepScreen generated Text I/O receives a string that is a number, it generates the number as an event on its "float" and "integer" animations. It should round the number to an integer for the "integer" event just like Altia Design and Runtime, but it was not doing this. This was fixed for 8.07 MR1.

- Incident #934. In DeepScreen, if a design contains an Image Object and code was generated with the "Image Object use files" option, the Image Object might allocate memory that it did not free if the PNG file load failed because of a corrupt PNG file. This was fixed for 8.07 MR1.

- Incident #935. Setting the "hilight_color" animation for a DeepScreen generated Text I/O or Multi-line Text object (MLTO) using a string of hex numbers in the style "#RRGGBB" would give the wrong color. This was fixed for 8.07 MR1.

- Incident #936. There is a new Text I/O justify mode. It is "justify_mode" value of 7 and it is only valid if "clip_on" is non-zero. This new "justify_mode" right justifies to a sibling object (just like the existing "justify_mode" values of 4, 5, and 6 justify to a sibling) and clips text on the left side instead of the right side as is the case for the other justify modes. This new justify mode is supported in Altia Design, Altia Runtime, and DeepScreen generated code. This was added for 8.07 MR1.
  - Incident #937. DeepScreen generated code now supports the "altiaSetOpacity" built-in animation if the "Generate code for Built-in Functions" option is enabled at code generation time. This was added for 8.07 MR1. For more about this built-in animation, please see the Altia Design 8.0 User's Guide, Animation Editor Chapter 7, Section 7.8.9 - Miscellaneous Functions.

- Incident #938. The DeepScreen targets that compile on Linux (such as the altiaGL linuxfb target) now explicitly link with -lz (the libz.a file compression library) when the generated code contains an Image Object and the "Image Object uses files" option is selected at code generation time. Some Linux systems require explicitly linking with -lz. Other targets do not require this, but still accept the -lz option at link time. This was added for 8.07 MR1.

- Incident #939. All DeepScreen targets that generate code for compiling on Windows now protect all render activity with a semaphore. This avoids unexpected reentrancy into the rendering code if there are windows in the application that are not DeepScreen windows. This was added for 8.07 MR1.

- Incident #942. DeepScreen altiaGL custom targets can now choose two background colors for raster and text objects to support special behaviors at the driver level. See the Altia Design software installation file usercode/altiaGL/README.txt for details. Search for ALTIA_USE_BACKGROUND_COLOR_FLAG in the README.txt file. This enhancement was added for the 8.07 MR1 release and then improved for the 8.08 MR2 release to support two background colors instead of just one.

- Incident #944. DeepScreen Text I/O objects were failing to correctly justify 90 degrees rotated text (based on the "justify_mode" setting). This was fixed for 8.07 MR1.

- Incident #945. If the CALLBACKSIZE macro is defined at compile time (such as with a -DCALLBACKSIZE option at compile time), the DeepScreen version of the Altia API code now makes sure that the selected event name array is at least as large as the size defined by the CALLBACKSIZE macro. This was added for 8.07 MR1.

- Incident #947. All DeepScreen Windows and Windows CE Makefile scripts now support a system with a VS 2008 compiler installation. This was added for 8.07 MR1.

- Incident #949. DeepScreen rotated text (especially 270 CW) was changing pixels beyond the extent of the characters. This was fixed for 8.07 MR1.
- Incident #955. In DeepScreen, custom targets can enable inlining of fixed point multiply and divide with the ALTIA_USE_INLINE_FX_MATH macro defined to 1 at compile time. This was added for 8.07 MR1. Please see the Altia Design software installation usercode/altiaGL/README.txt file for more information.
- Incident #956. In DeepScreen, custom targets can enable target driver 64-bit fixed point math with the ALTIA_USE_DRIVER_FX_MATH macro defined to 1 at compile time. This was added for 8.07 MR1. Please see the Altia Design software installation usercode/altiaGL/README.txt file for more information.
- Incident #957. In DeepScreen, custom targets can disable background fill for performance with the ALTIA_NO_BACKGROUND_FILL macro defined to 1 at compile time. This was added for 8.07 MR1. Please see the Altia Design software installation usercode/altiaGL/README.txt file for more information.
- Incident #967. The Altia Design models library decks.dsn has been changed for 8.07 MR1 so that the components it contains are well suited for DeepScreen code generation. Earlier versions of the components contained Control code that might not compile for some deeply embedded DeepScreen targets.
- Incident #971. For DeepScreen custom target development, see the new usercode/altiaGL/README.txt file with hints for developing a new target and the various flags for configuring the execution of the code. This was added for 8.07 MR1.
- Incident #972. Changed DeepScreen altia/wCharStrings.c to use stdargs.h by default. It will only use varargs.h if OLD_VARARGS is defined at compile time. This was added for 8.07 MR1.

17.13 Altia Design 8.06 MR 9 Enhancements and Fixes

The following are incidents following the April 2007 8.062:

- Incident #335. Increased DeepScreen code generator structure Altia_CustomSValue_type element value[256] to value[512]. This doubles the size of space available for the string in an MLTO at code generation time. This fix was made prior to 8.06 MR9. This is not the absolute best fix, but it will help a lot especially for MLTO used to hold radio station info in many projects. The best fix would be to allocate the size, but this has complications for UNICODE support.
- Incident #789. Support for compiling DeepScreen generated code as a DLL for Windows and Windows CE ARM and x86 targets. This was added prior to 8.06 MR9.
- Incident #801. For Altia Design, Runtime, and DeepScreen, flipped/rotated bitmaps were missing a row of pixels on 1 or 2 edges and duplicating a row of pixels on opposite edge. The fix was completed for DeepScreen prior to 8.06 MR9. Altia Design still needs a fix so that when the flip/rotate is performed in the editor, it does not change the position of the flipped/rotated object by 1 pixel horizontally or vertically. This fix will be in a later release.
- Incident #842. DeepScreen images were failing to keep proper transparency when rotating. This is fixed for altiaGL and accelerated targets (e.g., Altera D/AVE and OpenGL ES), but not yet for Windows native and WinCE native (WinCE ARM through WinCE x86) targets. This fix was made prior to 8.06 MR9. More improvements were made for the 9.0 release. DeepScreen Windows target still shows some artifacts, but all other DeepScreen targets do not.
- Incident #849. In DeepScreen, an extent change for a hidden object (such as changing the text for a hidden Text I/O object) was still causing a redraw of the area even though there was no visual change. This fix was made prior to 8.06 MR9.

- Incident #852. There was a problem merging extents for a target with layers. The problem would result in some areas not redrawing. This fix was made prior to 8.06 MR9.
- Incident #853. Code additions were made to improve support for Unicode in altiaGL targets (especially linuxfb). If UNICODE is defined and WCHAR_IS_WCHAR_T is defined and ALTIA_CHAR_UNICODE is not defined (it is only defined for the ActiveX target), the compiler is assumed to be compliant with the C99 standard and conditional statements make sure the compiled code will be compatible with the C99 standard. This fix was made prior to 8.06 MR9.
- Incident #854. If ALTIAGL is defined at compile time, the altiaAPICbk.c code can include a driver os_Wrapper.h to get OS time function declarations (instead of attempting to directly include time.h or sys/time.h). This adds flexibility for supporting targets that do not have an operating system. This fix was made prior to 8.06 MR9. Please see the Altia Design software installation usercode/altiaGL/README.txt file (the section labeled "Overview for Getting Time for Altia API Timer Callbacks") for more information.
- Incident #855: The DeepScreen code generator was ignoring a $STARTOPTION or $ENDOPTION when it had a trailing tab. This fix was made prior to 8.06 MR9.
- Incident #858: For DeepScreen, if an image with alpha component is all opaque at code gen time, it is now generated as a 24-bit image instead of 32-bit image to conserve memory. This fix was made prior to 8.06 MR9.
- Incident #859. For DeepScreen, a fix was made for a crash when the built-in animation "altiaPopToTop" is done for an object that has a parent. This fix was made prior to 8.06 MR9.
- Incident #862: For DeepScreen "Windows" target code generated with the "Generate code for clones" option, the executable would crash on exit/close if clones existed and it would not clean up all allocated memory if it did not crash. These fixes were made prior to 8.06 MR9.
- Incident #867. Image object reading PNG files for DeepScreen Windows CE was added. Initially just doing it for ARM and X86 targets. This fix was made prior to 8.06 MR9.
- Incident #868: For DeepScreen "Windows" target code generated with the "Generate code for clones" option, creating a new clone would fail if the new clone number was the same as the object id for a child object of an already existing clone (got it?). This fix was made prior to 8.06 MR9.
- Incident #869. If Image obj was not showing image at DeepScreen code gen time, it caused a bad extent calculation at execution time and could crash the target (for example, altiaGL windib). This fix was made prior to 8.06 MR9.
- Incident #870. A transformed Image object might show bad colors in DeepScreen altiaGL targets when the image comes from a file and the target has a relatively low color depth. This was fixed prior to 8.06 MR9. altiaGL must up-convert low color depth (e.g., 16-bit 565) color bits to 24-bit color when creating a DIB from a DDB.
- Incident #871: DeepScreen generated code compiling on Windows was compiling with /MTd for VS 2005. When not compiling for debug with VS 2005, must set NODEBUG=1 so that compile does not have /MTd as an argument. /MTd forces a reference to libcmtd.lib. This causes problems linking with external code compiled with /MT. There are conflicts between libcmtd.lib and libcmt.lib. This was fixed prior to 8.06 MR9. Problem was discovered using VS 2005 with Simulink/RTW.
- Incident #873: Added built-ins altiaGetObjWH/altiaObjW/altiaObjH to DeepScreen. See 8.06 MR9 or newer Enhancements Summary for details.
- Incident #876: Fixed bad DeepScreen code gen for a control EXPR statement when left operand holds a string (Text I/O, MLTO, or GLOBAL), right operand is a string array, and the index for the right operand string array is an animation name. Generated code would not compile. The

code generator thought the index animation value needed to become a string. This was fixed prior to 8.06 MR9.

- Incident #879: Added support for line widths to 100 pixels for DeepScreen altiaGL and X11 targets. Previous, only line widths up to 16 pixels were supported. This is already supported on Windows and Windows CE. The thickness of lines supported by accelerated targets (e.g., OpenGL ES, Altera D/AVE, Fujitsu, Toshiba) depends on the capabilities of the underlying graphics accelerator. This was fixed prior to 8.06 MR9.

- Incident #880: If DeepScreen Windows CE code was generated for a design with a transparent area, the transparent area color would show around objects. This was fixed prior to 8.06 MR9. This defect was introduced when the border style for a Windows CE window was changed to WS_EX_CLIENTEDGE in 2006.

- Incident #881: If DeepScreen Windows CE code was generated for a design with a transparent area, the window disappeared when executing on the Windows CE device. This was fixed prior to 8.06 MR9. This defect was introduced when view support was added for Windows and CE.

- Incident #882: When DeepScreen Windows or Windows CE code is generated for a design with transparent area(s), set main window to be on top. This is different from runtime and X11 (at least for now), but it seems like it is how the user would typically want it. Disable this new behavior by defining ALTIA_NOT_TRANSPARENT_TOPMOST at compile time. This was fixed prior to 8.06 MR9.

- Incident #884. An "alpha" element is added to the Drawing Area object (DAO) AltiaEx_GraphicState_type data structure to support DeepScreen DAO rendering with an alpha channel component when using the "altiaEx" interface to the DAO. For now, the DAO in Altia Design or Altia Runtime is ignoring the new "alpha" element to minimize the impact of the change (i.e., DAO DLLs developed to run with Altia Design and/or Altia Runtime do not require source code changes and recompiling). The OpenGL object source code and DLL and DAO "exapi" demo were updated to set the "alpha" element. When writing application code to use the "altiaEx" interface to the DAO for DeepScreen generated code, set the "alpha" element of AltiaEx_GraphicState_type data structures to 255. If this is not done, most likely the DAO will appear not to render. In reality, it is rendering with an alpha component of 0 or close to 0 (i.e., transparent). If using the OpenGL DLL, update to the new version "bin/oglobj.dll" from 8.06 MR9 or newer.