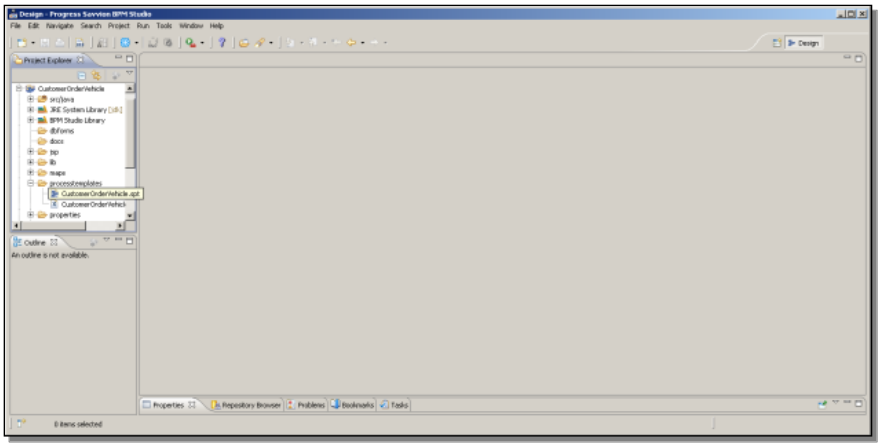
BUILDING AND USING WEB APPLICATIONS

John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
July 2010

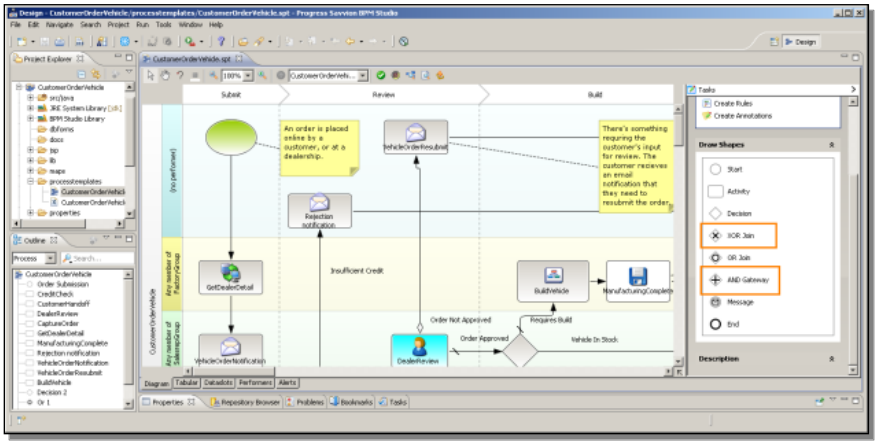
DISCLAIMER

This document accompanies a two-part video in a series on building business process applications using OpenEdge Business Process Management (BPM). In other sessions I've walked through parts of the sequence of forms that the customer uses to order a car in the AutoEdge Factory sample application. In this paper I'll show you a bit about the type of process that sequence represents, called a **Flow** or a **Web Application**. I'll also show you how to create a new one.

First I open the main BPM process for the Factory application again, called **CustomerOrderVehicle**. Within its project, I find it under the **processtemplates** folder. And it has a suffix of **spt**, for **Savvion Process Template**:

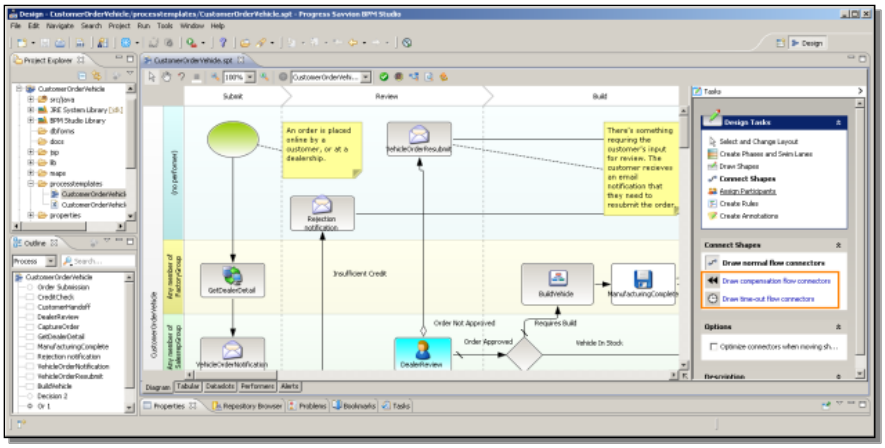


This is the main process diagram for the **AutoEdge | The Factory** sample. You've seen in other sessions that it has swimlanes representing different parts of the overall process performed by different types of users. There are various types of steps executed by non-human users as well, like email messages and web services. And the paths through the process can include complicated routes that represent multiple parts of the job being done in parallel. If I open up all the BPMN diagramming shapes available to me, and scroll down to see them all, you can see the complete set.

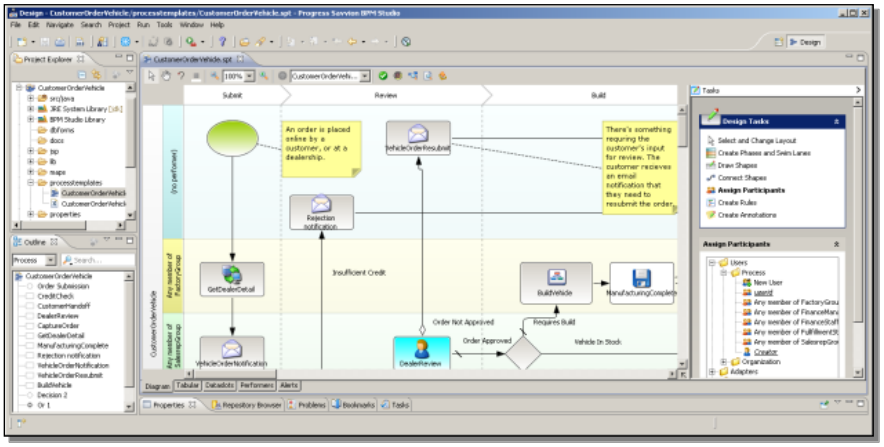


I want to point out a couple of shapes that can be used in complex process diagrams like this one. The **AND** gateway allows you to define multiple paths that execute in parallel, and that can all come together again when they've all completed. The **XOR** join shape allows you to specify that only one of multiple paths completes before the process continues. These types of paths are possible because a BizLogic process can have multiple tasks going on at the same time.

Next I select the **Connect Shapes** task type. In addition to normal arrow connectors, you can define compensation flows for rollback operations, and timeouts for certain paths as well:



Now under the participants for the process, you can take a look at all the process user types defined for the sample, and see the list that includes all of the performer types identified in the various swimlanes:

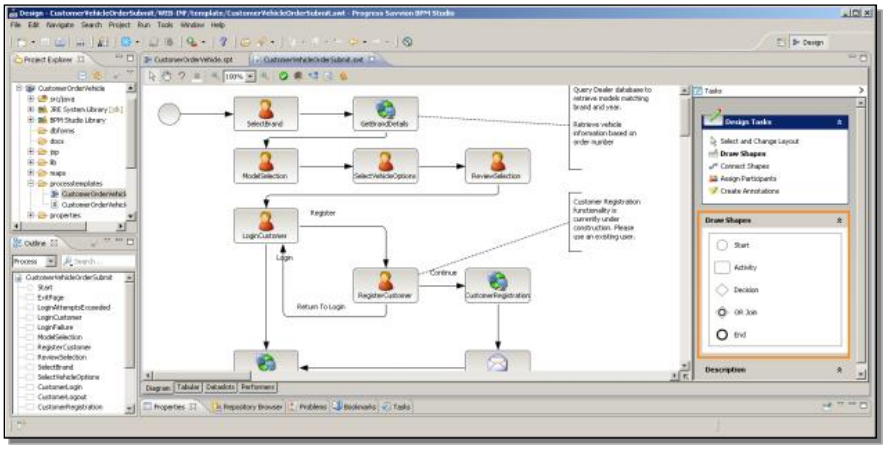


These are all notations that are available for a standard BPM process, what Savvion calls a **BizLogic** process.

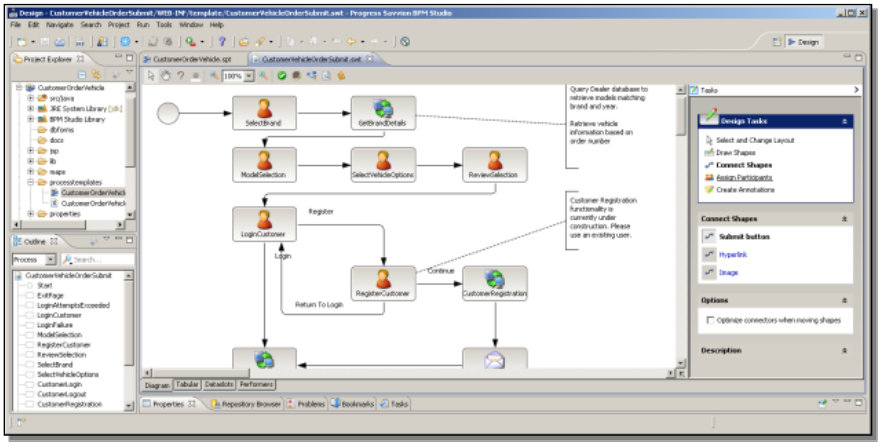
If I now look at the **Start** step for the vehicle order process, I see that in place of a single form is something called a **Flow** with a **bizsolo** suffix. You can set the type of a start step or an activity step to be a process of this type in the step's properties. **CustomerVehicleOrderSubmit** is actually a separate process. It's called a bizsolo process because it's intended to be executed by a single user, in a Web browser, which is why it can be identified as a **Web application**, and why it has the **swt** suffix, for **Savvion Web Template**. And it is also referred to as a **presentation flow** because it represents the visible flow from one web form to the next, in a prescribed sequence.

Let me try to make its characteristics clear by comparing what **Design Tasks** are available to this diagram compared with the BizLogic process. First, there's no Design Task category for swimlanes and phases at all. You don't need swimlanes if there's only one user.

Selecting **Draw Shapes**, you see that only the shapes shown below are available to you within the Flow. There's no AND gateway or XOR gateway because you can't have multiple paths executing concurrently:



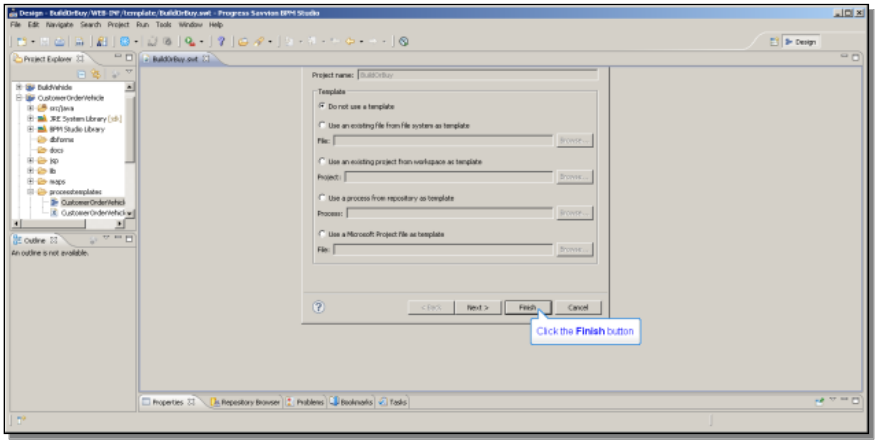
Under **Connect Shapes**, there is a different set of choices. You can connect one step to the next in the form of a **Submit** button, a **link**, or an **image**:



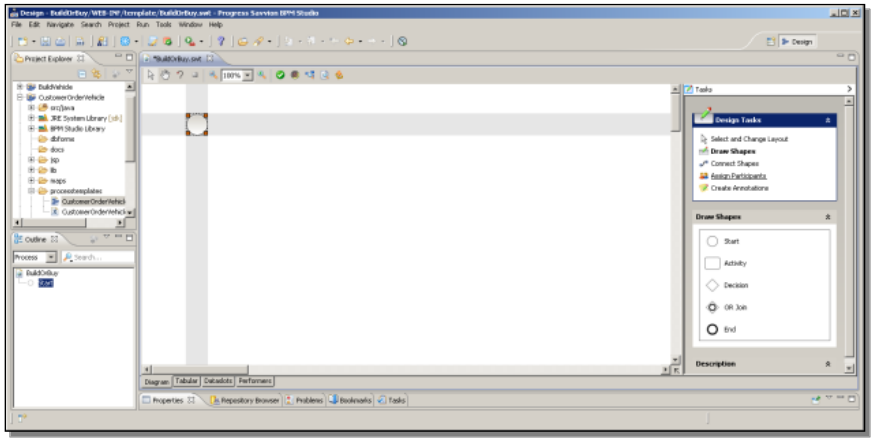
Under **Assign Participants**, there's no **Users** folder, and if I expand **Process**, there's just one generic **User**. All this confirms that the flow is just one user's experience working through the web application. Note that you *can* use **Adapters**, though, such as **Web services**, and in fact in the **AutoEdge | The Factory** Web application there are a number of steps in between user forms that use Web services to call out to OpenEdge to get and submit data, like this one, as was shown in another presentation.

A presentation Flow can be used as a standalone application invoked from a Web browser; I show later in this paper how to build one of these. Or it can be a part of a larger process as this one is. Here the Web application is a specialized kind of sub-process that lets you separate out a particular sequence of steps to be executed by a single user in a web-based user interface. In a follow-on presentation and paper I show you how to do that integration as well.

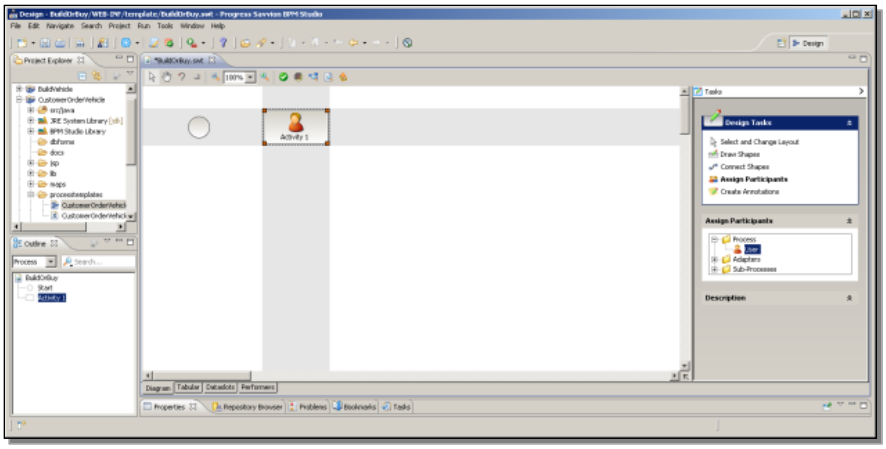
The remainder of this paper shows you how to build a new independent Web application in Savvion. Under **File -> New**, instead of a BPM Project, you can create a **Web application project**. You give it a name like any other project; I call this one **BuildOrBuy**:



That's all I need to specify. Under Draw Shapes, I grab a **Start** step, and drag that onto the design diagram:

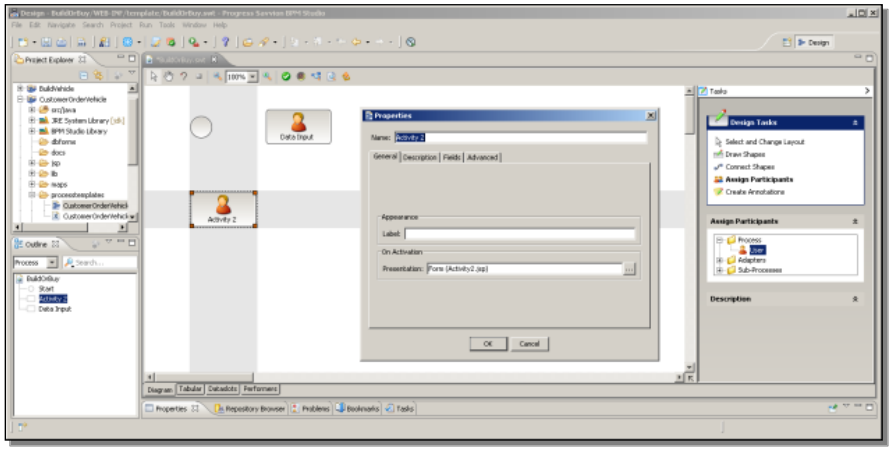


I want some **Activity** steps; remember that they can only have one user. I drag that generic user onto the form to create a user activity:

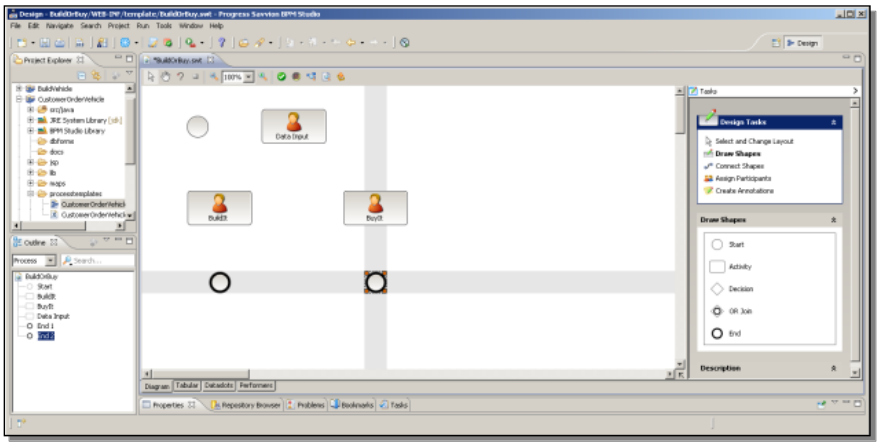


In this first step the user just enters his or her name. That's as much Data as I need in this simple example to make it work. There's a form created for the step whose name needs to be adjusted to match the name of the activity, so I name the Start step **Data Input**.

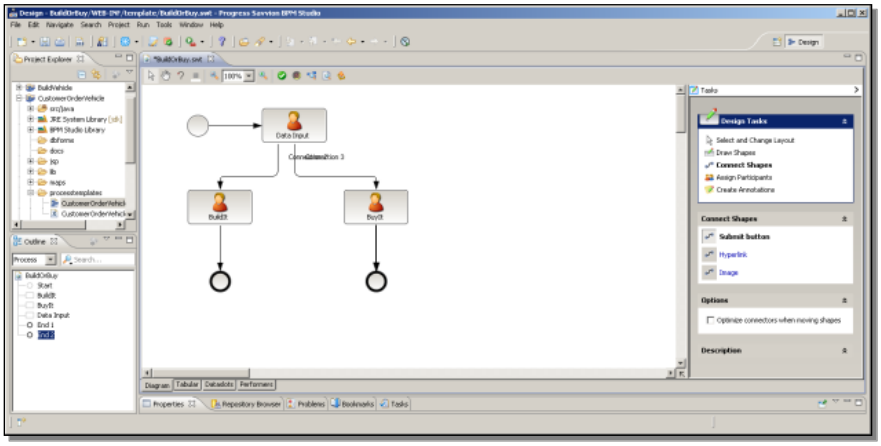
Next I want to show two different paths the user can take from the first Activity. One of them is the request that something be built, so I call this step **BuildIt**:



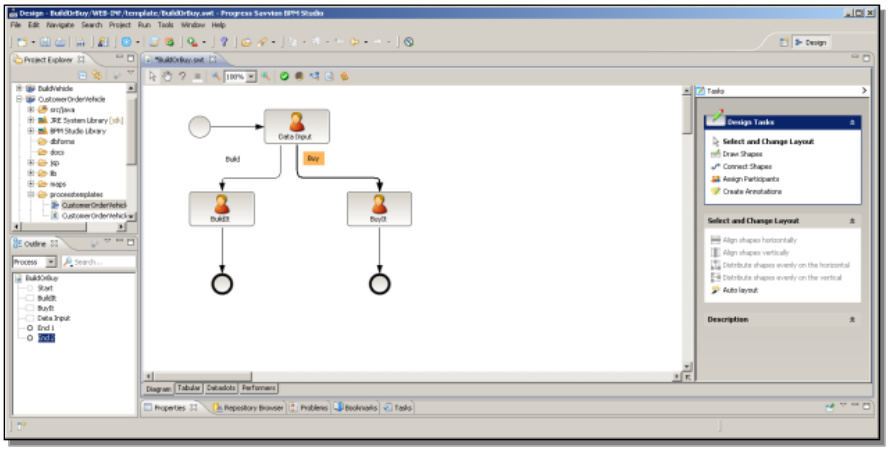
The other path will be to request that the something should be bought instead, so I call that one **BuyIt**. I also need an **End** step for each of those alternatives, one for the Build option, and a second for the Buy option:



Now I need to connect all these shapes together in the sequence they should follow. The default when you select **Connect Shapes** is the same as for a regular process. You draw arrows between shapes to establish a sequence. The Start step just links directly to the Data Input step. That step in turn can go either to the BuildIt Activity or to the BuyIt Activity. The process doesn't do anything else, so BuildIt just goes to an End step, and so does BuyIt. Here all the links are complete:

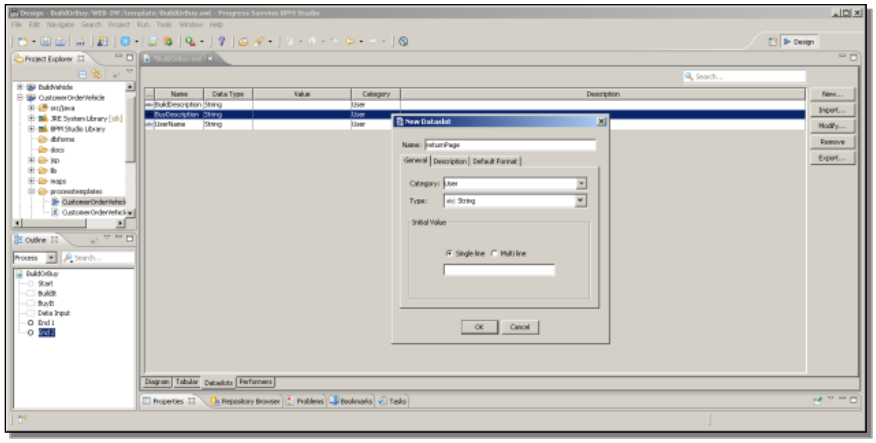


By clicking on **Select and Change Layout** in the **Tasks** pane to the right, I cancel the Connect Shapes activity. Now I can move things around like a connector's label, and importantly, I want to right-click on it to *change* its label. This isn't just to make the diagram look better, but as you'll see in a moment, when I give this link a label of **Build**, that affects how the Data Input step's form looks at runtime. I do the same for the other link that comes out of the Data Input step, and I label this one **Buy**:



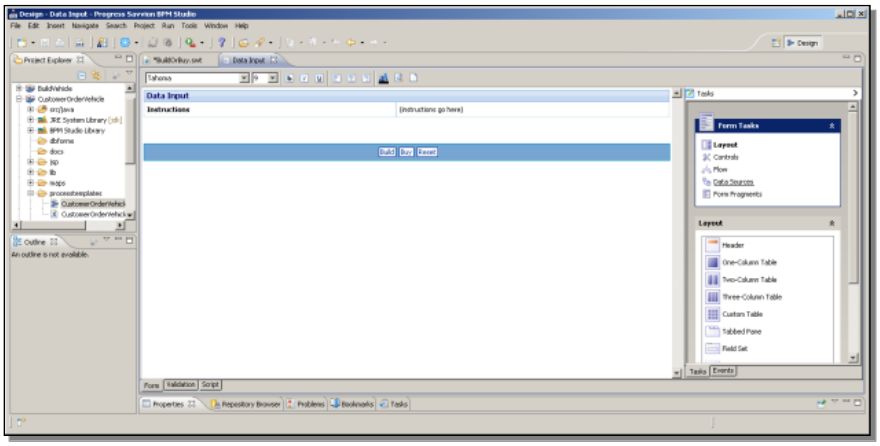
Likewise, I want to name the links from the BuildIt and BuyIt steps to their End steps. The first one I call **Done Building**, and the second one **Done Buying**. The link labels from Acitivity steps to an End step aren't displayed in the diagram, but they still have an effect on how the forms look.

Next I need to define just a couple of Dataslots to have some data for the forms. The Data Input step will just let the user enter a username. The other two forms will display that user name and provide a description field, just so we can tell the Build and Buy forms apart, and confirm which path the process has taken. In addition, there's a special Dataslot you need to define if you want the Web application to go to a specific web page when it completes. Remember this I'm first building this as a standalone Web application; it will be in the next video that I show how to make it part of a larger process. The Dataslot that lets you return the user to another web page must be named **returnPage**, and note how it's camel-cased. Dataslot names are case-sensitive.



I give the Dataslot an initial value that points the user to the Progress Savvion web page – http://www.progress.com/savvion -- when the Flow completes.

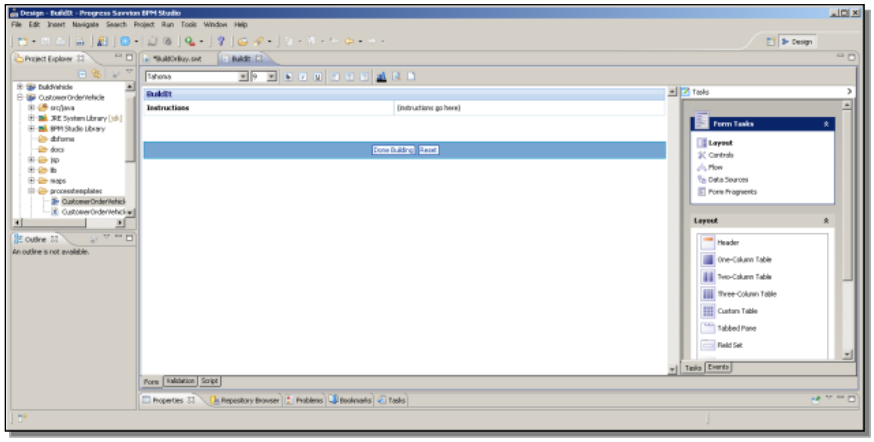
This is enough Dataslots to get some minimal forms built. Now I'll show you what the forms look like when the activities are connected together the way they are. First I open the form for the Data Input step. Because there are two connections coming out of this step, and I labeled them Build and Buy, Savvion has added two buttons to the form's footer labeled Build and Buy:



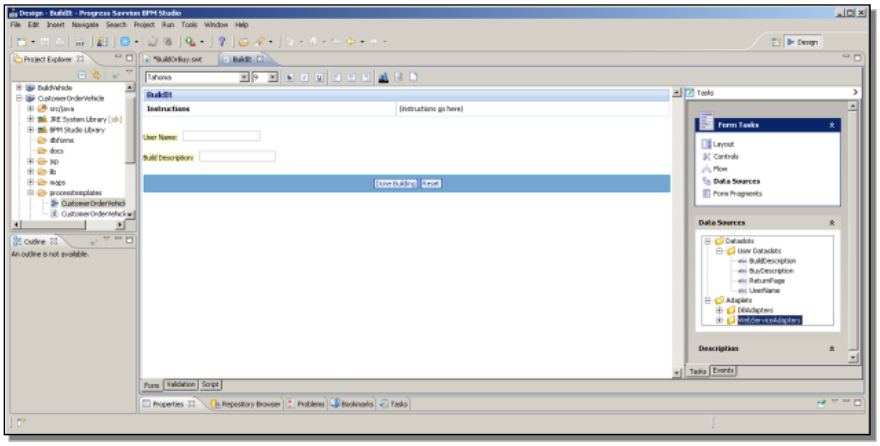
This is why it was significant that I give the connectors meaningful labels. If the user clicks **Build** after filling in the form, the Flow follows the **Build** link to the **BuildIt** step. If the user clicks the **Buy** button, it will follow the **Buy** link to the **BuyIt** step.

I need at least one field in the form so that the user can do something in the form, so I grab one of my Dataslots, the one called **Username**. That's as much as I need to create a rudimentary form.

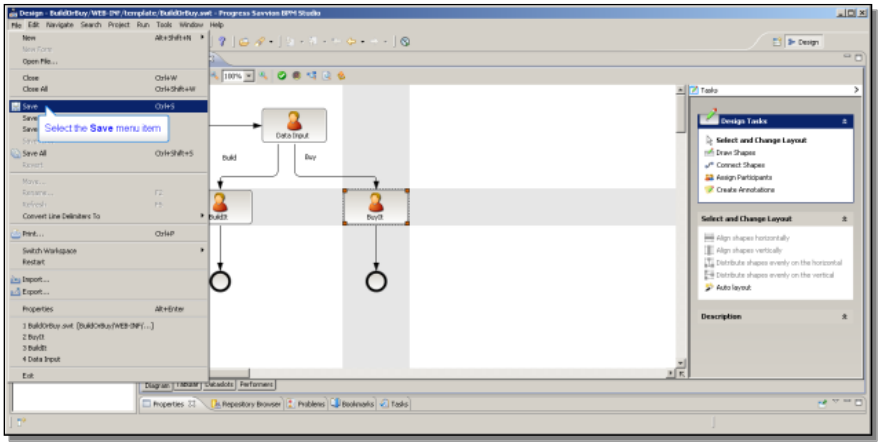
Next I create a form for the **BuildIt** step. Below you can see that the link from this step to the End step has resulted in a **Done Building** button for this form. When the user clicks this button, the Flow is complete:



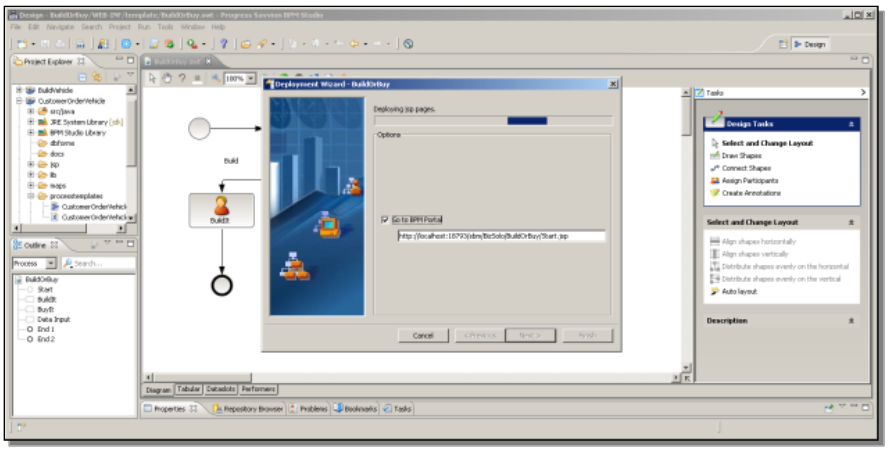
In this case, I grab the **Username** Dataslot again, just so I can confirm it's been set by the previous step. I also add the **BuildDescription** Dataslot, which is what makes this form distinct from the BuyIt form:



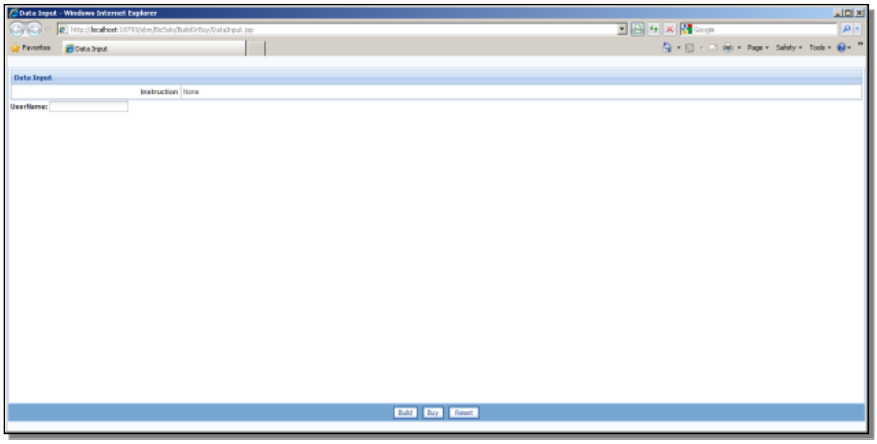
I do exactly the same for the **BuyIt** form. For this form there's a **Done Buying** button that signals the completion of the Flow. I drag the **Username** and the **BuyDescription** Dataslots onto this form, and that's all there is to it. Finally, I need to save the Web application itself:



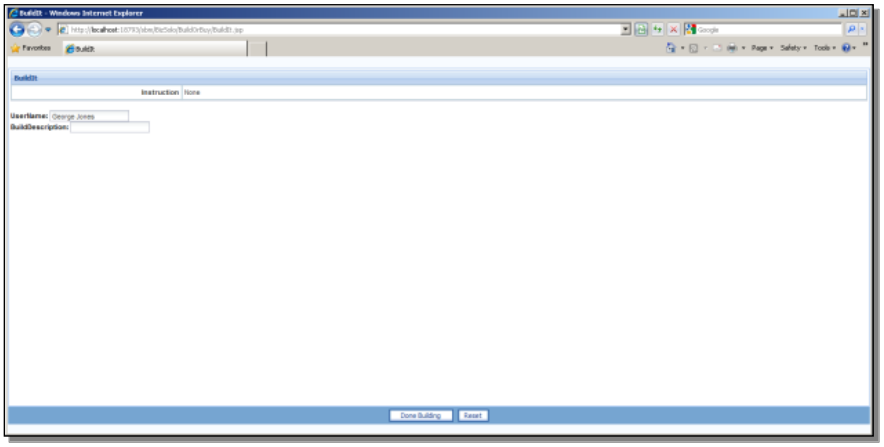
Then I deploy the Web application to the server as I would any other process. In this case, the URL of the Flow's Start step is important to note, because that's what the user will enter in a browser, or link to from somewhere else, to start the Web application:



I copy that URL and Finish the deployment. Next, in a browser, I paste in that link and press return, and the Web application is off and running. Here's the Data Input form, for the step the Start step links to:



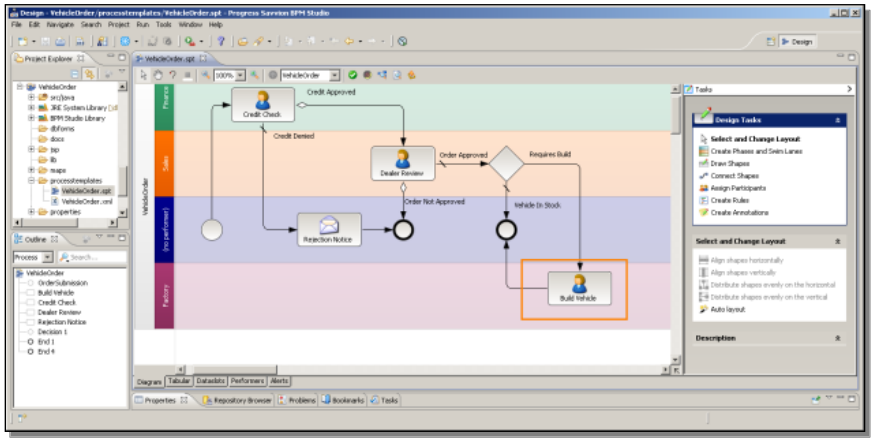
I enter a username, and I can select Build or Buy. I click **Build**. Below you can see that the Flow has followed the Build link to the **BuildIt** step, and here is its form:



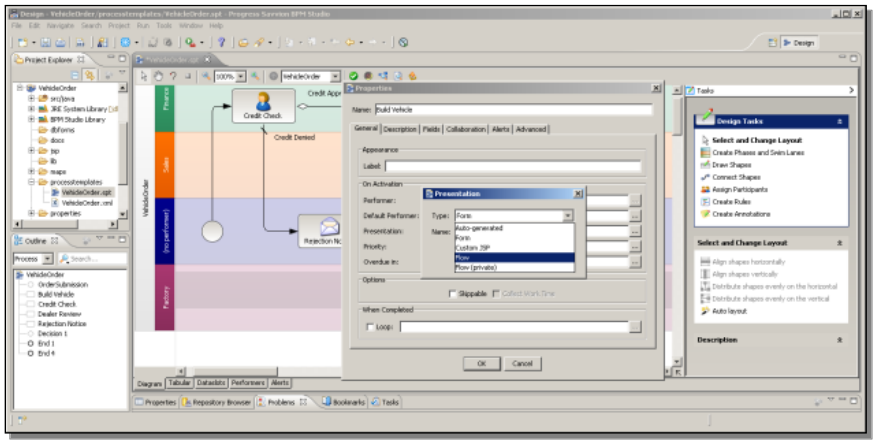
I enter a description field, and click **Done Building**, which ends the Flow. Since the Flow has completed, Savvion looks for the **returnPage** Dataslot and directs the browser at that page:

This completes the first part of the introduction to using Flows, or Web applications, in Progress Savvion. I've showed you an example from the **AutoEdge | The Factory** module of a flow integrated into a larger process. In that case the flow lets you separate out a sub-process that an individual executes, an important part of modularizing a large complex process. I also showed you a simple standalone web application. Even though this is executed by a single individual, BPM Studio still gives you the ability to create a complex decision map that sends the user interface in different directions based on data entered or retrieved in the flow.

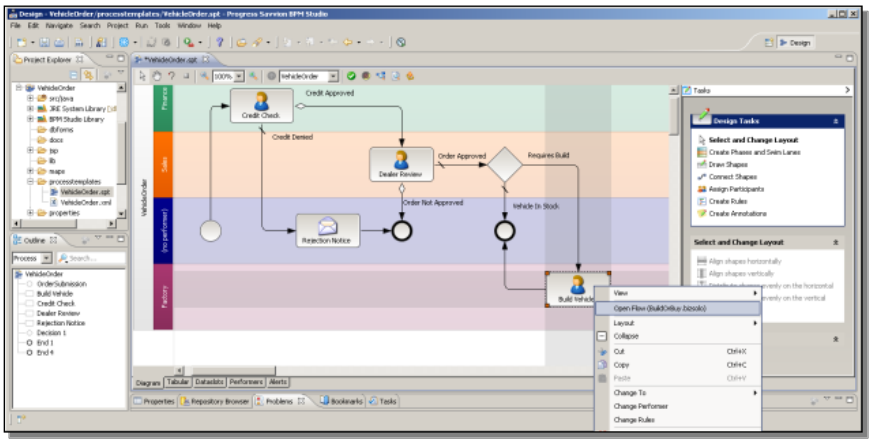
In the next part of this paper I show you how to integrate the BuildOrBuy Flow into a larger process, the sample process that I built in the series on building your first project, the **VehicleOrder** process. You can see that the process has a dummy workstep called **BuildVehicle** that I added just to have something at one end of a decision step. Now I'll give that workstep something to do.



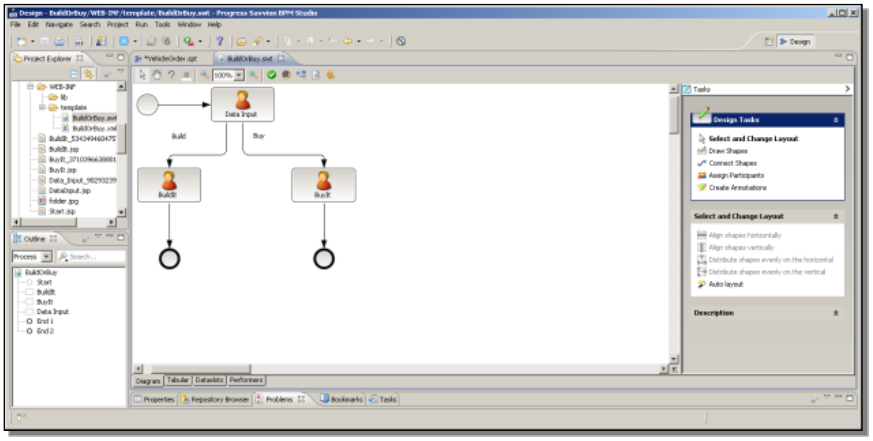
In its properties, I change the presentation **Type** from a form to a **Flow**. This means that when this workstep is reached, it will initiate a Flow as a sub-process:



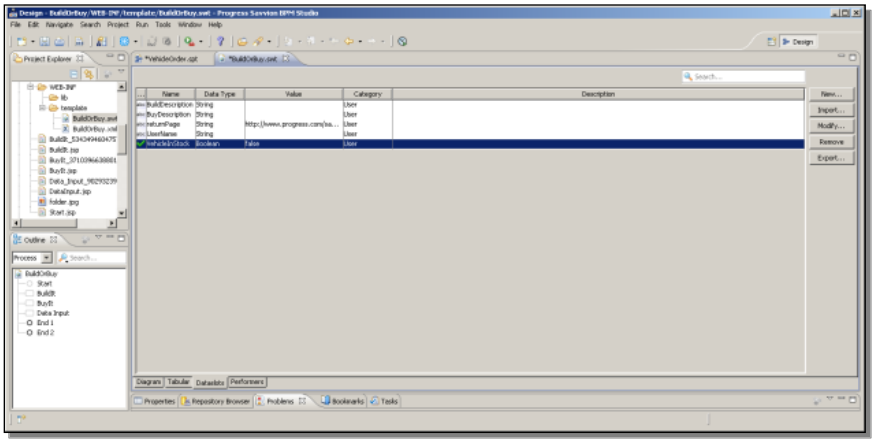
I give the flow the right name, which is the name of the Flow I already have, **BuildOrBuy**, and when I OK that, and go back into the workstep Properties, you can see that the step now knows that its job is to invoke a flow called **BuildOrBuy.bizsolo**:



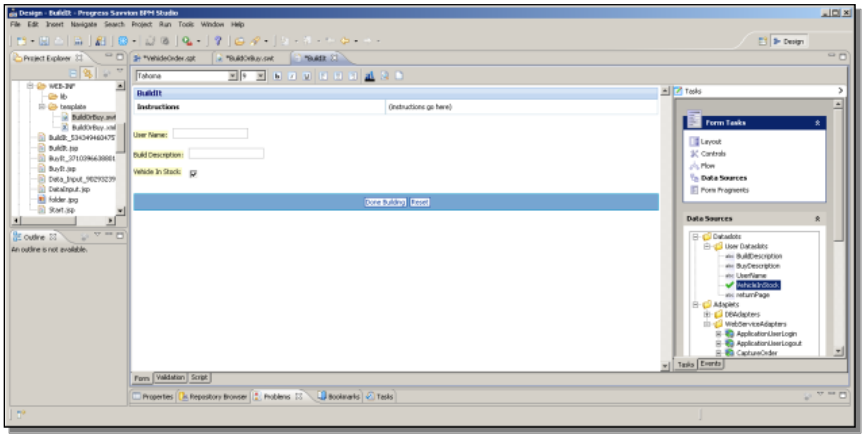
Opening it, you can see that BPM Studio has found the Flow successfully in the project workspace:



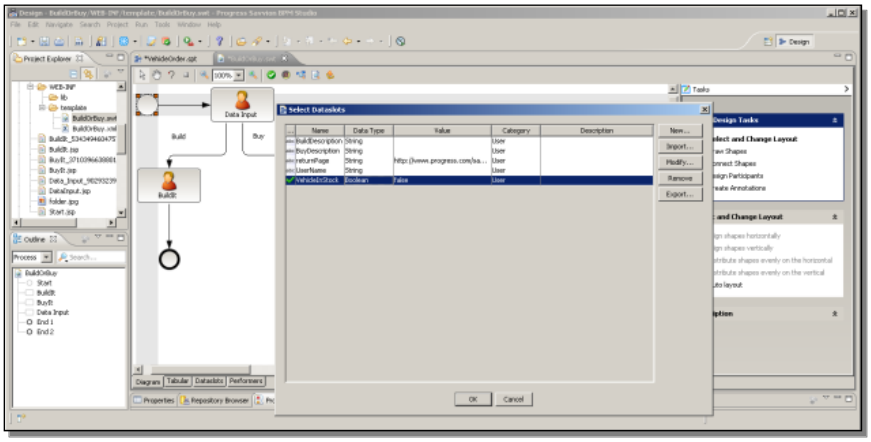
I want to set up just a bit of communication between the flow and its parent process, so I add a Dataslot to the flow. Remember that the Flow is a separate process, used here as a sub-process, so it has its own namespace and its own Dataslots. I create a new boolean dataslot in the Flow called **VehicleInStock**, which is the same name as a Dataslot in the parent. Now I have something I can easily pass back and forth between here and the parent process.



Back in the diagram for the Flow, I want to make a change to a couple of its little forms. **BuildIt** is the step the Flow goes to if the user clicks the **Build** button in the first form. I want to add the **VehicleInStock** dataslot to the form, so it can be set there. I choose it from the Flow's Dataslots, drag it onto the form, and place the checkbox after the label.

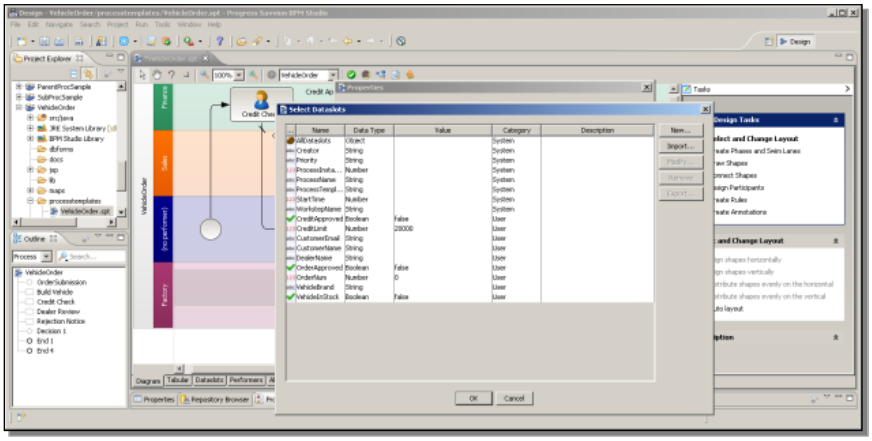


Next I do the same thing to the **BuyIt** step. In order to pass the value of the VehicleInStock dataslot from parent to Flow and back again, I need to do something similar to defining parameters to a regular sub-process, as I showed you in another of these presentations. In the Flow's **Start** step Properties, I identify the dataslot to add to what the Start step expects.This is how you identify a dataslot that will be passed from the parent process to the flow.

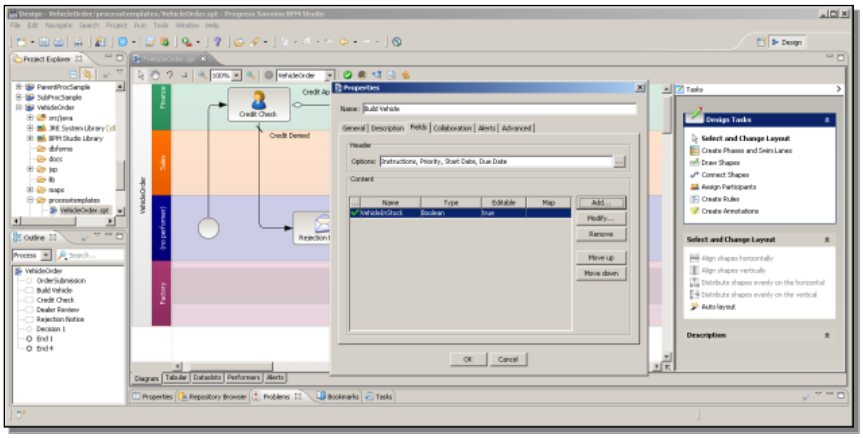


Because the Dataslots have the same name in both processes, I don't need to do any more mapping than this. I'm done with the changes to the Flow itself.

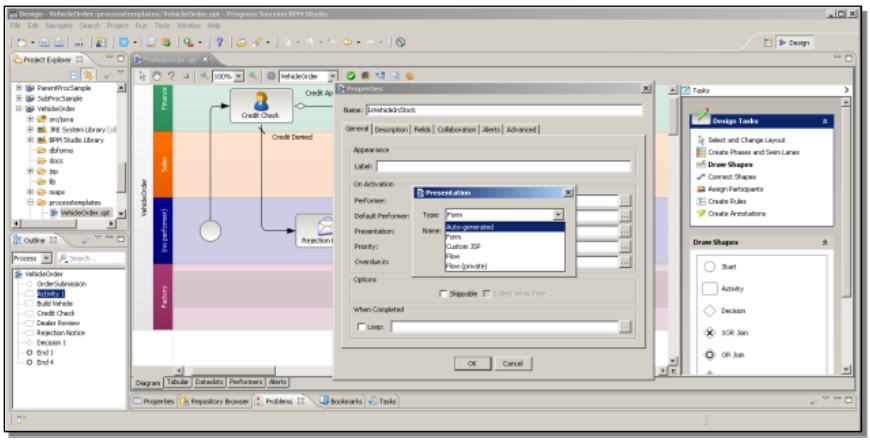
Now in the Parent process, I go into its workstep properties, and I add the Dataslot as a **Field** for this Flow workstep. This tells Savvion to pass the Dataslot as a parameter to the Flow. Again, this is similar to how you pass a Dataslot as a parameter to an ordinary sub-process. And remember that I'm now referencing the **VehicleInStock** Dataslot that's defined in the parent process, **VehicleOrder**. The Flow has its own Dataslot with the same name.



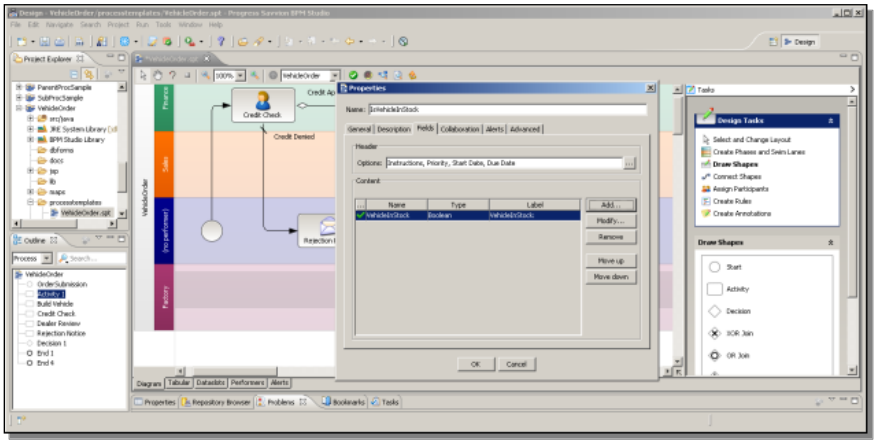
Once again, because the dataslot names match, that's as much as I need to do to get them mapped to one another.



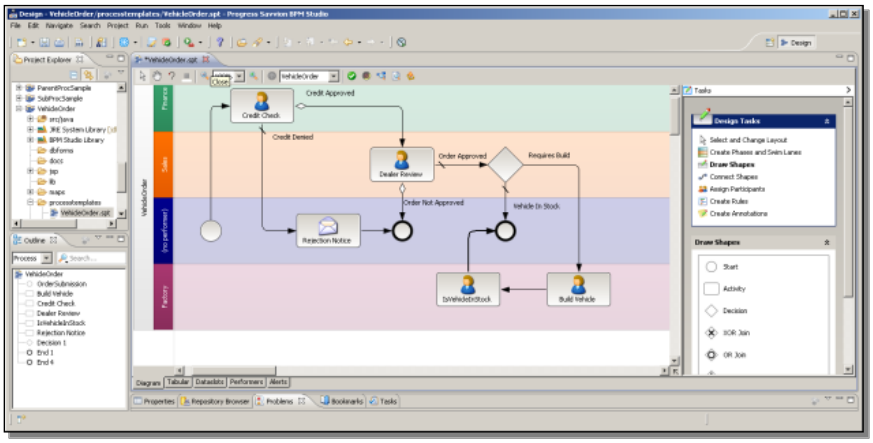
Now just to confirm that the Dataslot value is passed back successfully from the Flow, I drag another **Activity** onto the parent diagram, and name it **IsVehicleInStock**, because its purpose is to display the value of the Dataslot after the Flow completes. I just make this a simple auto-generated form:



The only Dataslot I need to add as a **Field** is **VehicleInStock**, and once again, this is the parent process Dataslot whose value the form will be showing.

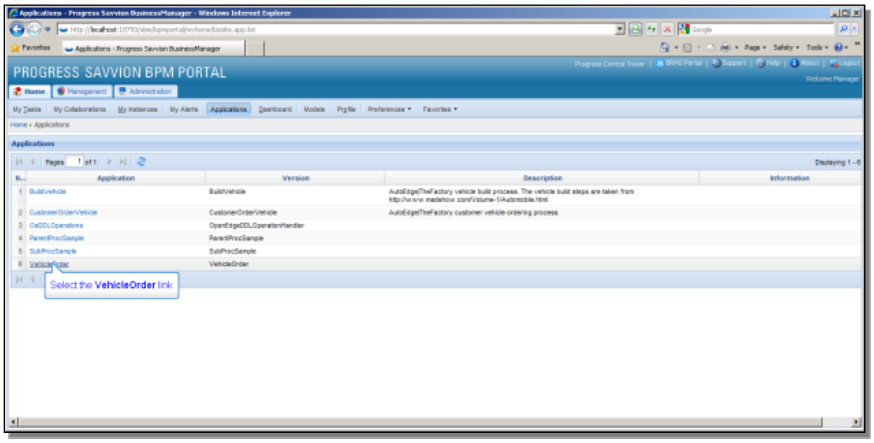


That's all I need to do here. I insert the new Activity into the process sequence in between the **BuildVehicle** Flow step and the **End** step:

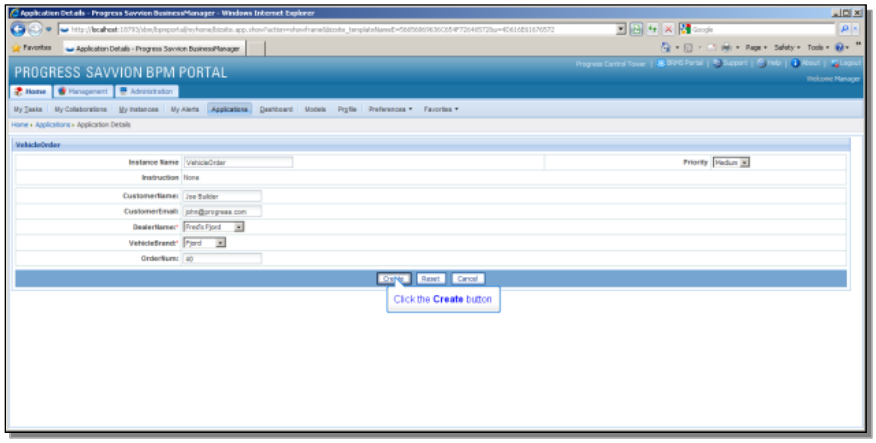


I'm done with changes to the VehicleOrder process, so I can save it. Now I need to redeploy both processes to the server, so I can see how they work together.

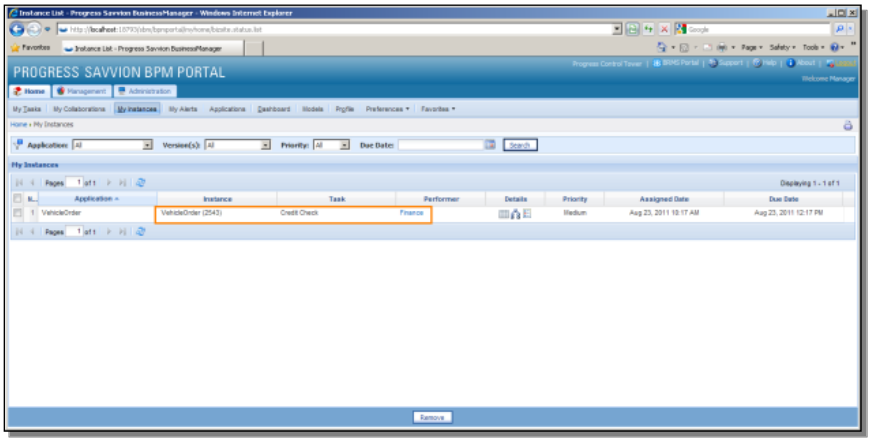
After deploying the two modified processes to the server, I can log into the BPM Portal, and select the **VehicleOrder** application to start up an instance of it:



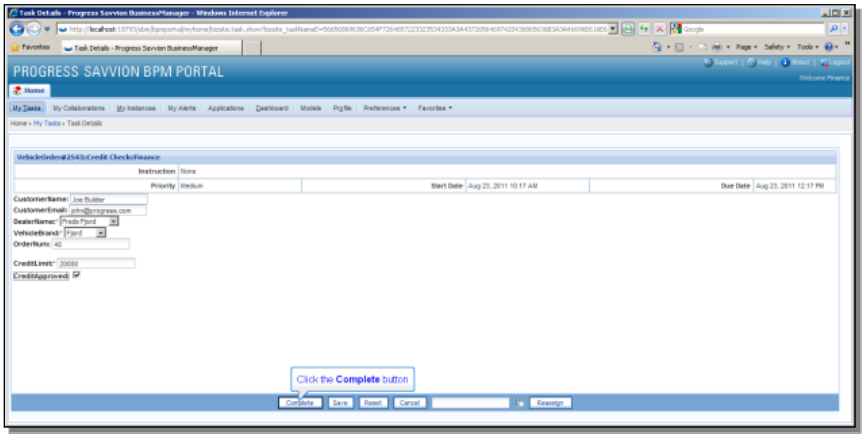
You will remember what this looks like from the presentation on building your first project. I enter a customer name, and an email, assign it an order number, and set the process running:



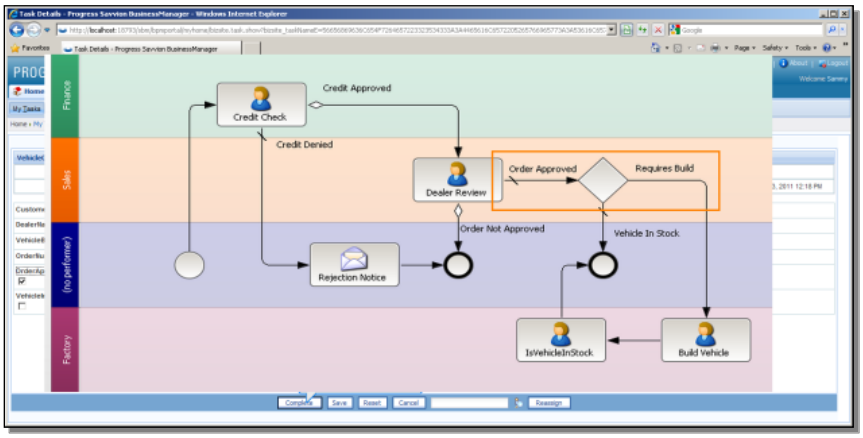
Back in the Portal, if I check the running instances, I can see the one I just created, and remind you that the next step is the **CreditCheck** that someone in **Finance** does:



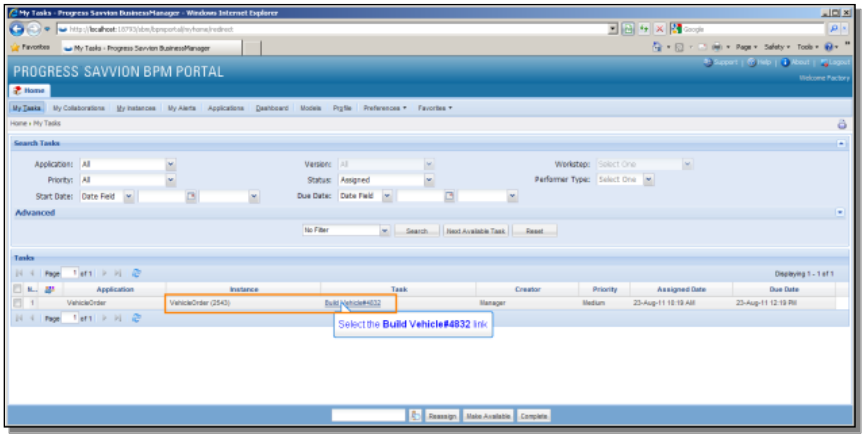
After I log out and log back in as **Finance**, I see the **CreditCheck** task, where I can approve the buyer's credit, and complete that step.



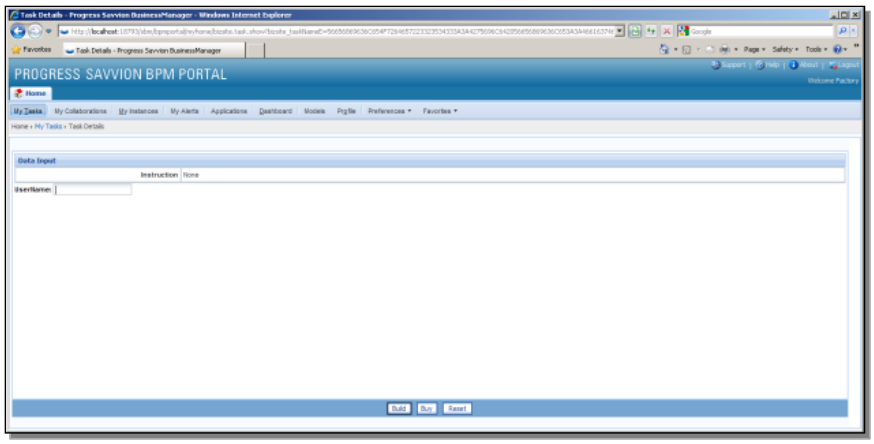
Then it goes on to the **Sales** department for **DealerReview**. In this case I approve the Order but don't check **VehicleInStock**. Taking another look at the process diagram, you can confirm that if the order is approved but the vehicle is not in stock, the process goes to the BuildVehicle step; that's now the presentation flow, and it's executed by someone at the **Factory**:



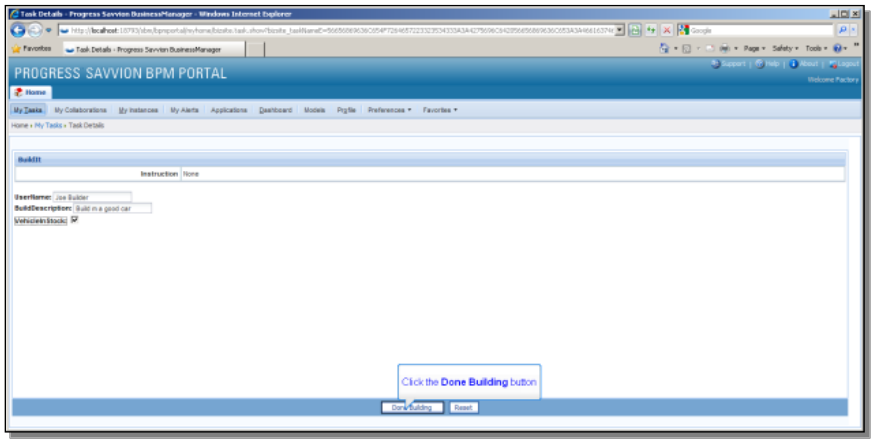
Logging into the Portal as **Factory**, you see that the **BuildVehicle** task is the next one up:



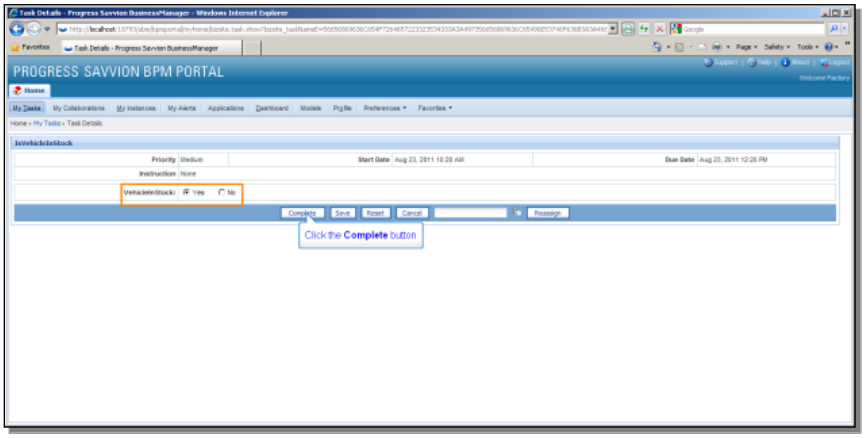
When I initiate that task, it runs the Flow, and its first form, the **DataInput** form, comes up. I enter a username. I can use the same name as the customer name I entered earlier, but keep in mind that this is one of the Flow's Dataslots, independent of the parent process.



I click **Build**, and the modified **BuildIt** form comes up next. I enter a description, and I'll just presume that a miracle happens, so that as soon as I do that, the vehicle is built and is now in stock. So I click that checkbox, and I'm done building the car.



Because the Flow is now embedded in a larger process, the **returnPage** Dataslot with the website URL, which I defined when I used **BuildOrBuy** as an independent Web application, doesn't have any effect; it's just ignored when the Flow is used as a sub-process. Instead, the Flow completes like any other subprocess. Back in the **Factory** view of the Portal, the new **IsVehicleInStock** Activity workstep is next. This is where we get to see if the VehicleInStock dataslot value was successfully passed back to the parent process. And sure enough, it was:



The VehicleOrder process is complete, and so it this presentation. Here I showed you how to take a Presentation Flow, which executes a series of browser-based forms navigated by a single user, and integrate it into a parent process that has other components involving other users and adapters. Like any other subprocess, the Flow can be reused in other parent processes where the same behavior is needed.