

Custom Instrument Tutorial - Hooking the Script up to the GUI

We have our designed instrument panel. We have our DelphiScript code to achieve the desired manipulation of the output. We now need to hook our script up to the GUI, so that the relevant procedure can be called when a certain event occurs within the panel.

Each component/control has a number of events associated with it. These are the events that the component/control can 'react' to. The script code that gets used when an event is 'fired' is that event's handling code. The parent procedure or function written in the script is therefore the 'event handler'. The logical place then, to link event to event handler, is the value of the event – defined on the **Events** panel, when the **Design** tab of the dialog is active.

When monitoring inputs or controlling outputs – wired to the Custom Instrument – we are dealing with not just a single control object on the instrument's panel, but rather the entire panel itself. The event we are interested in, and to which we will link our code to write values onto the `Data_Out` line, is the **OnReadWrite** event.

The panel's **OnReadWrite** event is hard coded to 'fire' each time the instrument polls. The polling interval is 250ms by default, and can be changed in the *Custom Instrument - Options* dialog (accessed by clicking on the **Options** button included on the panel).

IO values are updated by nature of the polling, not by the **OnReadWrite** event. The latter is simply a method to synchronously call code which can process the signal values at each poll point.

Each time the instrument polls, the current value for each input signal is read into an internal storage structure. Similarly, the current value internally stored for each output signal is made available at the corresponding output pin of the instrument. The internal storage structure is transparent to the user and, as such, cannot be accessed in any way. The event handling code linked to the **OnReadWrite** event is essentially used to process the IO as required.

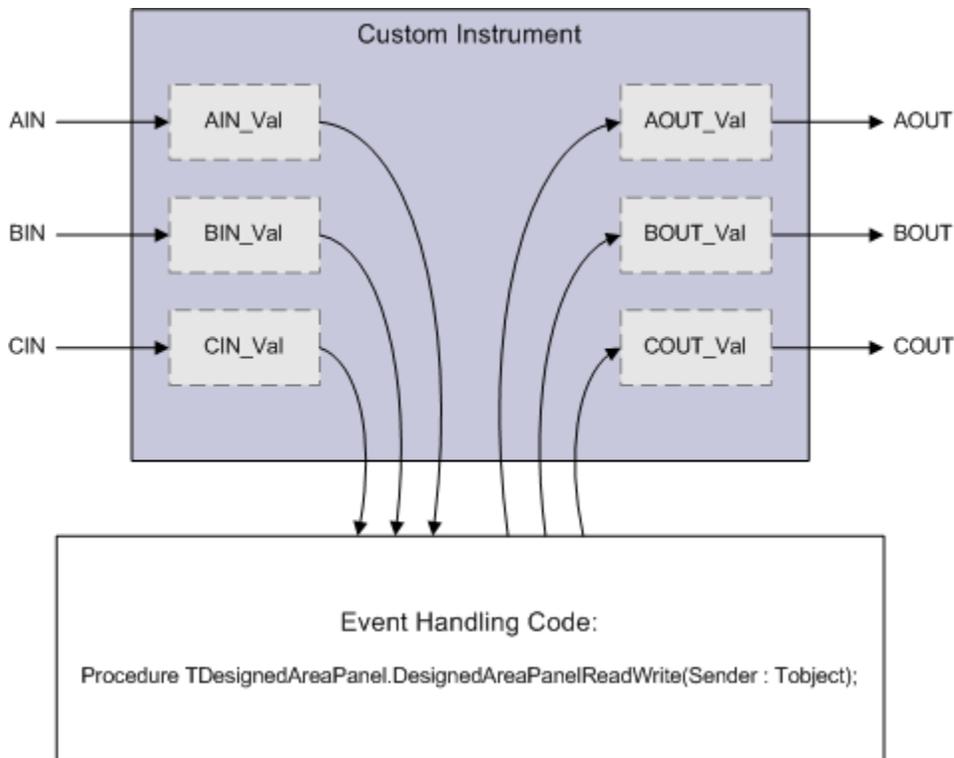


Figure 1. Use event handling code linked to the panel's *OnReadWrite* event to process signal IO each time the instrument polls.

⚠ If polling is turned off (by setting the **Update** value in the *Custom Instrument - Options* dialog to 0 ms), the **OnReadWrite** event will not fire. It will, however, fire if the **Synchronize** button is available on the instrument's panel, and is pressed. This would essentially be a 'manual poll'.

Let's go ahead and hook up our two scripting procedures to the relevant objects and events.

1. Open the *Custom Instrument Configuration* dialog (if not already) and make the **Design** tab active.
2. Click anywhere within the bounds of the panel, away from controls, to select the `DesignedAreaPanel` object.
3. From the associated **Events** panel, click inside the field to the right of the **OnReadWrite** event and use the drop-down to choose (and assign) the `DesignedAreaPanelReadWrite` procedure.
4. Now click to select the **SOFTWARE OVERRIDE** Button control.
5. In the associated **Events** panel, click inside the field to the right of the **OnClick** event and use the drop-down to choose (and assign) the `Panel_Override_ButtonClick` procedure.

That's it. We have completely finished configuring our custom instrument. Time to [target](#) and [program](#) the physical FPGA device with our design, so that we can see it in action!