Altia Design C Code Tutorial

Using Altia to Design an Interface and Connect to a Client Program

1.0 Introduction

Altia Design graphics and user interface software can be used in conjunction with various simulation tools (such as Simulink and Statemate) or external C code. The Altia Design package includes an editor, runtime engine and numerous libraries of components for quickly creating user interfaces.

In addition to using the supplied components to create graphical front ends, the **Altia Design** product allows users to make modified versions of existing components and create their own components in the editor without programming. With these features, a modeler can quickly create a user interface prototype for product simulations that looks and behaves like the product's real user interface.

The goal of this tutorial is to explore the process of creating an Altia Design GUI and connecting it to an external C client program. In this tutorial, we will discuss every step necessary to develop a simple system that exercises this connection.

If you are interested in connecting Altia Design to a particular simulator, you may want to get the tutorial that deals specifically with that connection from our web page (www.altia.com).

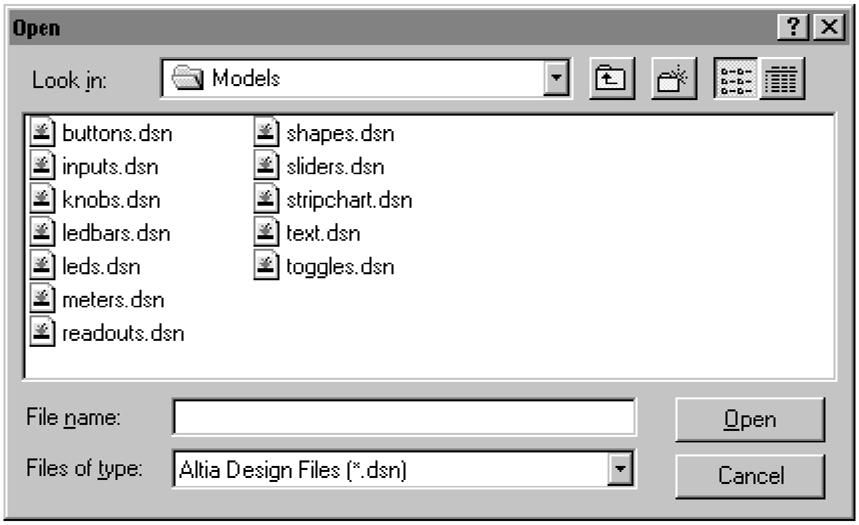
2.0 Tutorial Overview

Before you begin this half-hour tutorial, make sure you have Altia Design installed on your machine. If you need a new copy of Altia Design, please call Altia at **(719) 598-4299** or visit our web site at www.altia.com.

- The first two sections, labeled *Use Altia Design to Create a Simple GUI* and *Creating External Connections For the GUI*, will step through the procedures required to build a virtual front panel to drive and monitor a client program.
- The next section, *Write Your Client Code*, explains a simple C program that can be used to control the GUI built in the first section.
- The final section (*Test Your Altia GUI/Client Code Interaction*) will show you how to verify that your GUI and the executable are connected correctly.

3.0 Use Altia Design to Create a Simple GUI

1. Open the Altia Design editor by choosing its icon from the **Altia Design** program group.
2. In this initial design we will only have a slider bar and an analog meter. To create the slider bar, click on the **Libraries** button which opens the dialog shown in Figure 3-1.



Look in: Models

buttons.dsn
inputs.dsn
knobs.dsn
ledbars.dsn
leds.dsn
meters.dsn
readouts.dsn
shapes.dsn
sliders.dsn
stripchart.dsn
text.dsn
toggles.dsn

File name:

Files of type: Altia Design Files (*.dsn)

FIGURE 3-1 Open Libraries Dialog

3. Choose the **sliders.dsn** library and then click **Open** to view a collection of pre-built slider bars. Pull the slider of your choice into your design by left-clicking on it and dragging it into the Altia Design window.

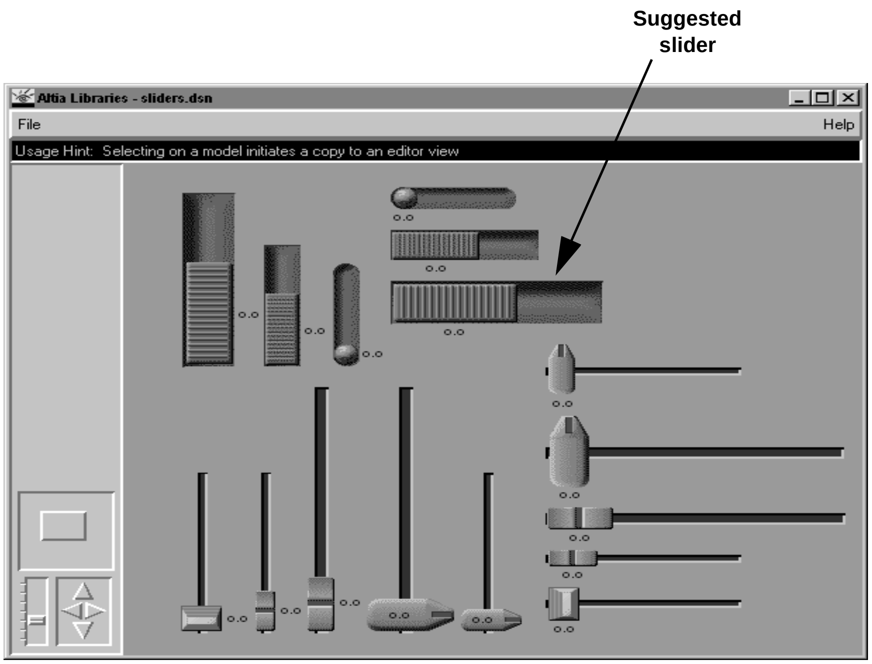
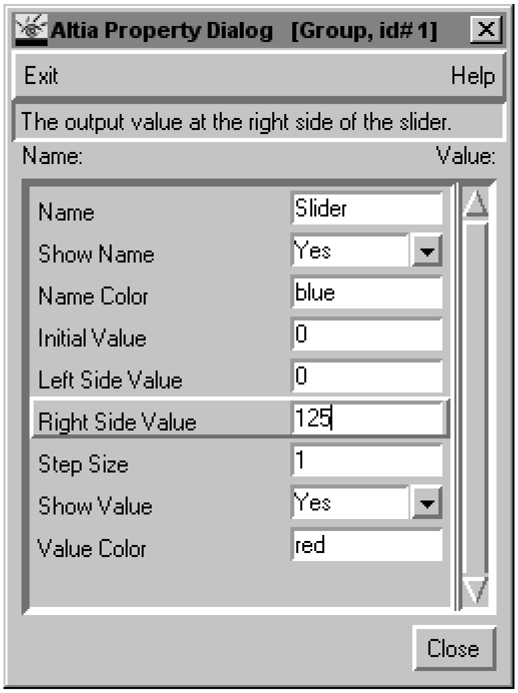

Suggested slider
Altia Libraries - sliders.dsn
File
Help
Usage Hint: Selecting on a model initiates a copy to an editor view
0.0

FIGURE 3-2 Library of Sliders

4. Select the Altia Design main window and double-click the slider to open its **Property Dialog**.

5. Change the **Right Side Value** property from **100** to **125**.

6. Next, change the **Show Name** property drop-down box to **Yes**, the **Name Color** to **blue** and the **Value Color** to **red** (see Figure 3-3).



The output value at the right side of the slider.

Name	Slider
Show Name	Yes
Name Color	blue
Initial Value	0
Left Side Value	0
Right Side Value	125
Step Size	1
Show Value	Yes
Value Color	red

FIGURE 3-3 **Slider Property Dialog Box**

There are several other properties associated with this slider. We won't be altering 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 one in Figure 3-4.

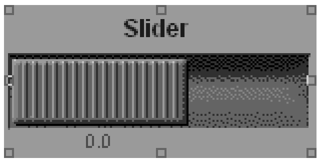
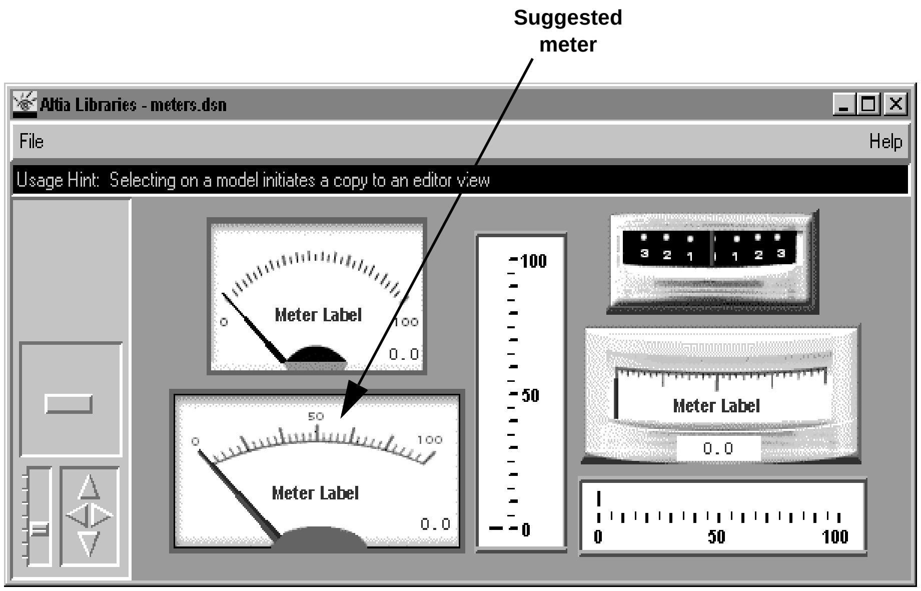


FIGURE 3-4 **Finished Slider**

7. Several analog meters are located in the **meters.dsn** model library. Open this component library by closing the Libraries window and then clicking on the **Libraries** button again. Select **meters.dsn** and click the **Open** button.



Usage Hint: Selecting on a model initiates a copy to an editor view

FIGURE 3-5 Library of Meters

8. Choose an analog meter and drag it into the Altia Design main window just below the slider. After placing the meter in Altia Design, close the Altia Libraries window(s) by choosing **Close** from its **File** menu.

9. Double-click on the meter to open its properties dialog. Change the **Text Label** to **Gain Meter** and the **Value Decimal Digits** property to **2**. The finished meter should look something like the one in Figure 3-6.

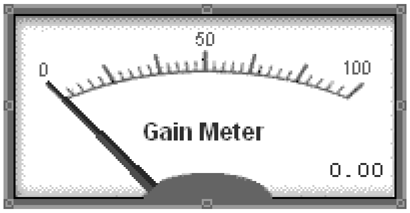


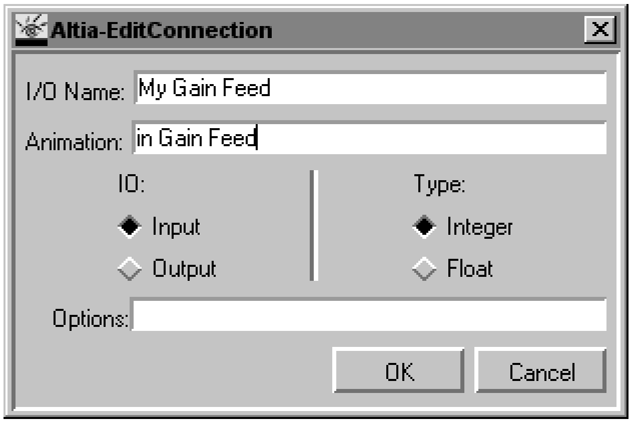
FIGURE 3-6 Finished Meter

10. We are not done with our design just yet, but let's save before moving on to the next section. In the Altia Design editor, choose **Save** from the **File** menu. In the File name box, type **tutorial.dsn** and then press the **Save** button. We have now saved this simple design to the file **tutorial.dsn**.

4.0 Creating External Connections For the GUI

1. The process of sending data from C code to our Altia design is simplified by the use of external signals (or connectors). These connectors offer easily accessible entry points for external programs. Let's add a few connectors to our sample design.
2. From the Altia Design **Connections** menu, choose **External Signals** to open the external signals dialog. Initially, there will be no external signals in the dialog.
3. From the **Edit** menu of the external signals dialog, choose the **Add Connection...** option. This will open a blank **Edit Connection** dialog.
4. In the **I/O Name** field, type **`My Gain Feed`**. In the **Animation** field, type **`in Gain Feed`** (as shown below) then press the **OK** button. The name in the **Animation**

field (**`in Gain Feed`**) is the case-sensitive name we will later use to connect the slider to our C code.

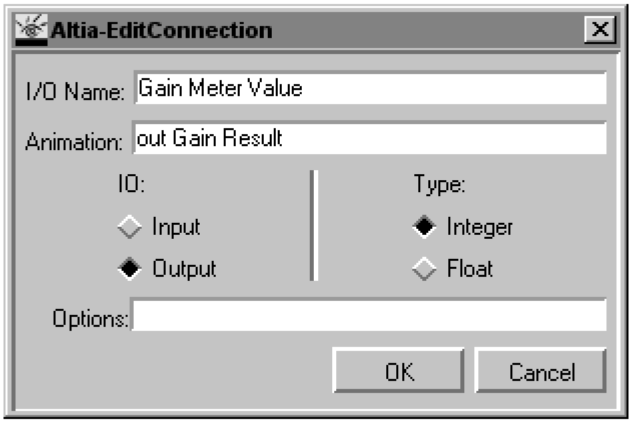


I/O Name: My Gain Feed
Animation: in Gain Feed
IO: [✓ Input] Output
Type: [✓ Integer] Float
Options:

FIGURE 4-7 Gain Feed External Connector

5. Now let's add an output connector to control our meter. From the **Edit** menu of the external signals dialog, choose the **Add Connection...** item.

6. In the **I/O Name** field, type **`Gain Meter Value`**. In the **Animation** field, type **`out Gain Result`**. Click the **Output** radio button and then press the **OK** button. The name in the **Animation** field (**`out Gain Result`**) is the case-sensitive name we will later use to connect our C code to our meter.

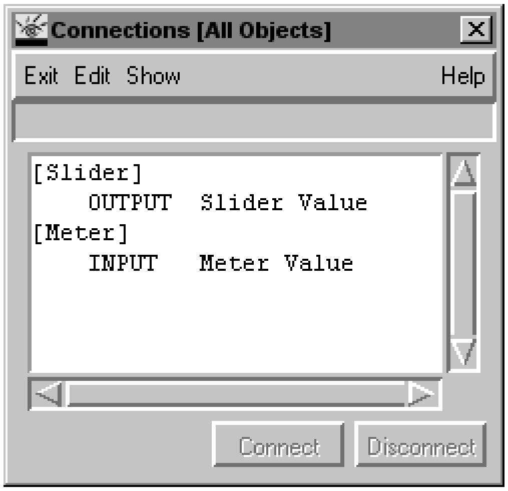


I/O Name: Gain Meter Value
Animation: out Gain Result
IO: Input [✓ Output]
Type: [✓ Integer] Float
Options:

FIGURE 4-8 Gain Result External Connector

7. Now we must connect these external signals that we have built to our Altia objects. This is easily done in Altia.

8. From the Altia Design editor’s **Connections** menu, choose **All Objects**. A window containing the available connections of our two objects will open.



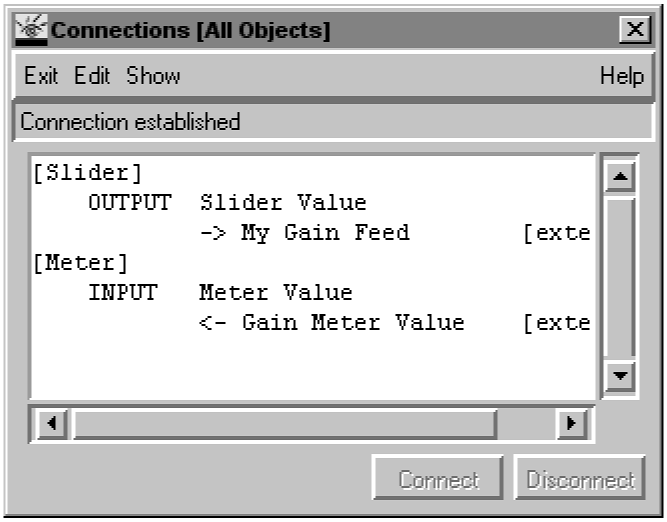
```
[Slider]
    OUTPUT  Slider Value
[Meter]
    INPUT   Meter Value
```

FIGURE 4-9 Connectors of All Altia Objects

9. In the connection dialog that shows all of the objects, click on the signal labeled **OUTPUT Slider Value**, then click on the signal in the external connectors dialog labeled **INPUT My Gain Feed**.

10. When this is done the **Connect** button will become available. Press it to connect the slider to the external signal going to the C code (which we will write in the next section). The connections dialogs should reflect the fact that the two objects are now connected (see Figure 4-10).

11. To connect the output from the C code to the analog meter, click on the signal labeled **OUTPUT Gain Meter Value** in the external connections dialog, then click on the signal labeled **INPUT Meter Value**. Press the **Connect** button to connect the output of the code to the meter.



```
Connection established
[Slider]
    OUTPUT   Slider Value
             -> My Gain Feed        [exte
[Meter]
    INPUT    Meter Value
             <- Gain Meter Value    [exte
```

FIGURE 4-10 Connection of All Objects

12. Now, let’s save our design again by choosing **Save** from the **File** menu.
13. After saving, close all connections dialogs by choosing **Close All** from the **Exit** menu in one of the connections dialogs (but leave the Altia Design editor open).

    Although the graphics look neat, right now they don’t do much because we haven’t written any code to control them. This is our next task.

5.0 Write Your Client Code

This section contains some simple code that can be used to monitor and control our design. An uncommented version is listed first, but an extensively commented version is also included later on in this section. Reading the code’s comments is a good way to learn how to write C code that is connected to an Altia design.

NOTE: The `altia.h` and `libdde.lib` files referenced in this example come from the `Altia\libfloat\ms32` directory.

Uncommented Version of Code

```
#include <altia.h>
#define SLIDER "in Gain Feed"
#define METER "out Gain Result"
void main (argc, argv)
int argc;
char *argv[];
{
        AltiaEventType tmpVal;
        char *tmpName;

        altiaSelectEvent(SLIDER);

        while (altiaNextEvent(&tmpName, &tmpVal) == 0)
        {
                altiaSendEvent(METER, (AltiaEventType) (tmpVal * 0.8));
        }
}
```

That is all that is required to listen for events from the slider and, when they occur, send a value that is 80% of the slider to the meter. To compile, it is usually handy to use the included sample **`Makefile`** (from the same directory as the header and library files) as a starting point.

Commented Version of Code

```
/* Only two files are required to build this file:
 * altia.h and libdde.lib. Both are located in the
 * Altia\libfloat\ms32 directory.
 *
 * See the Makefile in the same directory for sample
 * build instructions.
 */
#include <altia.h>

/* The following lines define the Animation names
 * specified in the External Signals dialog in
 * Altia. Note that the “in”/”out” labels are with
 * respect to the C code.
 */
#define SLIDER "in Gain Feed"
#define METER "out Gain Result"

void main (argc, argv)
int argc;
char *argv[];
{
    /* AltiaEventType is a special variable type
     * that the API uses for Events to/from Altia.
     * When using the float library (libfloat),
     * AltiaEventType is a double.
     *
     * tmpVal will hold the value returned by the
     * altiaNextEvent() call. For this program, that
     * will always be the value of the slider.
     */
    AltiaEventType tmpVal;

    /* tmpName will hold the name of the incoming
     * event. Since we are only listening for the
     * slider, we don't really care about the name.
     * However, if we were listening to many signals,
     * it would be quite handy.
     */
    char *tmpName;
```

```
        /* The altiaSelectEvent() call specifies what events
         * in Altia we care about. For this program, we only
         * want to listen for "in Gain Feed" events (i.e., SLIDER).
         */
        altiaSelectEvent(SLIDER);

        /* The altiaNextEvent() call will wait until an event
         * that has been selected (just "in Gain Feed" in our
         * case) occurs or a communication failure occurs. In
         * the former case it returns a 0, in the latter, a -1.
         *
         * When a SLIDER event occurs, the animation name
         * ("in Gain Feed") is pointed to by tmpName and the
         * animation value is given to tmpVal.
         *
         * There are smarter ways to implement code that watches
         * for events, but we'll save those for the sequel. This
         * is probably the simplest approach there is. Even
         * though it is an infinite loop, the altiaNextEvent() call
         * actually waits (using no CPU) for new events. We can always
         * kill the loop by using Altia's
         * Client->Disconnect All Clients menu option
         * or just by closing Altia (which will cause the
         * aforementioned communication failure).
         */
        while (altiaNextEvent(&tmpName, &tmpVal) == 0)
        {
              /* The altiaSendEvent() call just sends an event
               * to the specified animation name. In our case,
               * we send 0.8 times the value of SLIDER ("in Gain Feed")
               * to METER ("out Gain Result"). Easy, huh?
               */
              altiaSendEvent(METER, (AltiaEventType) (tmpVal * 0.8));
        }
    }
```

6.0 Test Your Altia GUI/Client Code Interaction

1. After creating an executable file from the program in the previous section, go back to the Altia Design window and choose **Start Client...** from the **Client** menu. In the dialog that opens, find and select your executable then press **Open**.
2. To interact with the client code that is now running, all we have to do is put Altia Design in Run mode so that our actions are perceived to be interactions with the objects (and not attempts to edit them).
3. Select the Altia Design main window and press the **Run** button directly beneath the **Object** menu. Click on the slider with your mouse and drag it from side to side.
4. Notice how the meter displays a value that is 80% of the slider's current value. Wowie.

   A really cool feature of Altia is that it is very easy to change the properties of the objects on our virtual front panel *while the simulation or code is still running*. Switch Altia Design back into Edit mode and give it a try.
5. When you are ready to stop the client code, choose **Disconnect All Clients** from the Altia Design **Clients** menu.

7.0 Tutorial Summary

In this tutorial, we have created an Altia GUI using libraries of pre-built components. We used the connection dialogs in Altia Design to create external connectors and then connected them to our GUI. We then wrote a simple client program. Finally, we ran the client code and stimulated it using our Altia interface.