

Using Altia® with MATLAB® Simulink®

Altia® Design and DeepScreen® for Windows®

HMI Software

April, 2014



Table of Contents

1	Introduction	3
2	Installing the Altia Connection for Simulink	4
3	Matlab Installation Requirements	6
3.1	Configure 32-bit MATLAB to Use a Visual Studio or LCC Compiler	6
3.2	Configure 64-bit MATLAB to Use a Visual Studio Compiler	6
4	Tutorial Overview	7
5	Use Simulink to Create a Simple Model.....	8
6	Use Altia Design to Create a Simple GUI	9
7	Connect Your Altia GUI to Your Simulink Model.....	13
8	Test Your Altia GUI/Simulink Model Interaction	15
9	Using Altia Runtime with Simulink Generated Code	16
10	Important Technical Details	19
11	Doing a Simulink Code Generation Build that Compiles the Altia Block Code as Unicode ..	21
12	How to Use DeepScreen Generated Code with Simulink Generated Code	23
12.1	Start with the Existing Tutor Simulink Model and Altia Interface	24
12.2	Generate and Compile DeepScreen Code for the GUI	24
12.3	Use MATLAB and Simulink to Build a Fully Functional Executable	28
13	Tutorial Summary	30

Table of Figures

Figure 1-	Simulink Model Layout.....	8
Figure 2 -	Altia S-Function Parameters Dialog	9
Figure 3 -	Available Sliders in the Sliders Library	10
Figure 4 -	Slider Object Selected	10
Figure 5 -	Slider Properties with Changes	11
Figure 6 -	Slider Object with Changes	11
Figure 7 -	Finished Meter and Slider	12
Figure 8 -	Connections Pane with Meter and Slider Selected.....	13
Figure 9 -	Connection Pane with All Connections Linked	14
Figure 10 -	The Completed Simulink Tutor Model	24
Figure 11-	The Completed AltiaDesign.dsn GUI.....	25
Figure 12 -	Altia Code Generation Options Dialog Configured for Windows Code Generation	26
Figure 13 -	Example Successful Completion of Code Generation	27
Figure 14 -	Example Successful DeepScreen Generated Code Compile	28

1 Introduction

This document accompanies an Altia developed connection to MATLAB Simulink. The Altia Connection for Simulink is an optional purchase.

The Altia Connection for Simulink provides a method for using Altia Design HMI (Human-Machine Interface) development software with The MathWorks® Simulink product and Simulink Code Generation (formally Real-Time Workshop) on Microsoft® Windows® to create an HMI for a Simulink model. In this document, HMI may also be referred to as GUI (Graphical User Interface), UI (User Interface), or graphical front panel.

The Altia Connection for Simulink provides an interface between an Altia HMI and Simulink model running in Simulink simulation mode on Windows or as a generated code executable on Windows. The Altia HMI runs in Altia Design, Altia Runtime, or as Altia DeepScreen generated code for Windows.

The Altia Connection for Simulink is not portable to embedded systems. Altia recommends using Stateflow® in conjunction with the Altia C API to interface to Altia DeepScreen generated code for an embedded target. The Altia Connection for Simulink package includes a demonstration showing how to use Altia with Stateflow.

The Altia Design package includes an editor, runtime engine and numerous libraries of components for quickly creating a user interface to Simulink simulations. In addition to using the supplied component libraries to create HMIs, users can make modified versions of the existing components or create custom components without programming. The benefits of such graphical front panels include involving the customer in the design process, increasing collaborative design and debugging models more efficiently.

This step-by-step tutorial will guide you through the process of creating a simple Simulink model, creating a GUI in Altia Design and then connecting the two.

Please see the next chapter for instructions to install the Altia Connection for Simulink.

2 Installing the Altia Connection for Simulink

For Microsoft Windows, the connection is packaged as a self-extracting ZIP file with a file name like `altMLzip.exe`. Perform the following steps to execute the self-extracting ZIP and install it:

1. In a Windows Explorer window, browse to the folder containing `altMLzip.exe` and double-click on it to execute it.
2. The result from step #1 is a dialog with Connection version information, the releases of MATLAB supported by the Connection, and additional installation instructions. Please read the dialog carefully and follow the instructions to extract the remaining setup files for the Connection. The additional installation instructions recommend extracting the remaining setup files to the MATLAB software work folder. This will make them easy to find from MATLAB. For example, on Windows 7, the MATLAB work folder is something like:

```
C:\Users\<USER_NAME>\Documents\MATLAB
```

On Windows XP, it is something like:

```
C:\Documents and Settings\<USER>\My Documents\MATLAB
```

Make note of the folder to which the setup files are extracted. It will be necessary to go to this folder in MATLAB to complete the installation.

3. Start MATLAB. **On Windows 7 or newer, you must start MATLAB as Administrator.** To start MATLAB as Administrator, right click on the MATLAB program icon (such as from the Windows **Start > All Programs > MATLAB R20XXx** menu item or from the **MATLAB R20XXx** program icon on the Windows Desktop if it is available) and choose **Run as Administrator**. If you do not have Administrator privileges, please ask your administrator to login and start MATLAB.
4. In MATLAB, change the current folder to where the setup files were extracted. For example, use the Current Folder interface tools to do this or use the `cd` command in the MATLAB Command Window (e.g., `cd C:\Users\<USER_NAME>\Documents\MATLAB`) to change directory to the folder.
5. At the prompt in the MATLAB Command Window, type:

```
altsetup
```

Press the keyboard <Enter> key. This executes the `altsetup.m` MATLAB script. It is one of the Connection setup files. This will do the following:

- Create a directory named `altia` under the `<MATLAB_ROOT>\toolbox` directory.
- Extract Connection files to this new `<MATLAB_ROOT>\toolbox\altia` directory.
- Add `<MATLAB_ROOT>\toolbox\altia\bin` to the MATLAB path.

6. The altsetup script will then ask where Altia Design is installed on your computer. Typically, this is in something like c:\usr\altia<Version_Info> (for example, c:\usr\altia11Build783). Just type in wherever you have Altia installed and press the <Enter> key.
7. The Altia Connection for Simulink package installation is now complete.
8. This document only describes how to use the Altia Connection for Simulink. If you are interested in how to interface Altia to Stateflow, please go to the installed <MATLAB_ROOT>\toolbox\altia\demos\AltiaStateflowDemo folder and open the **AltiaStateflowDemo.pdf** file for detailed instructions. The Altia Connection for Simulink is not portable to embedded systems. Altia recommends using Stateflow in conjunction with the Altia C API to interface to Altia DeepScreen generated code for an embedded target.

3 Matlab Installation Requirements

As of April 2014, this connection supports 32-bit or 64-bit versions of MATLAB.

3.1 Configure 32-bit MATLAB to Use a Visual Studio or LCC Compiler

Altia's recommendation and the default configuration for the Altia Connection for Simulink is to use a Microsoft Visual Studio compiler for building Simulink generated code. Configure MATLAB to use a Microsoft Visual C/C++ 2008 or newer compiler. For 32-bit versions of MATLAB, the MATLAB supplied LCC compiler is also an option. To configure the compiler for MATLAB, run the **mex -setup** command at the MATLAB Command Window prompt and follow the instructions displayed by the command.

If 32-bit MATLAB is configured to use LCC, be sure to follow the extra instructions later in this document for setting up Altia blocks to use LCC prior to a Simulink code generation build.

3.2 Configure 64-bit MATLAB to Use a Visual Studio Compiler

Configure 64-bit MATLAB to use a Microsoft Visual C/C++ 2008 or newer compiler for building Simulink generated code. The MATLAB LCC compiler is not supported by the Altia Connection for Simulink on 64-bit MATLAB. To configure the compiler for MATLAB, run the **mex -setup** command at the MATLAB Command Window prompt and follow the instructions displayed by the command.

4 Tutorial Overview


The goal of this tutorial is to show the basic link between Simulink and an Altia Graphical User Interface (GUI). In this tutorial, we will discuss every step necessary to develop a simple system that exercises this connection.

Before you begin this 30-minute tutorial, make sure you have both Altia Design and MATLAB with Simulink installed on your computer.

- The first section of this tutorial ([Use Simulink to Create a Simple Model](#)) describes how to design a model that has a simple gain block which outputs to, and is fed by, an Altia Design S-Function block.
- In the next section ([Use Altia Design to Create a Simple GUI](#)), we will step through the few procedures required to build a GUI that has a slider and a meter.
- The process of connecting our Simulink model to our Altia GUI is addressed in the next section, [Connect Your Altia GUI to Your Simulink Model](#). We will connect the two such that the slider bar is the input to the gain and the meter will show the gain's output.
- The [Test Your Altia GUI/Simulink Model Interaction](#) section shows you how to verify that your GUI and your Simulink model are connected correctly.
- The section [Using Altia Runtime with Simulink Generated Code](#) explains how to use your Altia GUI with Simulink Generated Code (referred to as Real-Time Workshop in older releases of Simulink).
- At this point in the tutorial, we summarize some [Important Technical Details](#). In addition, there is a section that describes [How to Build the Altia Block for Unicode](#).
- The last section, [How to Use DeepScreen Generated Code with Simulink Generated Code](#), describes creating a single Windows executable for the Altia GUI + Simulink model.

5 Use Simulink to Create a Simple Model

For the purposes of this tutorial, we will just use the MATLAB working directory to hold our files.

1. The first thing to do is start MATLAB.
2. From the MATLAB **Home** ribbon, click on the **Simulink Library** option (or in the MATLAB Command Window, type `simulink`) to open the Simulink Library Browser window.
3. Click on the New Model icon  (or do **File > New > Model**) to make a new Simulink model. The new Simulink model is typically given the name **untitled**.
4. Highlight the **Altia Connection** element in the **Libraries** pane of the Simulink Library Browser window. This shows the contents of the Altia Connection library. The only component of this library is the **AltiaDesign** block. Press the left mouse button on this component and, while still holding the left mouse, drag the component into the Simulink model.
5. Next, drag a **Gain** block from the **Simulink > Math Operations** library into the Simulink model. At this point, we can close the Simulink Library Browser window because we will not be adding any more components to our simple model.
6. Let's change the value of our Gain block by double-clicking on it, typing **0.8** and clicking the **OK** button.
7. Next, let's connect the **AltiaDesign** block to the **Gain** block as in the picture below.
8. We also want to label the inputs and outputs of the AltiaDesign block. Double-click on the output line of the AltiaDesign block and label it **gain feed**. Label the input line to the AltiaDesign block **gain result**.
9. We are now finished designing our simple Simulink model. Save your model by pressing the Save icon in Simulink (or do **File > Save**). When asked what to name your model, type in `tutor.slx` (or `tutor.mdl` in older releases of Matlab).

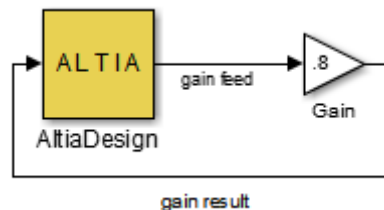


Figure 1- Simulink Model Layout

6 Use Altia Design to Create a Simple GUI

1. Open the Altia Design editor by double-clicking on your **AltiaDesign** block in the Simulink **tutor** model window. This will open an Altia S-Function Parameters dialog.

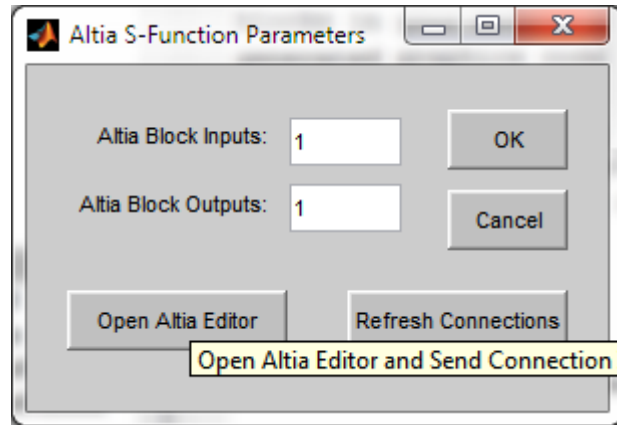


Figure 2 - Altia S-Function Parameters Dialog

2. We could change the number of inputs or outputs for our AltiaDesign block here so that many signals could come from or be sent to our Altia GUI. For right now, we will just stick with one input and one output. By the way, if you ever want to know what a feature of this dialog does, just let your cursor linger over it and a tool-tip will appear.
3. Since we want to open the Altia Design editor to create our GUI, press the **Open Altia Editor** button. Informational messages like the following will appear in the MATLAB Command Window:

ALTIA: Started Altia editor (AltiaDesign.dsn, Port AltiaDesign, Id=0).
ALTIA: Connection information sent to Altia.

Altia Design will open with a blank design file and name it **AltiaDesign.dsn** because the editor was started by double-clicking on the Altia block named **AltiaDesign**. The design file will reside in the same folder as the Simulink model file.

The name of the Altia design file and its location are very important. The name must match the name of the Altia block in Simulink and the folder location must be the same as the Simulink model file. Never perform a **File > Save As** in Altia Design and save to a .dsn file name that is different from the Altia block name in Simulink. Never save to a .dsn file name that is in a different folder than the Simulink model file.

4. If Altia Design opens with a black canvas area in the drawing universe, let's change it to white. Click the Altia Design **View** button. In the View ribbon, open the **Canvas Color** menu and choose the white color box. It is the box in the top-right corner. If you linger over it with the mouse cursor, it shows RGB color values for white in decimal (255,255,255) and hex (#FFFFFF). After clicking the white color box, the canvas area of the drawing universe should change to white. The canvas area size is typically 480 x 272 pixels which is fine for our GUI design.

5. In this initial design, we will only have a slider bar and an analog meter. We will use objects from Altia's included component libraries.
6. To create the slider bar, click on the Altia Design **Insert** button to open the Insert ribbon. This displays available component libraries. Click on **Sliders** to open the Sliders library. It opens in its own window. If the Sliders library window is covering up the drawing universe white canvas area, move the Sliders library window to show the canvas area.
7. Pull the slider of your choice into your Altia design by pressing and holding the left mouse button over the slider and then dragging it into the Altia Design drawing universe canvas area (the area of the drawing universe that is white). Close the Sliders library window from the **X** button in the top-right corner of the window.

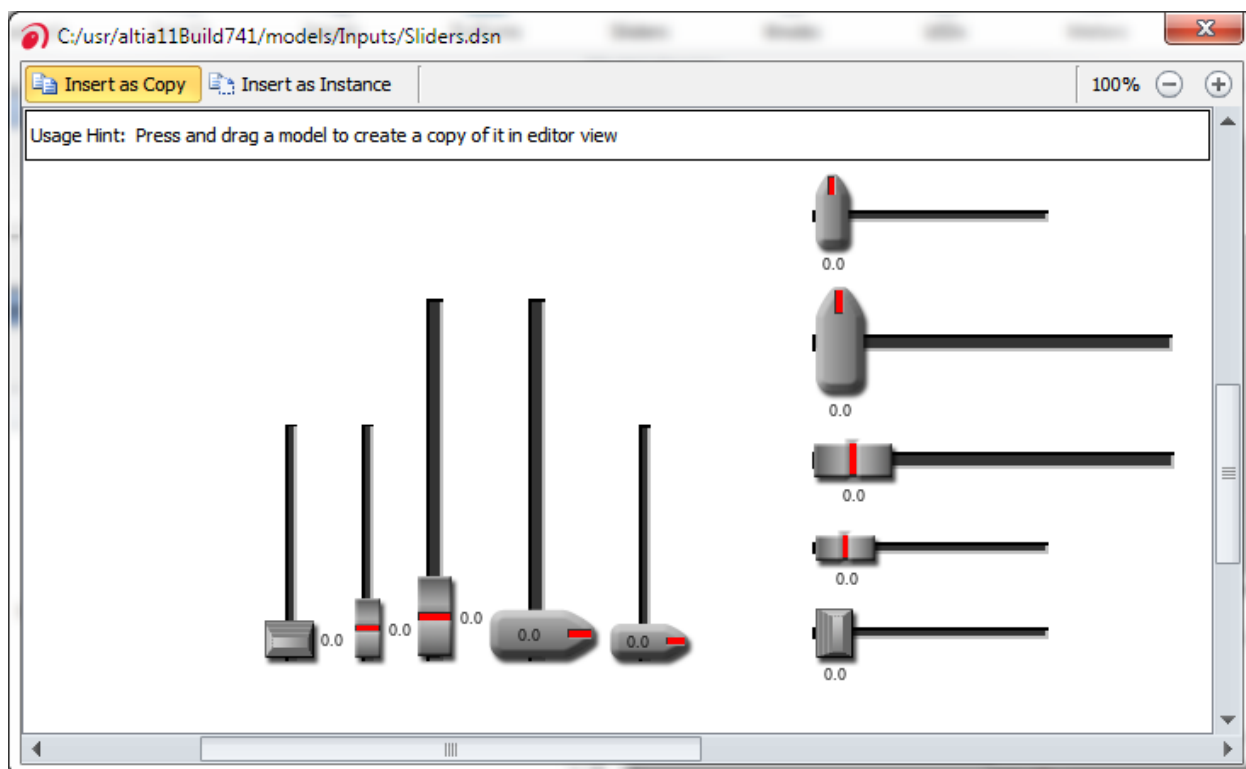


Figure 3 - Available Sliders in the Sliders Library

8. Return to the Altia Design main window and left-click on the slider to select it in the drawing universe canvas area if it is not already selected. It will show with selection handles around it. The drawing universe must be in **Edit** mode, not **Run** mode, to select the slider.



Figure 4 - Slider Object Selected

9. Open the **Properties** pane. With the slider object selected, this will show the properties of the slider. Change the **Right Side Value** property from **100** to **125**. Change the **Show Label** property drop-down box to **Yes**. Set the **Label Color** to **blue** (just replace the R,G,B number string with the word blue, press the <Enter> key, and the word blue is replaced with its R,G,B number string of 0, 0, 255). Set the **Value Color** to **red**.

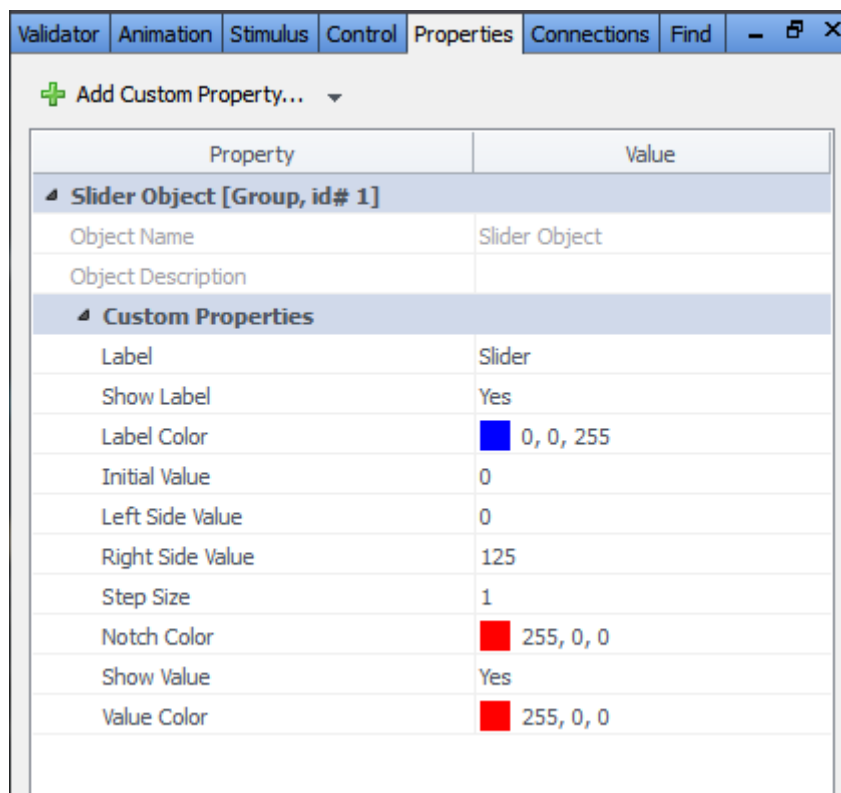


Figure 5 - Slider Properties with Changes

10. There are several other properties associated with this slider. We won't be changing any other properties for this tutorial, but feel free to experiment with them. Properties make changing the behavior and appearance of the component extremely easy. When you are finished, the slider should look something like the next picture.



Figure 6 - Slider Object with Changes

11. Now it is time to add a meter to the GUI. Click on the Altia Design **Insert** button to open the Insert ribbon (if it is not already opened from inserting the slider). This displays available component libraries. Click on **Meters** to open the Meters library. It opens in its own window. If the Meters library window is covering up the drawing universe white canvas area, move the Meters library window to show the canvas area. Choose an analog meter, drag it into the Altia Design drawing universe canvas area (the area of the drawing

universe that is white), and drop it below or above the slider. Close the Meters library window from the **X** button in the top-right corner of the window.

12. The meter should be the selected object after dragging and dropping it. If it is not, click on it to select it. In the **Properties** pane, change the **Text Label** to **Gain Meter** and the **Value** **Decimal Digits** property to **2**. The finished meter and slider should look something like the next picture.

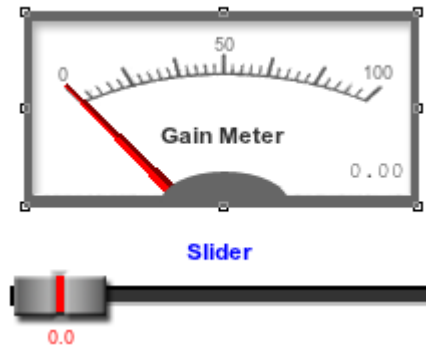


Figure 7 - Finished Meter and Slider

13. Now, let's save our GUI. In the Altia Design editor, just do **File > Save**. We have now saved this simple design to the file `AltiaDesign.dsn`.

The name of the Altia design file and its location are very important. The name must match the name of the Altia block in Simulink and the folder location must be the same as the Simulink model file. Never perform a **File > Save As** in Altia Design and save to a `.dsn` file name that is different from the Altia block name in Simulink. Never save to a `.dsn` file name that is in a different folder than the Simulink model file.

Although the graphics look neat, right now they don't do much because we have yet to connect them to anything. This is our next task.

7 Connect Your Altia GUI to Your Simulink Model

We must now connect the input/output labels of our Simulink model to the Altia GUI objects so that the Simulink **gain result** can control the Altia meter and the Altia slider can control the Simulink **gain feed**. This is easily done with Altia.

1. Click anywhere in the empty space of the Altia Design drawing universe area and then press Ctrl+A on the keyboard to select all objects (in this case, the meter and the slider).
2. Open the **Connections** pane. It should look like the next picture.

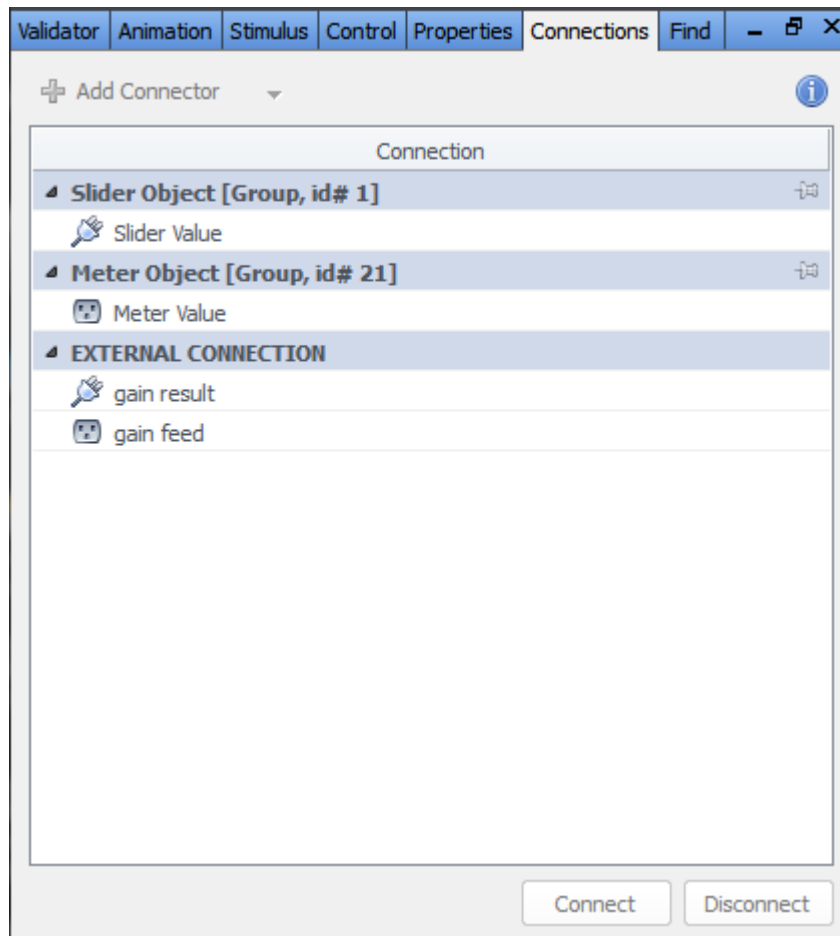




Figure 8 - Connections Pane with Meter and Slider Selected

3. Plugs  are outputs (they generate values) and sockets  are inputs (they receive values). A plug can only connect to a socket and vice versa. We are aware that the metaphor of the electrical plug (output) icon and socket (input) icon is not technically accurate, but it is easy to understand that plugs can only connect to sockets and vice versa. In this situation, we want to connect the plug (output) **Slider Value** of our Slider Object to the socket (input) **gain feed** External Connection. The socket (input) **Meter Value** of our Meter Object needs to be connected to the plug (output) **gain result** External Connection. It is easy to do this from the Connections pane.

4. Left mouse click on the **Slider Value** plug to highlight it. With the **Ctrl** key pressed on the keyboard, left mouse click on the **gain feed** socket to also highlight it. Both items should now be highlighted and the **Connect** button at the bottom-right of the Connections pane should be enabled. Press the **Connect** button at the bottom-right of the Connections pane to link the **Slider Value** plug to the **gain feed** socket. If you make a mistake and have the wrong items highlighted, just click in empty space of the Connections pane to unhighlight all highlighted items and start over.
5. Using the same technique, link the **Meter Value** socket to the **gain result** plug. The Connections pane should now show the connections linked as in the next picture.

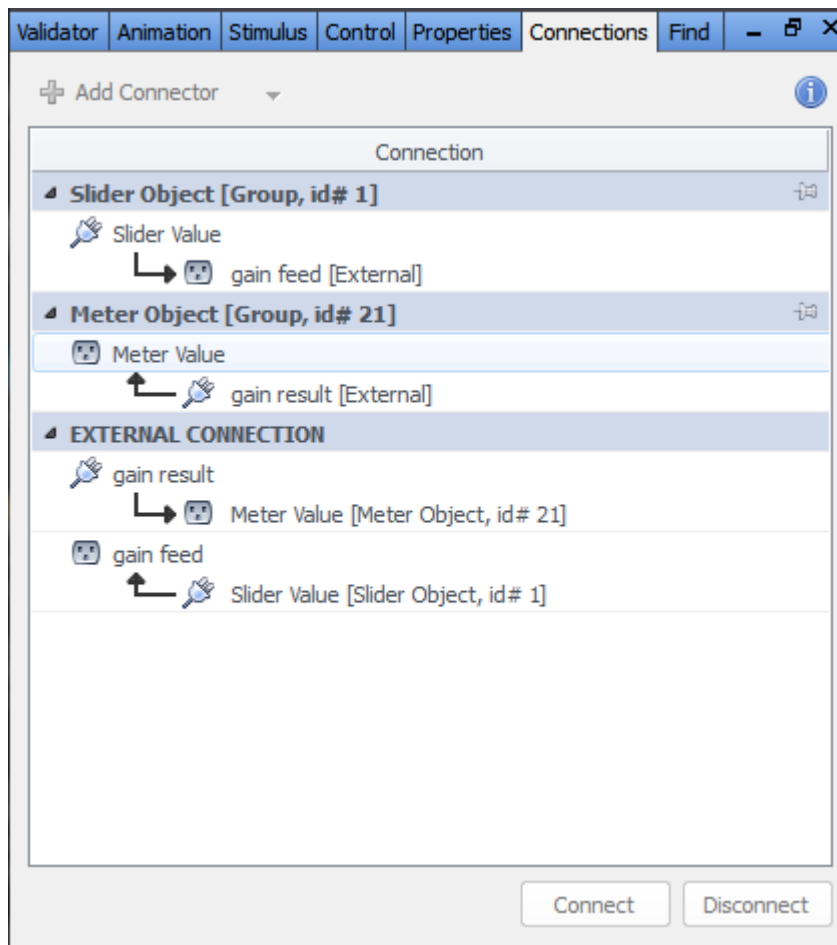


Figure 9 - Connection Pane with All Connections Linked

6. The connections have now been established. Save the changes (the connections are really the only changes) to your Altia Design GUI by choosing **Save** from the **File** menu.

CAUTION: You must save whenever external connections are modified. Otherwise, the changes will not be recognized by Simulink. The name of the Altia design file and its location are very important. The name must match the name of the Altia block in Simulink and the folder location must be the same as the Simulink model file.

8 Test Your Altia GUI/Simulink Model Interaction

1. Select the Simulink editor window that contains our **tutor** model. From the Simulink **Simulation** menu, choose **Model Configuration Parameters**. This displays the Configuration Parameters dialog.
2. In the **Solver** section, change the **Stop time:** to **inf** (just type `inf` into the field) so that the simulation will run continuously. While we are in the **Solver** section, change the Solver options **Type:** to **Fixed-step**. This will be necessary for later when we generate code and build an executable from the Simulink model (otherwise, code generation will fail with a Variable-step solver type). Click **OK** to set the parameters and close the dialog.
3. We are now ready to simulate. From the Simulink **Simulation** menu, choose **Run** and the simulation will begin. Your Altia Design editor window should still be displaying your GUI design from the previous section (Connect Your Altia GUI to Your Simulink Model). Change your focus to the Altia Design window. If the Altia Design editor is not already opened when a simulation starts, an Altia Runtime window automatically opens with your GUI design.
4. In Altia Design, you are probably in Edit mode. Switch the Altia Design editor to Run mode by clicking on the **Run** button in the top-left corner of the drawing universe area. If you want to unselect all objects before going into Run mode, click anywhere in the empty space of the drawing universe area (or press **Edit** and then `Ctrl+D`) and then press **Run** mode.
5. Press the left mouse button on the handle of the slider object and drag the mouse from side to side. You will notice as the slider handle moves, the meter changes to reflect the status of the Simulink gain output (which is 80% of the slider value).
6. While the simulation is running, you can go back to Edit mode in Altia Design, reposition objects, changes their properties, etc. Then simply return to Run mode and exercise the slider to generate new gain feed values for the Simulink model.
7. If Altia Design is not running when a simulation starts, it was previously mentioned that an Altia Runtime window automatically opens. An Altia Runtime window opens for each unique Altia block in the Simulink model. The size of the window and its background color are the size of the drawing universe canvas area in Altia Design and its color. Each Altia block will have a unique name and therefore be associated with a unique Altia design (.dsn) file. When a simulation is stopped, Altia Runtime windows do not automatically close. You must close them manually (from the X button in the top-right corner of the Altia Runtime window). When a simulation starts again, it will open one or more new Altia Runtime windows.

9 Using Altia Runtime with Simulink Generated Code

If you would like to create a Windows executable of your simulation that can connect to your Altia interface (so that you may distribute it to someone who does not have Simulink), you will need to generate and compile code from Simulink. To generate and compile code for a Simulink model containing one or more Altia blocks, several extra steps are required.

Before getting started, let's review the prerequisites for Simulink code generation:

- You must have a license from The MathWorks for Simulink Coder.
- For 32-bit MATLAB, Altia's recommendation and the default configuration for the Altia Connection for Simulink is to use a Microsoft Visual Studio compiler for building Simulink generated code. Configure MATLAB to use a Microsoft Visual C/C++ 2008 or newer compiler. For 32-bit versions of MATLAB, the MATLAB supplied LCC compiler is also an option. To configure the compiler for MATLAB, run the **mex -setup** command at the MATLAB Command Window prompt and follow the instructions displayed by the command.
- If 32-bit MATLAB is configured to use LCC, be sure to follow the extra instructions in this chapter for setting up Altia blocks to use LCC prior to a Simulink code generation build.
- **Configure 64-bit MATLAB to use a Microsoft Visual C/C++ 2008 or newer compiler for building Simulink generated code.** The MATLAB LCC compiler is not supported by the Altia Connection for Simulink on 64-bit MATLAB. To configure the compiler for MATLAB, run the **mex -setup** command at the MATLAB Command Window prompt and follow the instructions displayed by the command.
- Generating code and building an executable for a Simulink model containing Altia blocks is only supported for execution on Windows. The underlying Altia block source code is not supported for Altia DeepScreen embedded target code generation. Altia blocks in Simulink models are excellent for creating front panels for Windows desktop simulation purposes, but the Altia block source code is not portable to an embedded target. Altia recommends using Stateflow in conjunction with the Altia C API to interface to Altia DeepScreen generated code for an embedded target. If you are interested in how to interface Altia to Stateflow, the <MATLAB_ROOT>\toolbox\altia\demos\AltiaStateflowDemo folder contains a demonstration. Browse to the folder in Windows Explorer and open the **AltiaStateflowDemo.pdf** file for detailed instructions.

We will use our Simulink **tutor** model to demonstrate the creation of a Windows executable from a Simulink model.

1. Open the **tutor** model in Simulink if it is not already opened.
2. In the MATLAB Command Window, type:

```
altiaartwsetup
```

This runs the Altia Connection for Simulink code generation setup script for Simulink models containing one or more Altia blocks. RTW is an acronym for Real-Time Workshop. In earlier versions of Simulink, the C code generation feature was called Real-Time

Workshop and this is how the `altiarwsetup` script got its unusual name. This script will:

- Copy all of the necessary Altia Runtime files (`altiaart.exe`, `colors.ali`, `fonts.ali`) to the current project folder (i.e., the folder containing the Simulink model file and Altia Design GUI files). The resulting executable from doing a Simulink code generation and build will be created in this same folder. When it executes, it must find the Altia Runtime executable (`altiaart.exe`) and its support files (`colors.ali` and `fonts.ali`) in the same folder.
 - Create the folder `<Model_Name>grt_rtw` (for our example, `tutor_grt_rtw`) if it does not already exist. By default, Simulink generates code to this folder and compiles it from this folder. The folder name must be of this form if the Simulink model contains Altia blocks. If the folder name is changed, compiling Simulink generated code will fail if the Simulink model contains one or more Altia blocks.
 - Copy `altiasfn.obj`, `libsdde.lib`, and any other necessary Altia specific source code and object files to the `<Model_Name>grt_rtw` folder. Without these files, Simulink generated code will not compile/link if the Simulink model contains one or more Altia blocks.
3. **If and only if the compiler is LCC with 32-bit MATLAB**, also execute this script in the MATLAB Command Window:

```
altialccsetup
```

When prompted to choose an Altia API style, enter 1 to select to use LCC with DDEs. Only enter 2 to select LCC with TCP/IP if there is a very good reason (i.e., you really, positively know what you are doing). This script reconfigures the Altia block in a Simulink model to use LCC as the compiler. Altia has tested this up to 32-bit MATLAB R2014a. The Altia Connection for Simulink may stop supporting this capability in future releases.

If it is necessary to go back to using Microsoft Visual Studio as the compiler, reconfigure Simulink code generation with **mex -setup** and choose the appropriate version of Microsoft Visual Studio as the compiler. Then reconfigure the Altia block with the following command in the MATLAB Command Window:

```
altialibsetup
```

When prompted to choose an Altia API style, enter 1 or 3 to select to use DDEs. Only enter 2 to select TCP/IP if there is a very good reason (i.e., you really, positively know what you are doing).

4. It's time to generate code and compile it from Simulink. In the Simulink window, choose **Code > C/C++ Code > Build Model** (in older releases of Simulink, it is **Tools > Code Generation > Build Model**). Please note that it may be necessary to change your solver type to Fixed-step for a build to complete successfully. This is done by choosing **Model Configuration Parameters** from the **Simulation** menu and selecting the Solver tab.

5. After the code generates and compiles successfully, a new executable file `<Model_Name>.exe` (for our example, `tutor.exe`) resides in the current project folder (i.e., the folder containing the Simulink model file and Altia Design GUI files). You can run this executable stand-alone (i.e., without running Simulink). For example, open a Windows Explorer window, browse to the project folder, and double-click on the executable file (for our example, `tutor.exe`).
6. The executable (for our example, `tutor.exe`) opens a Windows Command Prompt window because it is compiled as a console application. Very soon after the Command Prompt window opens, the Altia Runtime window opens to display the Altia GUI. Use the Altia GUI components to exercise the stand-alone simulation executable just like in Simulink simulation mode. Closing the Altia Runtime window does not stop the executable. It continues to run until the Command Prompt window is closed by clicking on the **X** button in its top-right corner. Similarly, closing the Command Prompt window does not close the Altia Runtime window. You must manually close an Altia Runtime window by clicking on the **X** button in its top-right corner.
7. You can package up the stand-alone files (such as in a ZIP archive) , unpack them on another Windows computer, and run the simulation stand-alone. For our **tutor** Simulink model example with an Altia block named **AltiaDesign**, the necessary files are:

```
AltiaDesign.dsn  
AltiaDesign.mxc  
AltiaDesign.rtm  
altiaart.exe  
colors.ali  
fonts.ali  
tutor.exe
```

For this set of files, executing `tutor.exe` starts the stand-alone simulation.

10 Important Technical Details

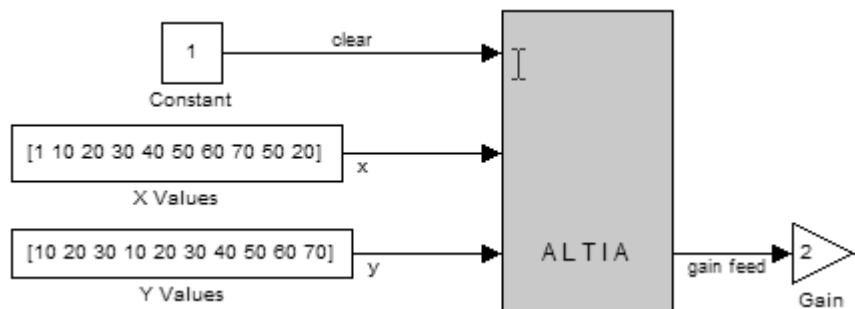
There are a few technical details that may be of interest:

- When your simulation finishes, your Altia Runtime windows will remain open so that you can view the interface's final states. If you wish to run the simulation again, you should close these Altia Runtime windows first. This frees up the system resources that they were using. Altia Design editor windows are re-used and do not need to be closed.
- You can have as many as 50 Altia blocks in a Simulink design connected to 50 different .dsn files (very cool!). Each block must have a name that is unique so that it references its own .dsn file (i.e., two or more Altia blocks cannot control the same .dsn file). If you fail to give each Altia block a unique name, the result will be very strange behavior during simulation.
- The Altia block(s) in the Simulink design will look in the current MATLAB directory to find files (.dsn's, .mxc's, etc.). Use the `pwd` command in the MATLAB Command Window to determine the current working directory, the `dir` command to see what files reside in the current working directory, and the `cd` command to change to the correct working directory if necessary.
- If you want to pass a string from Simulink, a character at a time, into an Altia text object, modify the connection of the object in Altia to have the option **continuous**. The indication that this is needed is that a word with double letters such as `connect` will appear as `conect`. To change a connection to be continuous, highlight the connection in the Connections pane, right-click, and choose **Edit Connector...** to open the connection edit dialog window. Press on the **Show Advanced** button. In the **Simulation Options:** field, type the word **continuous** and press the **OK** button.
- You can send vectors into the Altia block. For each execution of the block, the block will loop through the inputs for the maximum vector size. On each loop, the block will send the Nth item of each vector input to Altia. That is, you can have two separate arrays of X and Y values feeding into an Altia block and be confident that X1 and Y1 are sent as a pair then X2 and Y2, etc.

NOTE: When using vectors, you usually will want to change the connection to be **continuous** so that the connection will send consecutive values that are the same. That is, if the vector is `[98 98]`, only one 98 is sent to Altia if the connection is **not** specified as **continuous**. To force repeating values to be sent, modify the connection of the object in Altia to have the option **continuous**. To do this, highlight the connection in the Connections pane, right-click, and choose **Edit Connector...** to open the connection edit dialog window. Press on the **Show Advanced** button. In the **Simulation Options:** field, type the word **continuous** and press the **OK** button.

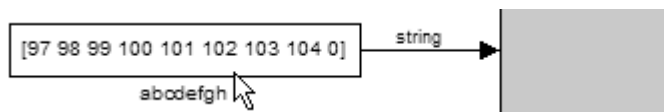
In the following figure, the `clear` signal is sent a 1, then `x` is sent 1 and `y` is sent 10. Next, `x` is sent 10 and `y` is sent 20. `clear` is not sent again yet, because it has no second item (it isn't a vector at all). The block will keep looping through until it gets to the last `x`

and y pair of 20 and 70 and then the block will start all over again with clear, sending its inputs to Altia.



- You can also use the vector capabilities to send strings to Altia (be sure the last number of the vector is a zero).

NOTE: When using vectors to send strings, you should change the connection to be **continuous** so that the connection will send consecutive values that are the same. That is, if the vector is [98 98], only one 98 is sent to Altia if the connection is **not** specified as **continuous**. To force repeating values to be sent, modify the connection of the object in Altia to have the option **continuous**. To do this, highlight the connection in the Connections pane, right-click, and choose **Edit Connector...** to open the connection edit dialog window. Press on the **Show Advanced** button. In the **Simulation Options:** field, type the word **continuous** and press the **OK** button.



11 Doing a Simulink Code Generation Build that Compiles the Altia Block Code as Unicode

This capability was added to the 08/19/2011 or newer version of the Altia Connection for Simulink.

Doing a Simulink Code Generation Build that compiles the Simulink Altia block code as Unicode is desirable for a model containing both a Simulink Altia block and Stateflow code where the Stateflow code is using the Unicode version of the Altia API. A Simulink Code Generation Build for such a model requires the Simulink Altia block to also use the Unicode version of the Altia API (because the Simulink Altia block code and Stateflow code reside in a single executable so they must link with the same version of the Altia API interface).

The steps to configure for a Simulink Code Generation build that compiles the Altia code as Unicode are:

1. Select the Altia block in the Simulink model, right click, choose **Mask > Look under Mask** and change the **S-function module:** field to:

```
libddeUnicode.lib user32.lib
```

Or, use the following if compiling with /MD:

```
libddeMDUnicode.lib
```

2. If you already did a Simulink code generation build prior to changing the **S-function module:** field, it may be necessary to delete the buildInfo.mat file from the <Model_Name>_grt_rtw folder. Altia's experience is that some releases, such as R2013b or newer, do not recognize the Simulink model has changed when only the **S-function module:** field is changed. By deleting the buildInfo.mat file, the Simulink code generation build regenerates all source code files.
3. From the Simulink window, do **Code > C/C++ Code > Code Generation Options...** and highlight **Code Generation** in the left pane. In the **Build configuration:** drop-down list, change the setting to **Specify**. Add this option to the **C Compiler** section (do not change any existing options, just add this option):

```
-DALTIAUNICODEAPI
```

Or, if linking with DeepScreen code, add:

```
-DALTIAUNICODEAPI -DDEEPCCREEN
```

Press **OK** to apply the changes and close the Configuration Parameters window.

4. For older versions of Simulink, step #3 is different. From the Simulink window, do **Tools > Real-Time Workshop > Options...** and modify the **Make** command to look like:

```
make_rtw OPTS="-DALTIAUNICODEAPI"
```

Or, if linking with DeepScreen code, change it to look like:

```
make_rtw OPTS="-DALTIAUNICODEAPI -DDEEPSCREEN"
```

5. After any execution of `altiarthwsetup` or `deepscreenRTWsetup` from the Matlab Command Window, delete the resulting `<Model_Name>_grt_rtw/altiasfn.obj` file if it exists (because it might not be a Unicode version).
6. Do a Simulink code generation build as usual.

12 How to Use DeepScreen Generated Code with Simulink Generated Code

The Altia Design package includes an editor, runtime engine and numerous libraries of components for quickly creating user interfaces. In addition to using pre-built components from included Altia libraries to create graphical front ends, the Altia Design product allows users to make modified versions of existing components and create custom components in the editor without programming. With these features, a modeler can quickly create a user interface for product simulations that looks and behaves like the product's real user interface.

Altia DeepScreen graphics code generation software allows users to generate deployable graphics code from almost any interface created in Altia Design. DeepScreen can take pre-built components from included Altia libraries as well as custom components and produce tight, efficient C code that compiles into a single standalone executable.

The goal of this chapter is to take an existing Simulink model with an Altia interface and turn it into a Windows executable that uses DeepScreen generated graphics code.

Some **advantages** of this approach over using Altia Runtime with Simulink generated code are:

- The result is a single executable (such as `tutor.exe` for this tutorial) and just the associated Altia `.mxc` file (such as `AltiaDesign.mxc` for this tutorial). The Altia Runtime executable (`altiaart.exe`) and its support files (`colors.ali` and `fonts.ali`) are not required.
- This approach does not require distribution of the Altia `.dsn` file as is necessary when using Altia Runtime with Simulink generated code. This is important if the `.dsn` file has proprietary content.
- External images referenced by Altia Image objects can be generated as DeepScreen code and compiled directly into the executable. This is useful if there are many external images and/or they are considered to have proprietary content.

The primary **disadvantage** of this approach is:

- The Simulink model must have only one (1) Altia block. DeepScreen generated code can only integrate with Simulink generated code for a single Altia GUI.

Before getting started, let's review the prerequisites for integrating DeepScreen generated code with Simulink generated code to build a Windows executable:

- You must have a license from Altia for the DeepScreen Intel x86 Software Render Windows code generator.
- You must have a license from The MathWorks for Simulink Coder.
- **The Simulink model must have only one (1) Altia block.** DeepScreen generated code can only integrate with Simulink generated code for a single Altia GUI.

- **You must use a 32-bit version of MATLAB and configure it to use a Microsoft Visual C/C++ 2008 or newer compiler.** A 64-bit version of MATLAB is not supported (because DeepScreen generated code for Windows only compiles as 32-bit object code) and the MATLAB LCC compiler is not supported for building a Windows executable that uses DeepScreen generated code. To configure the compiler for MATLAB, run the **mex -setup** command at the MATLAB Command Window prompt and follow the instructions displayed by the command.
- Generating DeepScreen code and integrating it with generated code for a Simulink model containing an Altia block is only supported for execution on Windows. The underlying Altia block source code is not supported for DeepScreen embedded target code generation. An Altia block in a Simulink model is excellent for creating a front panel for Windows desktop simulation purposes, but the Altia block source code is not portable to an embedded target. Altia recommends using Stateflow in conjunction with the Altia C API to interface to DeepScreen generated code for an embedded target. The Altia Connection for Simulink <MATLAB_ROOT>\toolbox\altia\demos\AltiaStateflowDemo folder contains a demonstration showing how to interface Altia to Stateflow. Browse to the folder in Windows Explorer and open the **AltiaStateflowDemo.pdf** file for detailed instructions.

12.1 Start with the Existing Tutor Simulink Model and Altia Interface

We will use the Simulink model and Altia GUI already created in the previous chapters of this tutorial. The Simulink model file is `tutor.slx` (or `tutor.mdl` in older releases of Matlab) and its Altia block references the Altia Design `AltiaDesign.dsn` file.

12.2 Generate and Compile DeepScreen Code for the GUI

Follow these steps to generate DeepScreen code and compile it.

1. Open the **tutor** model in Simulink (the Simulink model file is `tutor.slx` or `tutor.mdl` in older releases of Matlab). The model should look very similar to the next picture.

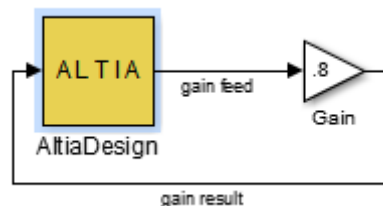


Figure 10 - The Completed Simulink Tutor Model

2. Double-click on the **AltiaDesign** block and choose the **Open Altia Editor** option to open the `AltiaDesign.dsn` file associated with the **AltiaDesign** block. The Altia GUI that opens in Altia Design should look very similar to the next picture.

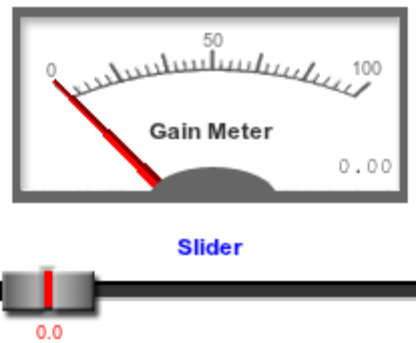


Figure 11- The Completed AltiaDesign.dsn GUI

3. Do **not** select any objects of the Altia GUI. We wish to generate code for the entire GUI with a window size the size of the white canvas area of the drawing universe. By not selecting any objects, we will get our wish. If objects are selected, then code is only generated for the selected objects and the window size will just be the area of the selected objects.
4. Click on the Altia Design **Code Generation** button to open the Code Generation ribbon.
5. From the Code Generation ribbon, choose the **Generate Source Code** option. This opens the Code Generation Options dialog similar to the next picture.

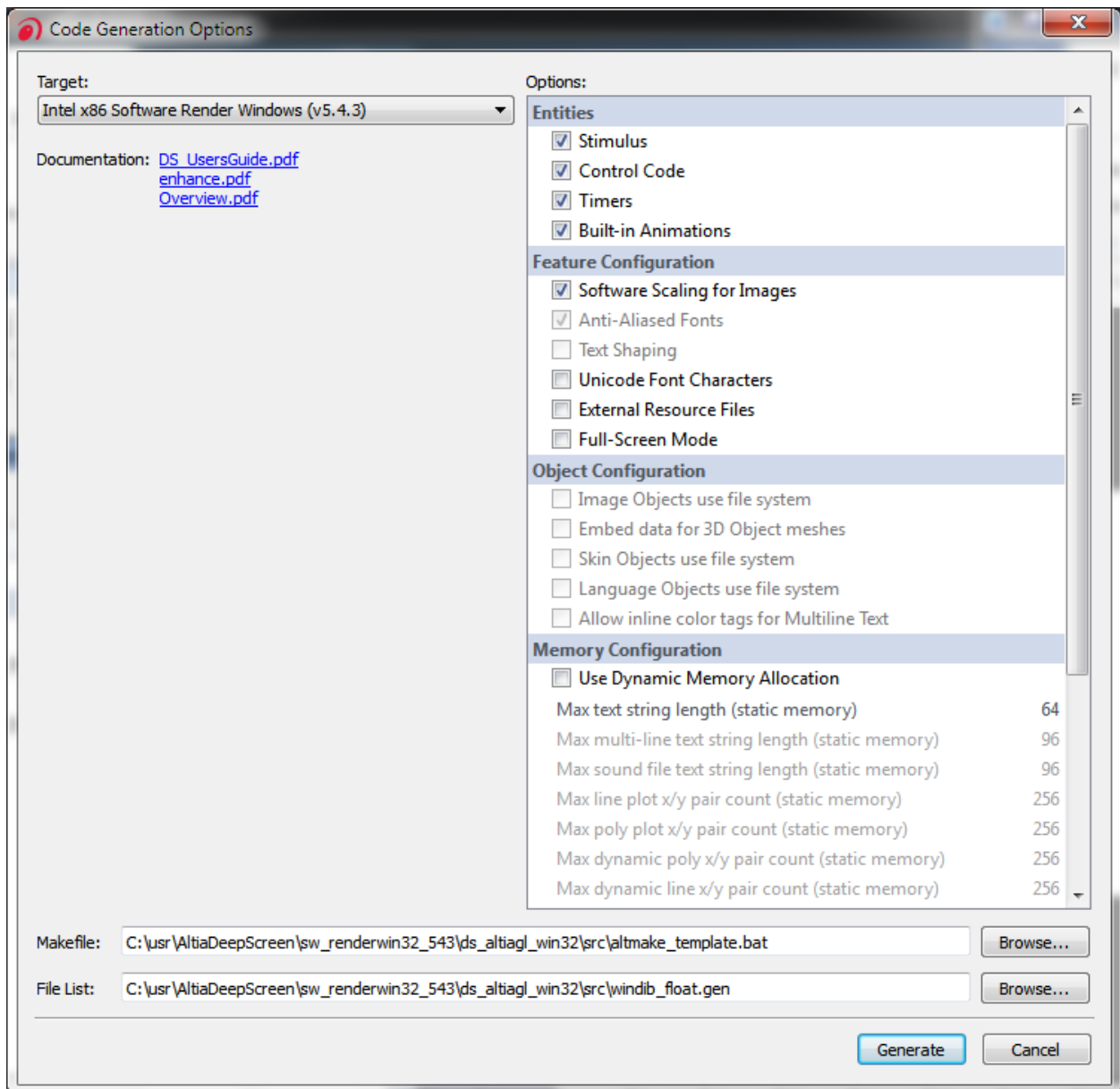


Figure 12 - Altia Code Generation Options Dialog Configured for Windows Code Generation

6. The above picture shows the typical settings for generating DeepScreen code for a Windows desktop application. The Code Generation Options dialog layout or exact wording on options may change from one release of Altia Design to the next, but it generally looks like this picture. Note the following:
 - The chosen **Target** is a version of **Intel x86 Software Render Windows** (in this picture, it is version 5.4.3).
 - The Options selected are **Stimulus** (to generate code for mouse/keyboard events), **Control Code** (to generate code for any Control logic in the meter or slider components if they use Control logic), **Timers** (to generate code for timers if any timers are used), **Built-in Animations** (to generate code for the Altia built-in

animation functions supported by DeepScreen), **Software Scaling for Images** (must be enabled for this target because it transforms images using software, not hardware acceleration), and **Anti-Aliased Fonts** (to generate code for smoothing the edges of text characters).

- Press the **Browse...** button to the right of the **Makefile:** field to select a Makefile script. The script to typically choose is `altmake_template.bat` for this target.
 - Press the **Browse...** button to the right of the **File List:** field. The file to typically choose is `windib_float.gen` for this target.
7. Now that the options are set, press the **Generate** button in the bottom-right corner of the Code Generation Options dialog to generate the DeepScreen graphics code for the GUI.
 8. Upon successful completion of code generation, there will be a dialog with a **Done** button enabled similar to the next picture. Press the **Done** button to close the dialog.

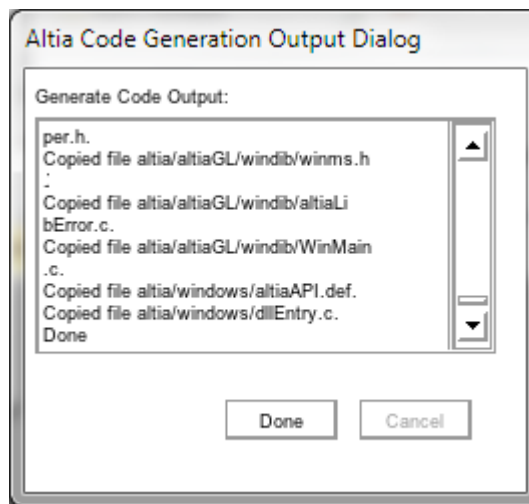


Figure 13 - Example Successful Completion of Code Generation

9. Altia Design should still be showing its Code Generation ribbon. Press the **Make Standalone** button. This opens a Windows Command Prompt window and executes Visual Studio compiler commands to compile the generated code. This is a necessary step to create object libraries `altiaWinLib.lib`, `altiaAPILib.lib` and `altiaAPIfloat.lib` from the DeepScreen generated source code. Later, the Simulink code generation build will link these object libraries with its generated code to create an executable. A successful compile session looks like the next picture.

```

C:\usr\altia11Build783\bin\runmake.exe
PS -01 -I"altia/altiaGL\libfloat" -I"altia/altiaGL" -I "altia/altiaGL/mi" -I"altia/altiaGL/fb" -I"altia/common" -I"altia/altiaGL/windib" -W3 altia\altiaAPICbkfloat.obj
altiaAPICbk.c
del altiaAPIlib.lib
Could Not Find C:\Users\Guest\Documents\MATLAB\altiaAPIlib.lib
del altiaAPIlibfloat.lib
Could Not Find C:\Users\Guest\Documents\MATLAB\altiaAPIlibfloat.lib
lib -out:altiaAPIlibfloat.lib altia\altiaAPIfloat.obj altia\altiaAPICbkfloat.obj
Microsoft (R) Library Manager Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

lib -out:altiaAPIlib.lib altia\altiaAPI.obj altia\altiaAPICbk.obj
Microsoft (R) Library Manager Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

link /INCREMENTAL:NO /NOLOGO -subsystem:windows,5.0 /NODEFAULTLIB:libc.lib -out:"AltiaDesign.exe" altiaUserMain.obj altia_music.lib altia3D_mesh.lib altia_live_video.lib altia_play_avi.lib altiaWinLib.lib "altiaAPIlibfloat.lib" wsck32.lib kernel32.lib ws2_32.lib mswsock.lib advapi32.lib user32.lib gdi32.lib comdlg32.lib winspool.lib user32.lib gdi32.lib winmm.lib altiaAPIlibfloat.lib
Done.
  
```

Figure 14 - Example Successful DeepScreen Generated Code Compile

10. Please note that a side effect of the **Make Standalone** is an executable `AltiaDesign.exe`. It is just an executable for the DeepScreen generated code, not any Simulink model generated code. Even so, it will execute. When it runs, it displays the Altia GUI window. You can move the slider handle, but it does not move the meter needle because the Simulink model generated code is missing. It is a good test to run `AltiaDesign.exe`. It demonstrates that the Altia GUI object libraries do indeed successfully link into an executable. You can run the `AltiaDesign.exe` easily by pressing the **Run on PC** option in the Code Generation ribbon.
11. You can close Altia Design at this point. We will return to MATLAB and Simulink to complete the build of a fully functional `tutor.exe` executable.

12.3 Use MATLAB and Simulink to Build a Fully Functional Executable

1. At the MATLAB Command Window prompt, type:

```
deepscreenRTWsetup
```

When prompted, select the option number **1**. This MATLAB script will convert the Altia block for use with DeepScreen. It goes under the mask of the Altia block and changes the **S-function module:** section to reference the `altiaWinlib.lib` and `altiaAPIlib.lib` object libraries previously compiled. It also copies necessary files to the `<Model_Name>_grt_rtw` folder. RTW is an acronym for Real-Time Workshop. In earlier versions of Simulink, the C code generation feature was called Real-Time Workshop and this is how the `deepscreenRTWsetup` script got its unusual name.

2. After the Altia blocks are modified and the necessary files have been copied, you will see the following message in the MATLAB Command Window: Altia blocks modified for DeepScreen Windows use.
3. If you already did a Simulink code generation build prior to `deepscreenRTWsetup` changing the **S-function module:** field, it may be necessary to delete the `buildInfo.mat` file from the `<Model_Name>_grt_rtw` folder. Altia's experience is that some releases, such as R2013b or newer, do not recognize the Simulink model has changed when only the **S-function module:** field is changed. By deleting the `buildInfo.mat` file, the Simulink code generation build regenerates all source code files.
4. Go to the Simulink **tutor** model window. Make sure that the **Simulation > Model Configuration Parameters** Simulation time and Solver options are set appropriately. The Solver type must be **Fixed-step**. If you want the program to run until it is closed, set the Simulation time **Stop time:** to `inf` (just type `inf` into the field).
5. In the Simulink **tutor** model window, do **Code > C/C++ Code > Build Model**.
6. If the build in the previous step is successful, a new executable `tutor.exe` is created in the same folder as the Simulink `tutor.slx` file (or `tutor.mdl` file in older releases). Run the `tutor.exe` executable from a Windows Explorer window just as you would run any Windows executable program. Move the slider and observe that the meter needle shows a value 80% of the slider's value. If you close the Altia GUI window (from the **X** button in its top-right corner), the executable still continues to run without the Altia GUI. Press the **X** button in the top-right corner of the executable Command Prompt window to stop the executable. If you stop the executable (by pressing the **X** button for the Command Prompt window) when the Altia GUI window is showing, the Altia GUI window closes automatically.
7. To run the executable on another Windows computer, just copy the executable and the associated Altia `.mxc` file to a folder on the other computer and start the executable. For this tutorial, the required files are:

```
tutor.exe
AltiaDesign.mxc
```

The `.mxc` file is opened by the C code generated for the Altia block in Simulink. It provides the mapping information between the pin numbers of the Altia block in Simulink with the external connection names in Altia. If it is missing, the interface between the Altia GUI and the Simulink model code is broken.

13 Tutorial Summary

In this tutorial, we created an Altia Design GUI and a Simulink model from scratch. We used the Connections pane in Altia to connect the GUI to the Simulink model. We ran the Simulink model in simulation mode and generated input using our Altia GUI. We also generated code from Simulink to run the simulation as a stand-alone executable interfacing to the Altia GUI displaying in Altia Runtime.

In this tutorial, we also took the existing Altia GUI and, using the incredibly powerful DeepScreen technology, generated graphics C code for this interface with the push of a button. We set up Simulink to use DeepScreen and compiled/linked the graphics code with the simulation code for Windows. This resulted in a single executable version of the Altia GUI + Simulink model code. The Altia .dsn file and Altia Runtime files do not need to be distributed with this executable.

Altia and DeepScreen are registered trademark of Altia, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders. Copyright 2011-2014 by Altia, Inc. All rights reserved.