How to use the UDE templates

The purpose of this file is to demonstrate the UDE templates by implementing the event that my version of the producer/consumer design pattern needs to pass data from the producer loop to the consumer loop.

Setting up a project

The first step in this process is to setup the project to hold our work. Because we will be coming back to this example many times to demonstrate many things, I think a good name for it would be "testbed". Going to your projects directory (mine is at *c:/workspace/projects*), create a subdirectory named *testbed* and then, using LabVIEW, create a project file inside it named *testbed.proj*. Also create a VI called *testbed.vi* and copy to its block diagram the code snippet from the first blog post. The result of this work is project containing a VI implementing the basic UDE-based producer/consumer design pattern. By convention, the project, the VI and the directory containing them, have the same base name.

Finally, inside the home directory for the testbed project (*c:/workspace/projects/testbed*) create a directory to hold all the UDEs the project might need called *_ude*. Note the leading underscore character. It is not a misprint, can you guess why it is there?

Using the UDE template

With this setup done, the next step is (if you have not done so already) to grab a copy of the *UDE.zip* file from the subversion repository at:

http://svn.NotaTameLion.com/blogProject/ude_template

When you open this archive, you'll also notice that it contains the 5 event files (4 VIs and 1 ctl) that you will need to create any UDE. You should also be aware that the process I'm about to describe will take a lot longer to describe than it will to do it. Trust me, once you have done it the first time, the whole process only takes a couple minutes.

Step 1 – Setup the Directory

Inside the *_ude* directory, create the directory to house the event. Give it the name you want the event to have – in this case, *Pass Data*.

Step 2 – Setup the Library

Use LabVIEW to create a library inside the *Pass Data* directory with the name *Pass Data.lvlib*. Also add the library to the project. (Beginning to see a pattern here?)

Step 3 – Copy the Template Files

With the archive containing the event template files open, drag a copy of all the files into the event directory. Afterwards, be sure to close the archive. Remember the reason for having these files in an archive is to protect them from cross-linking. Don't extract them unless you are going to be using them right away.

Step 4 – Install Templates Files

Open the library file (*Pass Data.lvlib*) and drag the files from the event directory into the library window. Before saving your work and continuing on, be sure to update the library icon and apply any changes in the icon to the files in the library.

Step 5 – Customize the Template Files

Make the following changes to the indicated files:

Event Data.ctl – If needed, change this typedef to hold the data you want the event to pass. If the event is going to be used simply as a trigger, you can leave this file as it is. In this case though, replace the variant control with a double-precision numeric.

Get Event Reference.vi – Change the label of the event reference indicator to what you want the event name to be. In this case, change it to *Pass Data*.

Register for Event.vi – On the block diagram you will notice a broken wire. This wire is broken to remind you that you need to create an indicator connected to this wire, and associate that new indicator with the top-right terminal on the connector pane. While you're at it, change the label to something short. Since this is an event that will be sourcing data coming from the DAQ subsystem of our testbed application, let's give it the label *daq*.

Generate Event.vi – This event will be passing a measured value. You'll notice that the event data control on the front panel has already changed to a double-precision float, so all you have to do is associate the control with the top-left terminal on the connector pane.

Destroy Event.vi – No modifications needed.

Step 6 – Using the Event

Typically you will be designing the new UDE into your application from the get-go, but in this case we are replacing existing logic. So first delete the *Create User Event* and *Register for Event* nodes to the left of the consumer loop and install an instance of You will also have to edit the events for case 0 of the event structure and associate it with the *Pass Data* event. Press Ctrl-B to remove the bad wires generated in this step.

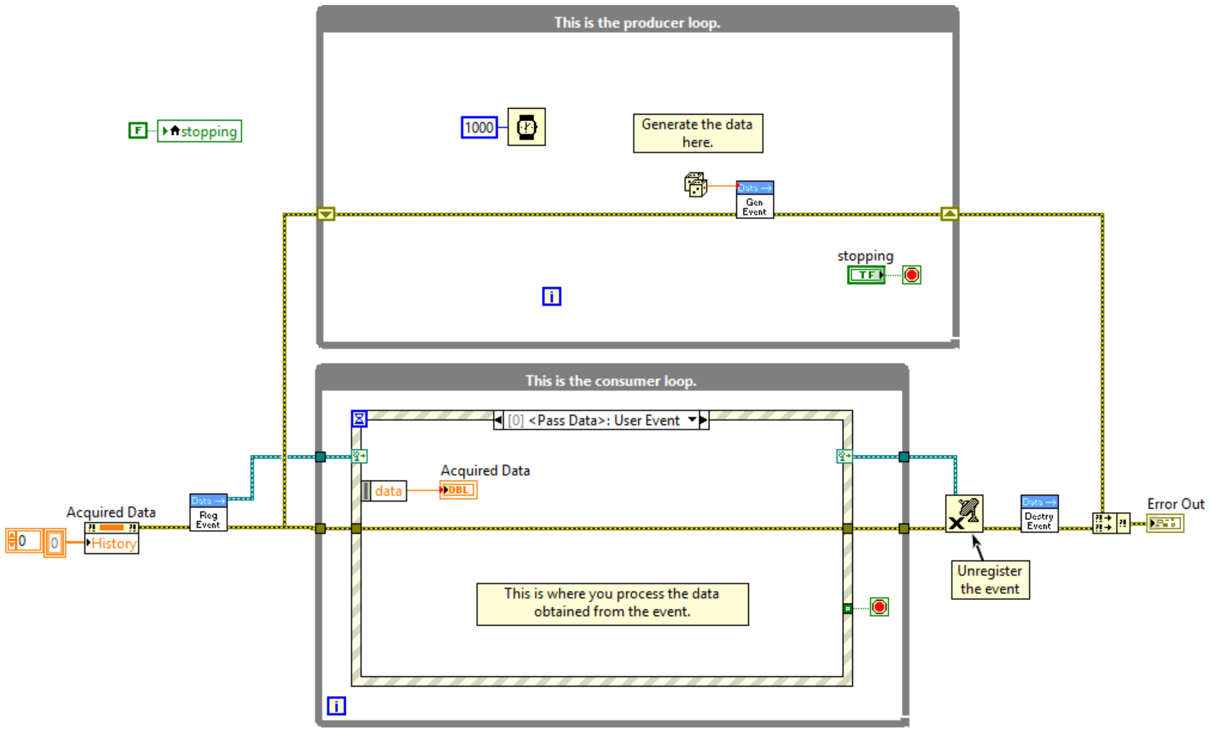
In the producer loop, replace the Generate User Event node with an instance of *Pass Data:Register for Event.vi*. Next delete the string formatting logic and substitute a random number generator as the event data source. Likewise remove the case structure (leaving just the contents of the true case) and add a Wait (ms) node to the loop with a 1000-msec delay value. Press Ctrl-B to remove the bad wires generated in this step.

To the right of the consumer loop, replace the *Destroy User Event* node with an instance of *Pass Data:Destroy Event.vi*.

Finally, on the front panel replace the *Acquired Data* string indicator with a strip chart and rewire the indicator's terminal to the *Pass Data* event data. Also reinitialize the new strip chart by writing an empty array of double-precision floats to the indicator's *History* property.

Checking Out Work

When finished with these modifications, the code should look like this:



When you run it, random data should start to appear on the strip chart at a 1 second rate and continue until the stop button is pressed. This is the basic functionality that we will be fine-tuning in future posts.