BUILDING SUBPROCESSES

John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
July 2010

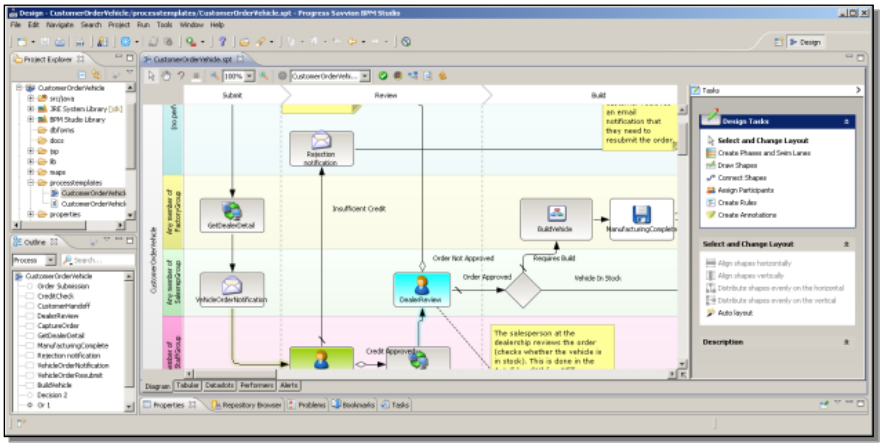
DISCLAIMER

Certain portions of this document contain information about Progress Software Corporation's plans for future product development and overall business strategies. Such information is proprietary and confidential to Progress Software Corporation and may be used by you solely in accordance with the terms and conditions specified in the PSDN Online (http://www.psdn.com) Terms of Use (http://psdn.progress.com/terms/index.ssp). Progress Software Corporation reserves the right, in its sole discretion, to modify or abandon without notice any of the plans described herein pertaining to future development and/or business development strategies. Any reference to third party software and/or features is intended for illustration purposes only. Progress Software Corporation does not endorse or sponsor such third parties or software.

This paper accompanies the video which is another in the series on building business process applications with OpenEdge BPM, combining OpenEdge and Progress Savvion.

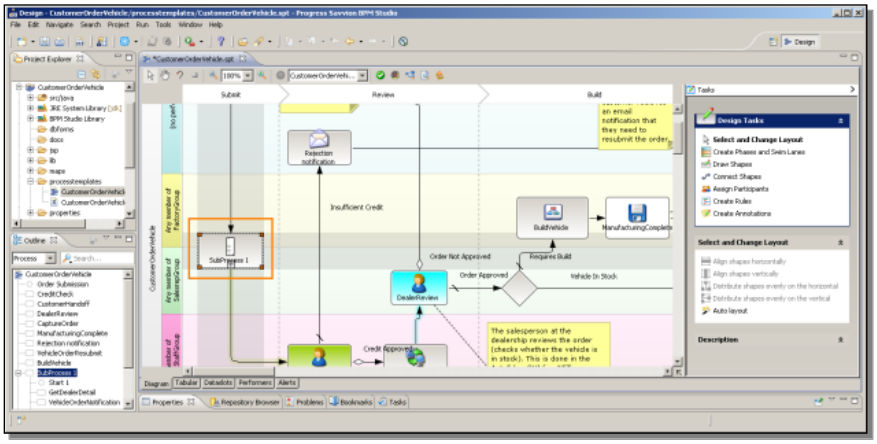
Just as you wouldn't want to put all of an OpenEdge application into a single ABL procedure, you won't want to put an entire complex business process into a single BPM project either. This paper shows you how to break up a process into multiple parts, by creating subprocesses. Just as you can with a procedure that represents part of a larger application, a subprocess can be reused as a part of many different larger processes, and a process that can stand alone under some circumstances can also be incorporated into other processes as a subprocess.

To start with, I'll open the main **AutoEdge | The Factory** process, **CustomerOrderVehicle.spt**:

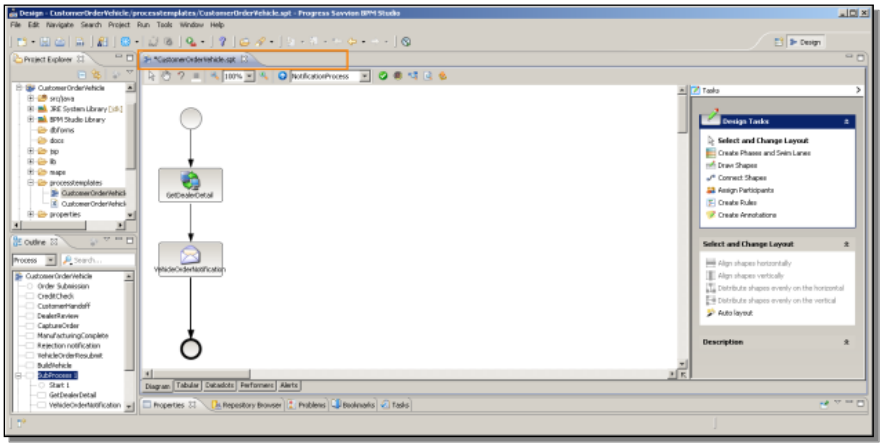


One simple way to break up a process visually is to select parts of it that you want to be able to drill down into from the main diagram. If I draw a box around the two steps **GetDealerDetail** and **VehicleOrderNotification** in the CustomerOrderVehicle process, for instance, I can then right-click on any of the selected steps, and select the **Collapse** option from the context menu.

All the steps I collapsed are replaced by a single visible step in the main process diagram. The fact that it's been collapsed is indicated by a plus sign in the step, but that's just decoration; if you click on it nothing happens.

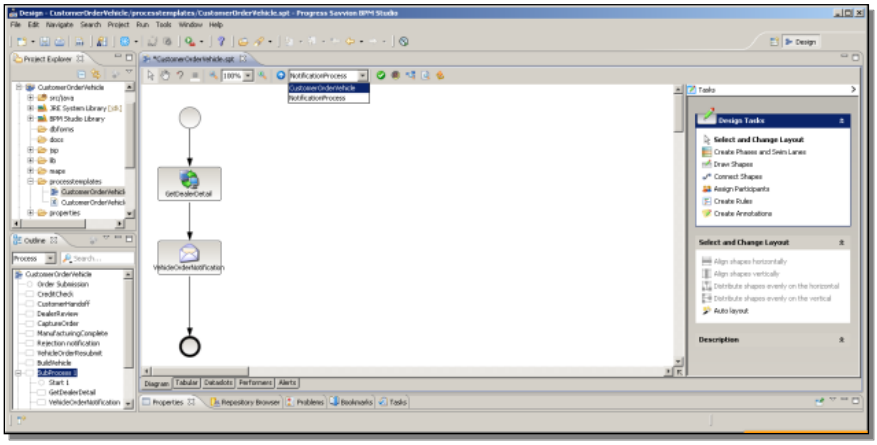


However, you can right-click on the subprocess step, and see that it has properties of its own. I can give the step a name, such as **NotificationProcess**, and I can also open it. Here the steps that I collapsed are shown in a new diagram window:



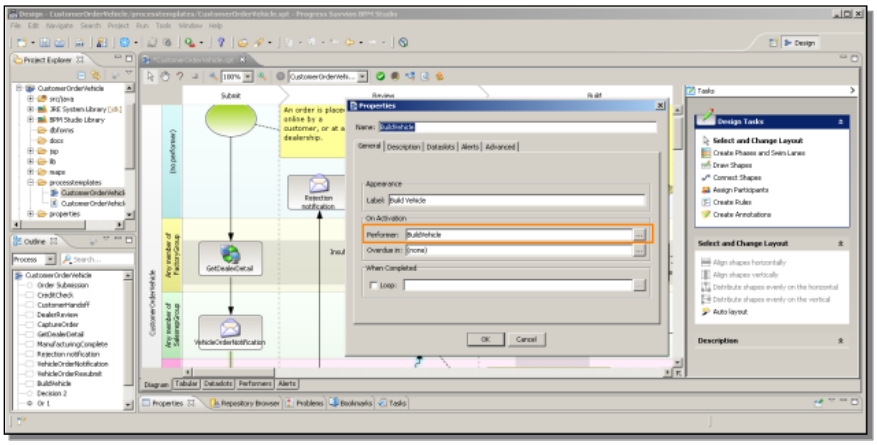
Note that there's no design window tab for the subprocess. What I just created is called an **internal subprocess**. It's just a separately-displayed part of the main process. It can't be saved independently, it can't be reused from some other process, and it shares the namespace of the process that it's part of.

If I look at the Dataslots from the subprocess, for instance, I see all the Dataslots in CustomerOrderVehicle. If I define a new one here, it is just added to the parent process's dataslots. The same is true of performers: parent process performers are available to the internal suib-process. So an internal subprocess has limited value. It just lets you break up the visual process diagram into multiple levels, which in a big process can be very useful; but you can't reuse the subprocess in any way. Since it doesn't have its own tab in BPM Studio, you can't really close it. Instead, to return to the parent process, you select the dropdown at the top of the design window, and select the main process diagram from there.

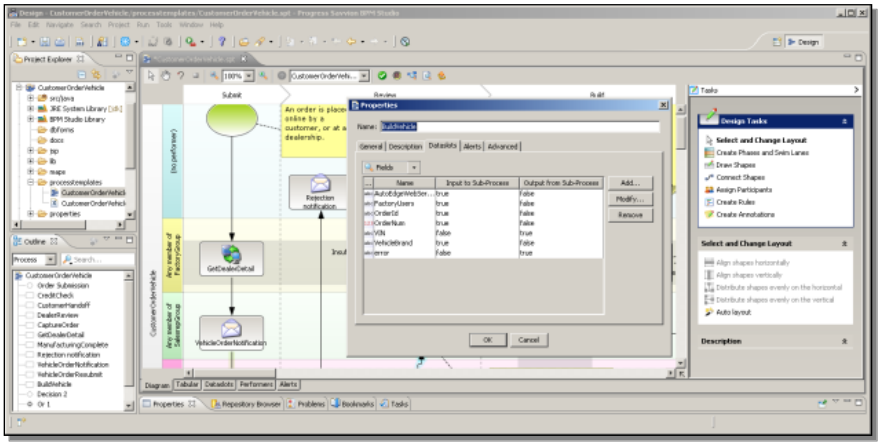


You could create multiple nested levels of internal subprocess, and then select which level you want to go back up to. That's a quick introduction to how to create and use an internal subprocess.

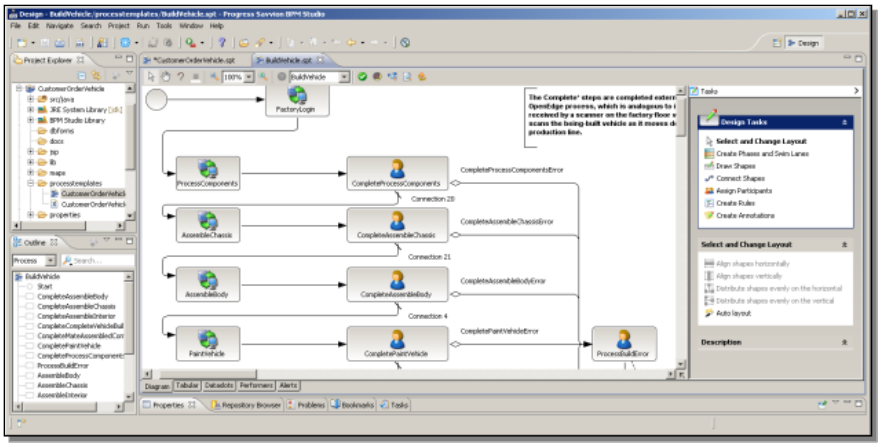
The more flexible way to separate out parts of a process is by creating an **external subprocess**, one that can be saved and used independently. The AutoEdge sample has an example of an external subprocess, called **BuildVehicle**. If I right-click on that step in the CustomerOrderVehicle process, and select its **Properties**, you see that the performer is called **BuildVehicle**:



If I look at its dataslots, I see CustomerOrderVehicle dataslots that are effectively passed as parameters from the parent process to the subprocess, and returned. The true and false flags indicate which ones are passed in, and which ones are returned as output:



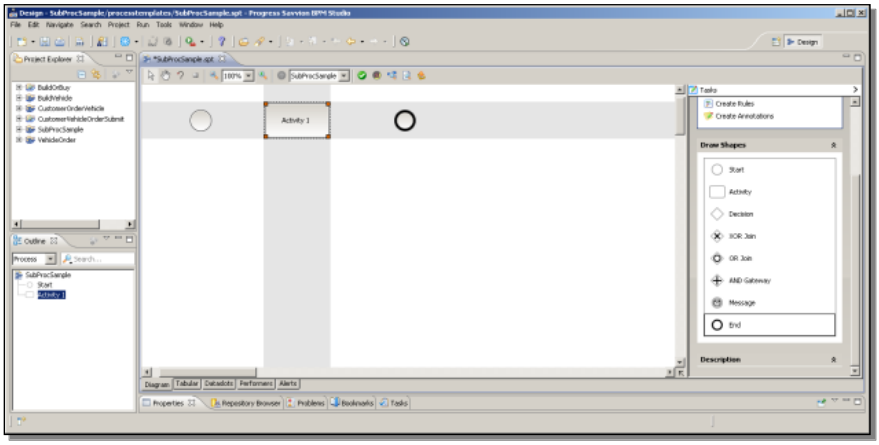
As with the internal subprocess, I can open BuildVehicle from here, but when I do, it gets its own tab for its design diagram, because this is in fact a separately saved and deployed process of its own, just like one ABL procedure called from another procedure:



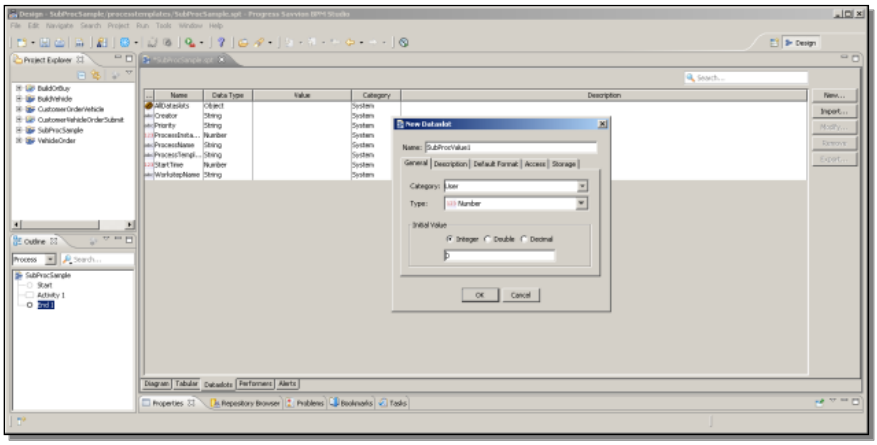
If I select the Dataslots tab in the subprocess, I see just the Dataslots that are defined for the subprocess. They occupy an independent namespace from whatever parent process invokes the subprocess. The same is true for the subprocess performers. Because BuildVehicle is built and deployed independently, it can be invoked from any number of other processes like CustomerOrderVehicle. This lets you separate out a part of a process that you want to be able to reuse in different places, just like a separate ABL procedure file.

What I will do now is to show you just how to create and configure an external subprocess. BuildVehicle is rather complicated to use as an example, so I'll just create simple examples to show you all the steps that are involved.

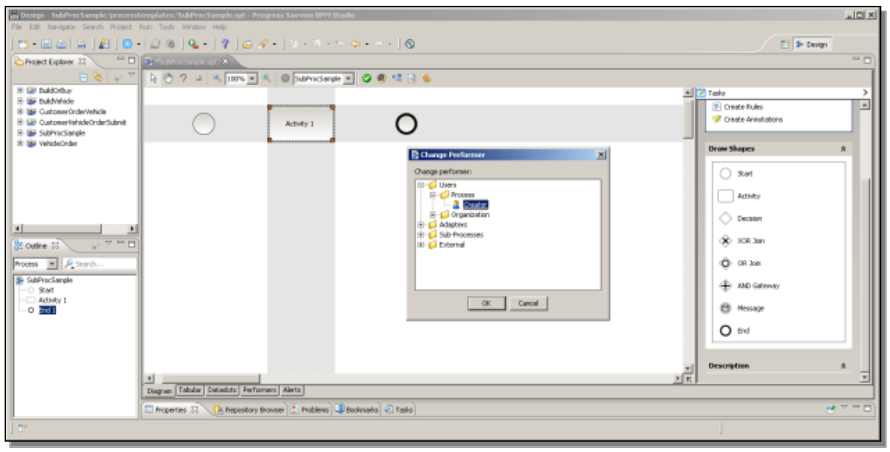
I create the subprocess first; it's a BPM Project just like any other process. I name it **SubProcSample**. All I need to create a minimal functional example is a Start step, a single Activity workstep, and an End step.



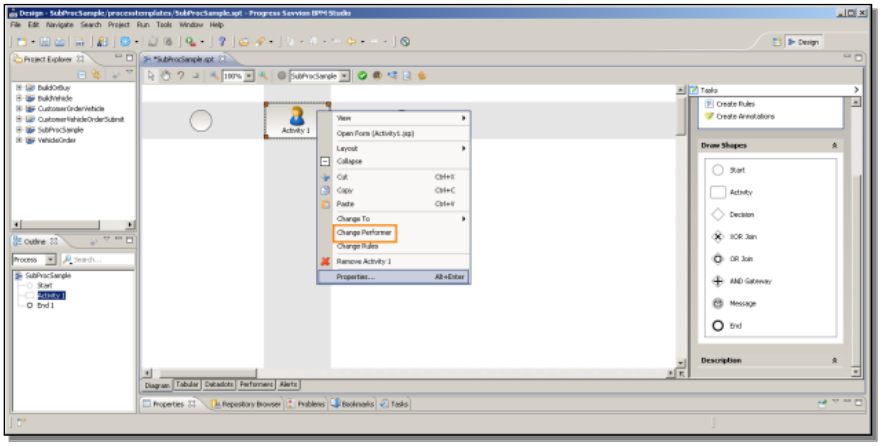
I'm starting from scratch with my dataslots, so I just see the built-in System dataslots. I create a user dataslot, which I define as a number:



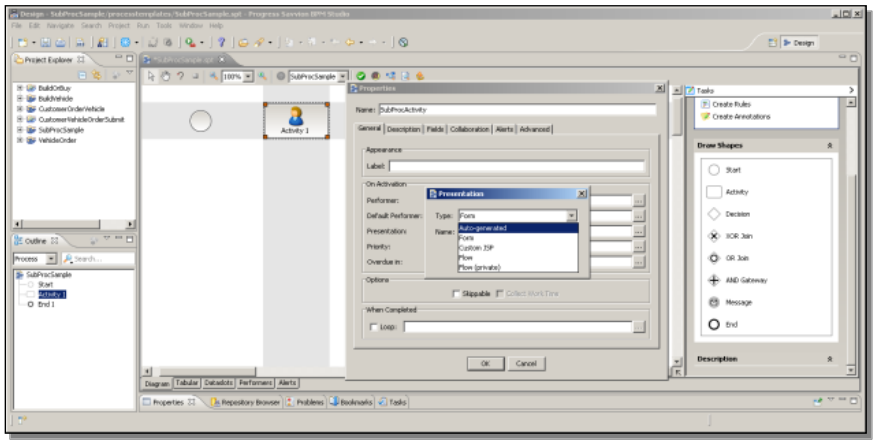
Then I create a second one, which I define as a string. These just give me something to display in the subprocess, and something I can use as parameters to pass back to the parent. The one subprocess Activity needs a performer, and for my purposes, there's a default performer just called **Creator** that I can choose, which means that whoever creates the process is expected to execute its worksteps.



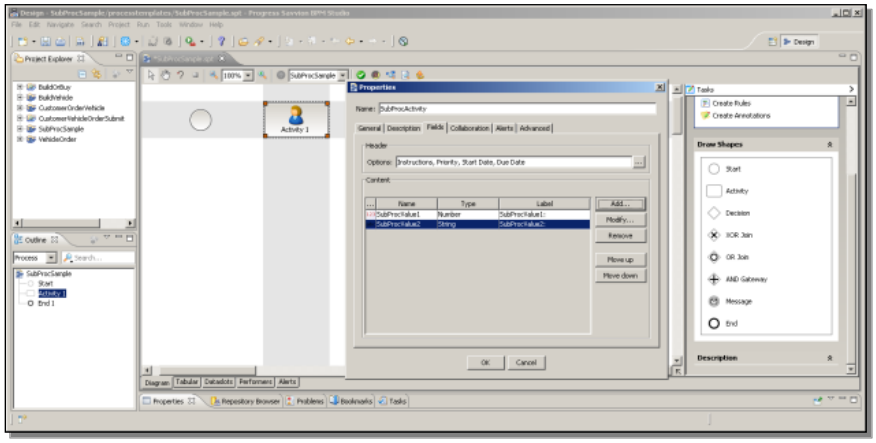
I go back into its properties, where you can note that you can also change the performer from the step's context menu:



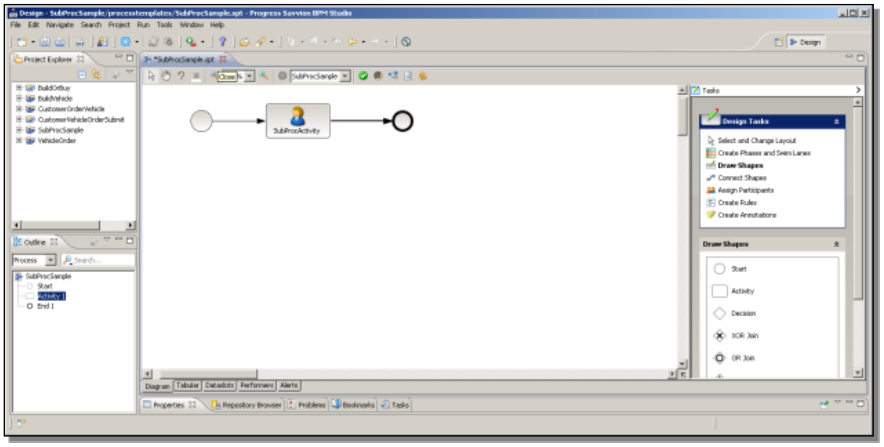
Note that you can also change the Performer from the workstep property sheet as well. In this case though, I just want to have Savvion auto-generate a form for me, so I select that Presentation type:



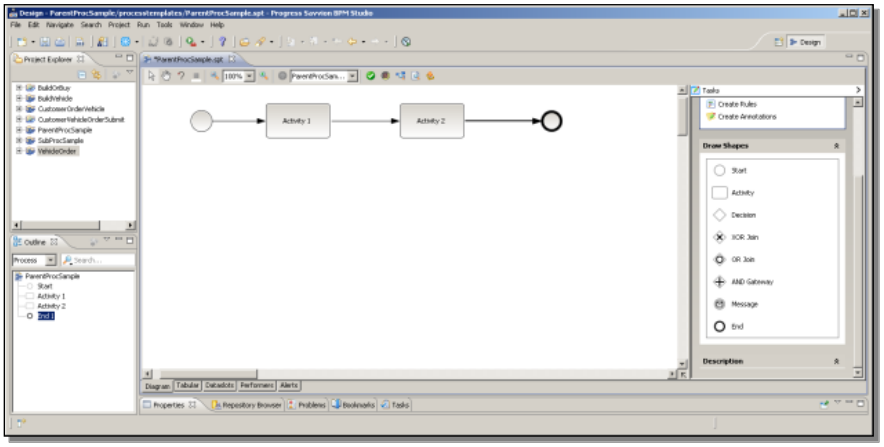
For an auto-generated form I just pick the dataslots I want to put into the form, by selecting them from the **Fields** tab. In this case I pick both of the dataslots I just defined:



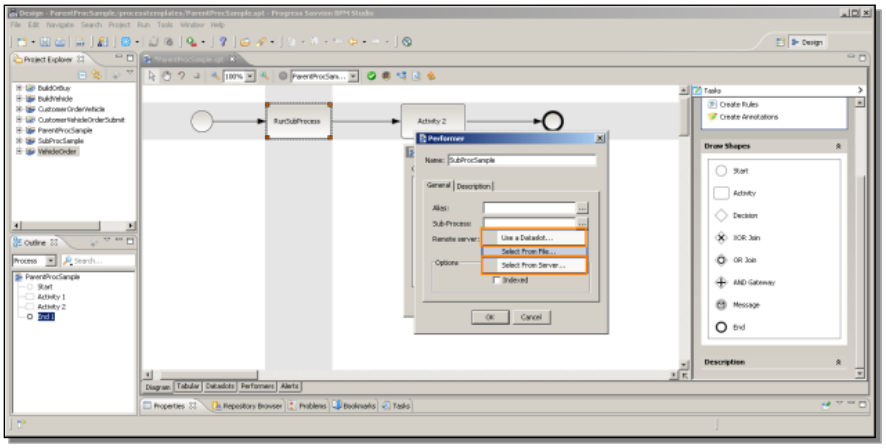
Now I have a single step defined in the subprocess. It displays the two subprocess dataslots, and allows me to change their values. I'm going to use these as parameters to receive as input from the parent process, and then to return. I can click on **Connect Shapes** to connect up my three step, or I can also just type **control-shift** and then **right-click** the mouse to do the same thing. I connect the Start step to the Activity that has the one form in the subprocess, and that Activity to the End step. That's all I need the subprocess to do.



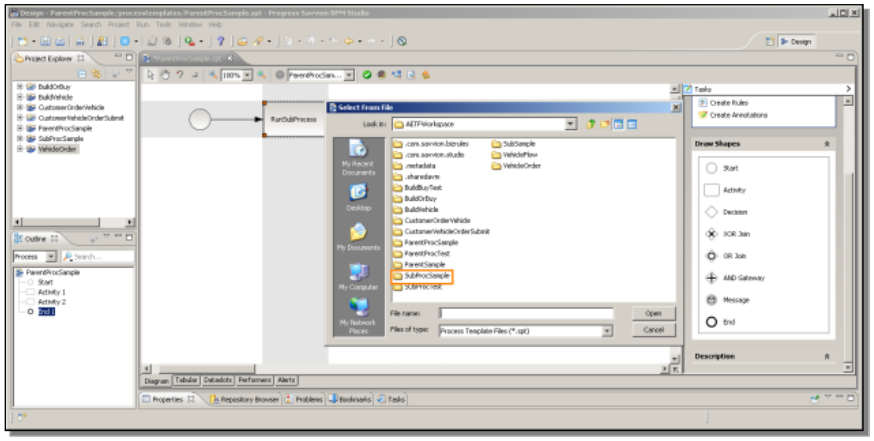
Now it's time to create the parent process. It's another BPM Project, of course, and I name it **ParentProcSample**. My simple example just needs two activities, plus of course a Start step and an End step. The first Activity is a call out to my subprocess. The second one displays the Dataslot values that come back. After dragging the right shapes onto the design surface I connect everything up.



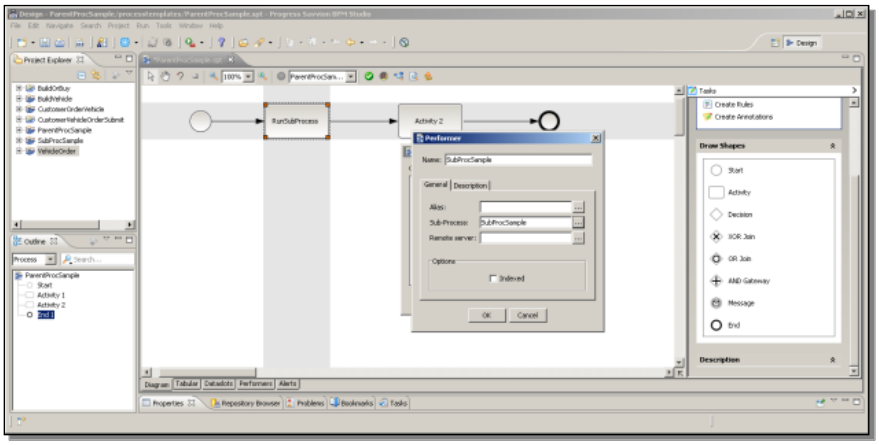
Next I define two Dataslots for the parent process, to match the two user Dataslots in the subprocess. Activity 1 is the step that runs the subprocess, so I name it **RunSubProcess**. I don't need to define a form for the workstep because I'm going to change the step's Performer to be a subprocess. Under Performers, there's a group called **Subprocesses**. Subprocesses that were already defined would be grouped under **External References**. I have to right-click to create a new one. There are various ways to do this, but the most straightforward is to use the **Sub-Process** option. I could use a Dataslot as the value of the subprocess name, which would allow me to set the name dynamically at runtime. I could also locate a process that has already been deployed to the server. But in this case I just select it from a file, because the process project been saved but not yet deployed:



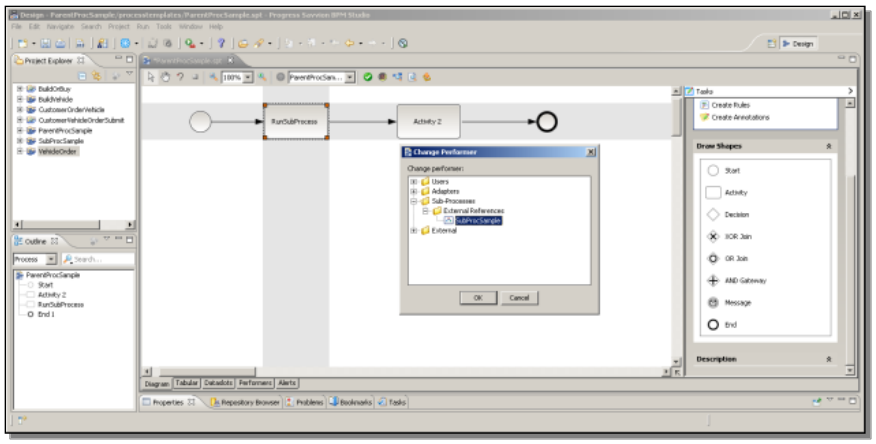
Here's the folder for **SubProcSample** in my BPM Studio workspace:



I select that Savvion Process Template file, and identify it as a performer available to this parent process:

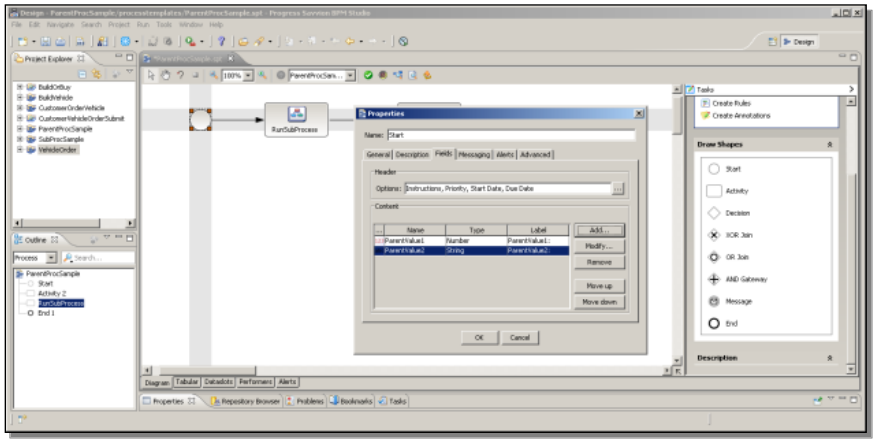


Now it's in the list of defined performers, so I can select it, and make it the performer for this workstep:

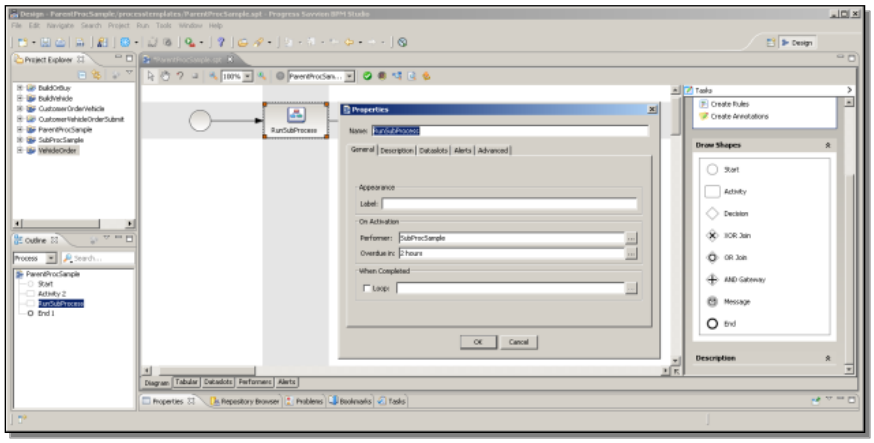


Now I've defined the first of the first two parent process activities to be a step that runs the subprocess. I want to define a simple form for the Start step, to set the two parent Dataslot values. That way I can pass them in to the subprocess and see what

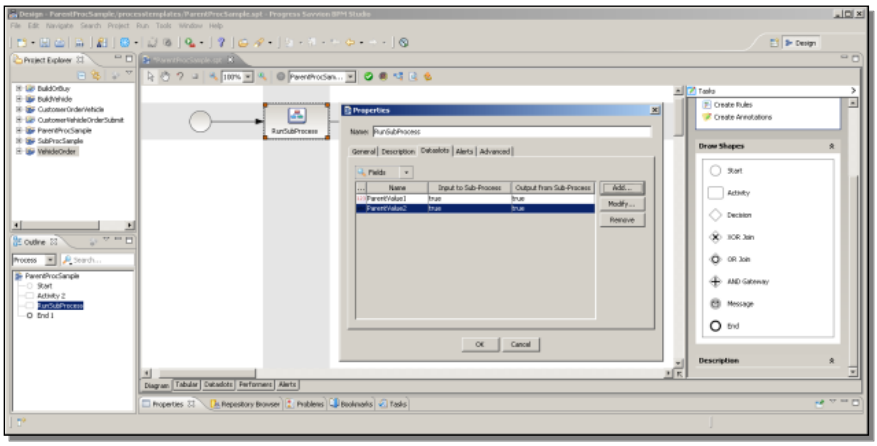
it takes to get them there and get them returned. The default Presentation type for a Start step is auto-generated, and I just click **Add** to add the two parent Dataslots to the form in the **Fields** tab:



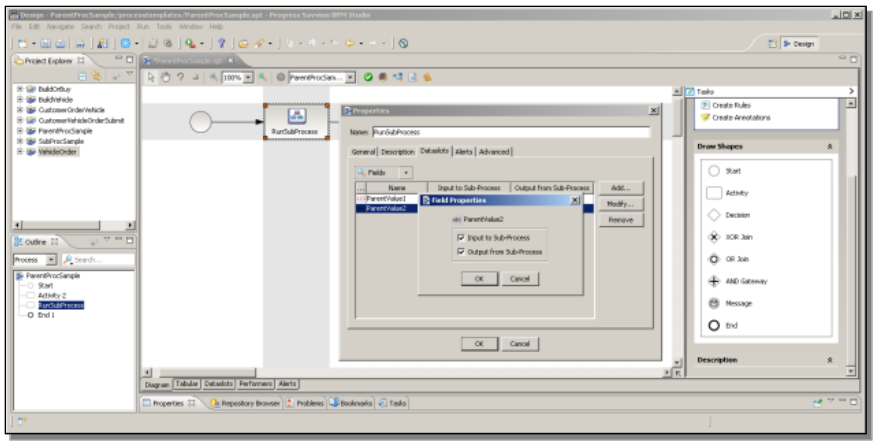
Next I need to define the parameters to the subprocess, and I do that by mapping Dataslots in the two processes:



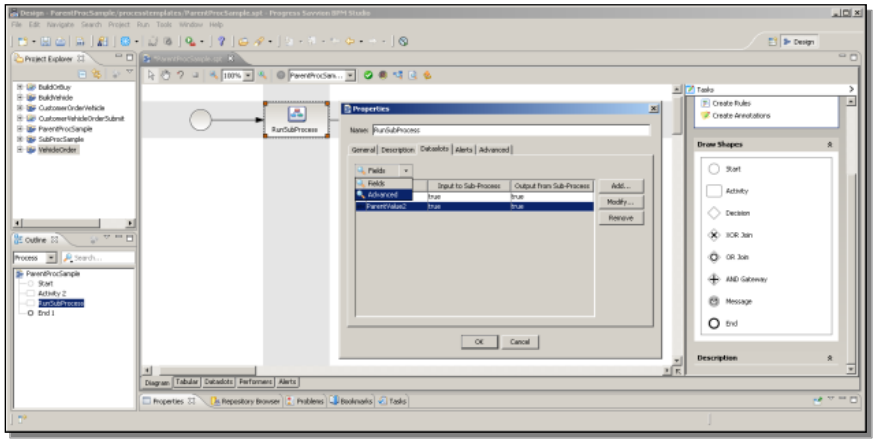
I add both my parent Dataslots as parameters:



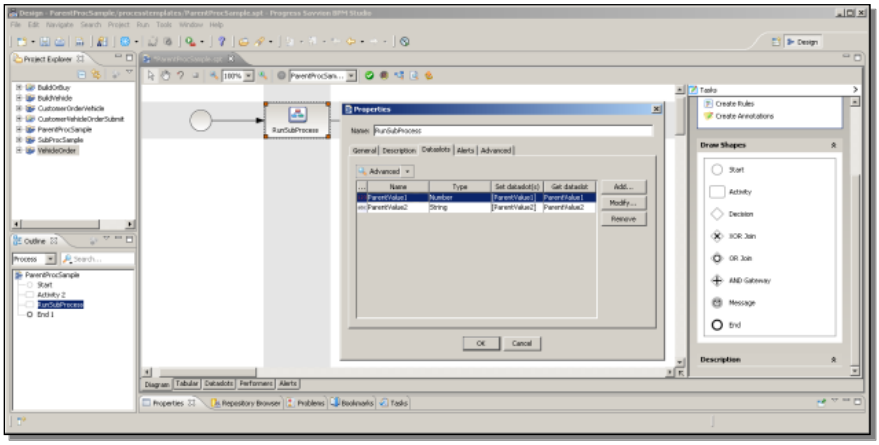
You can see that by default parameters are expected to be passed in, and also returned as output in case the subprocess modifies them. If I want to change that, I can pick a Dataslot and click **Modify**, and see two checkboxes that let me make a dataslot input only or output only:



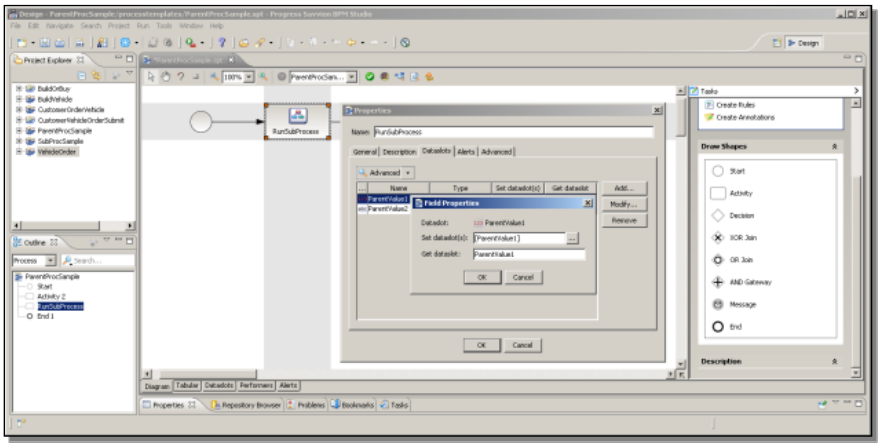
This basic display in the Dataslots dialog assumes, when I first add Dataslots as parameters, that the corresponding Dataslots in the subprocess will have the same names. Remember that the parent process and the subprocess have separate namespaces, so they can call their Dataslots whatever they want. If the target Dataslots in the subprocess have different names, then you need to select **Advanced** from this dropdown to map them:



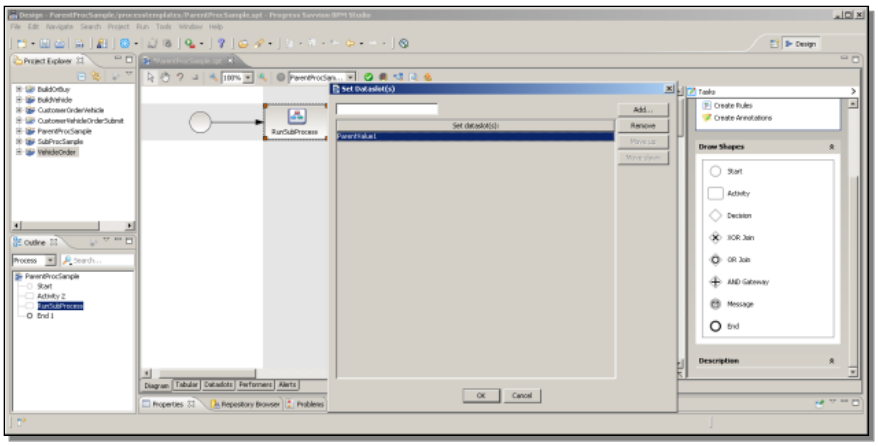
I select first **ParentValue1**, and click **Modify:**



A different dialog appears to let me change the **Set** Dataslot, that is, the name of the Dataslot in the subprocess that gets set to the parent value passed in:

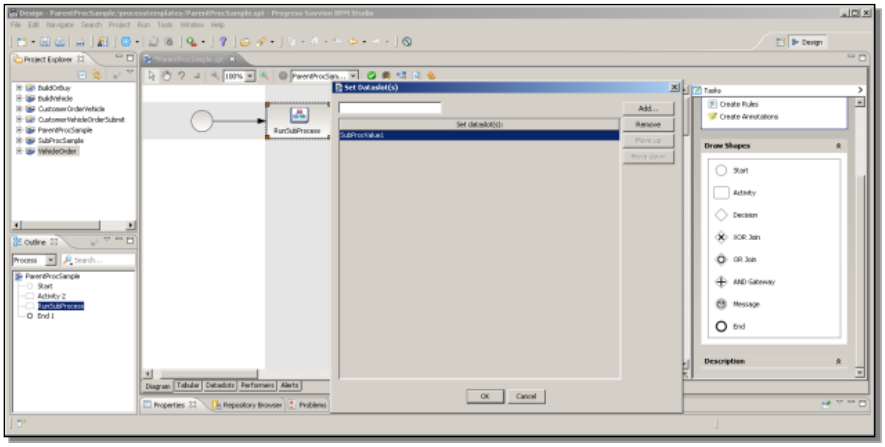


The dialog is prepared to let me map the value passed in to multiple Dataslots, which I don't want, so I select the default setting, and click **Remove**:

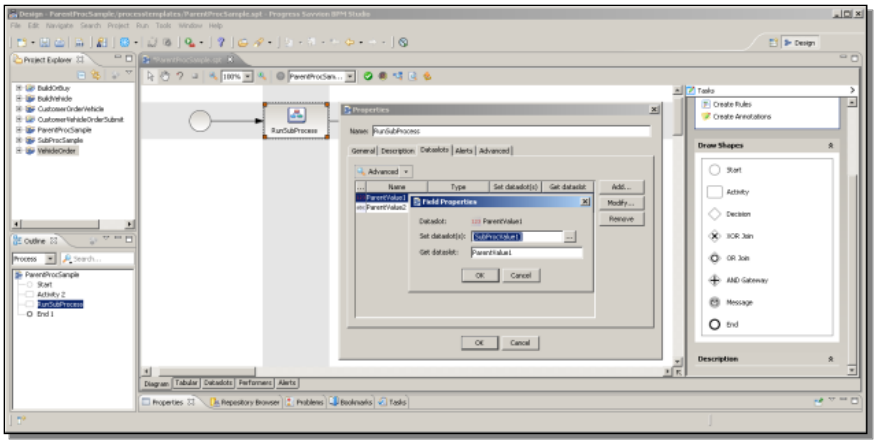


Then I type the name of the subprocess Dataslot I want to map to. BPM Studio doesn't actually reference the subprocess directly, so I have to type the name; I can't

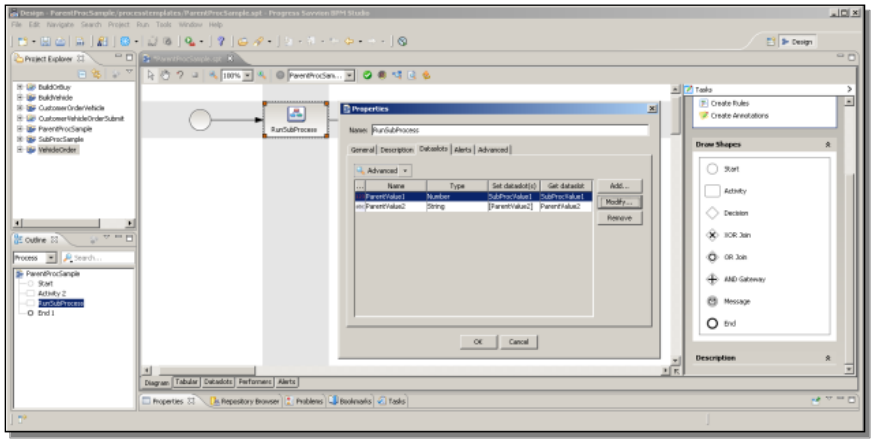
select it from a list, and there's no name checking at this point. So I click **Add**, and the input mapping is complete.



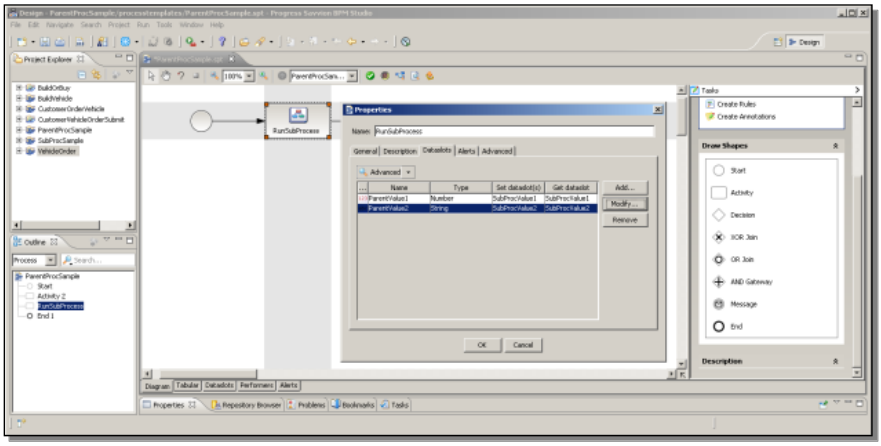
Next I select the **Get** Dataslot fill-in. This is the name of the subprocess Dataslot that will be passed back to **ParentValue1** as output:



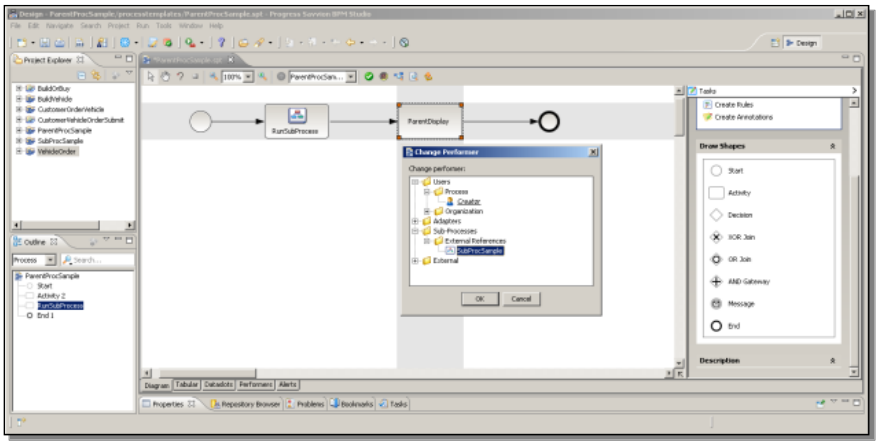
In this case I just edit it in place. I replace **ParentValue1** with **SubProcValue1**, and this parameter is complete:



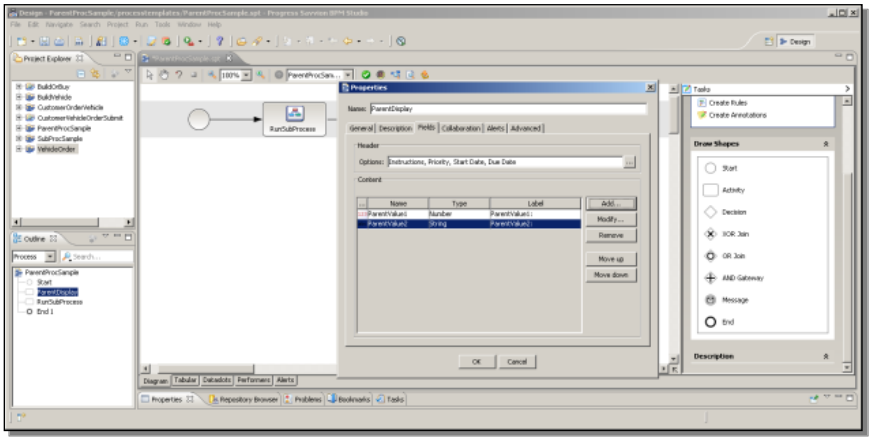
Now I do the same thing with **ParentValue2** and **SubProcValue2**. Here you can see the resulting mapping with the names of the subprocess dataslots that will accept input and return output:



Now I need to configure the second of the two parent activities, which is just going to display the values that come back, so I name it **ParentDisplay**, and define a form to display them. I need a performer for this step as well, and as I did in the subprocess, I just use **Creator**:

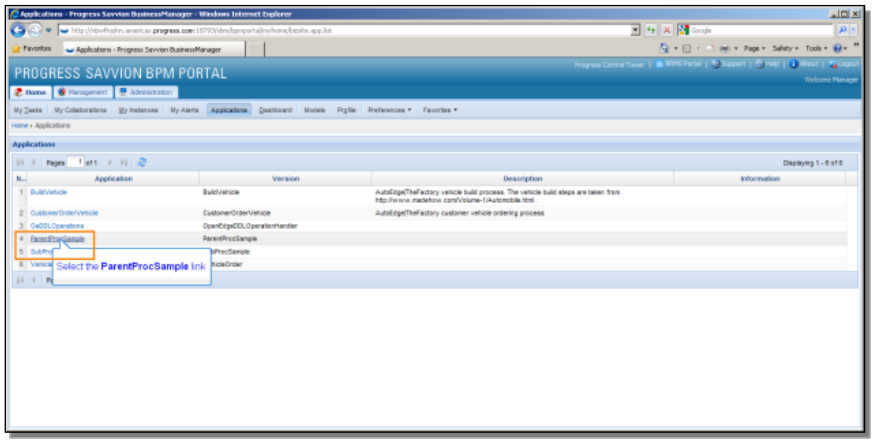


Now I need to add fields to the form, and I add both my two Dataslots, so that I can see what values were passed back from the subprocess:

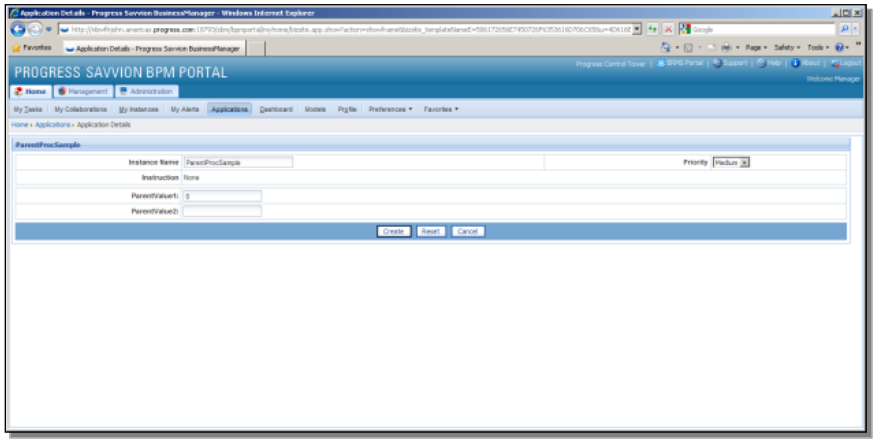


Now I save the parent process, and deploy both processes to the server. The subprocess gets deployed in exactly the same way as the parent process. After deploying both processes to the server, it's time to go see what happens when I run them.

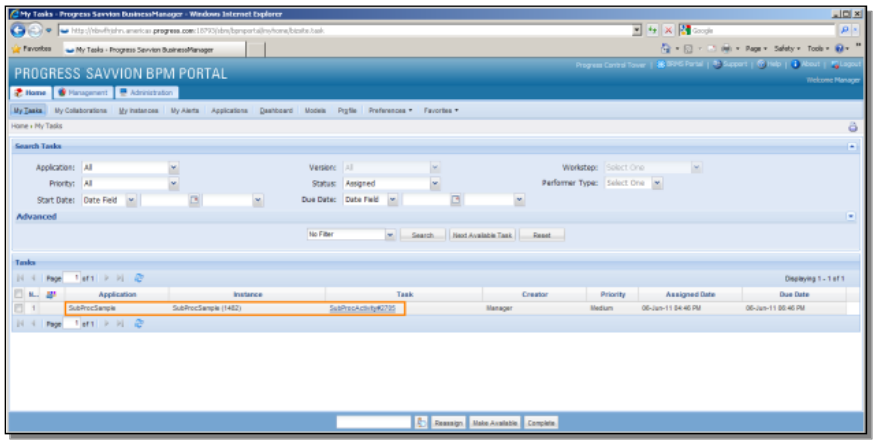
Below you see the BPM Portal, the runtime environment. As you can see, both the parent process and subprocess have been deployed and are listed here. I can click the Parent process link to start an instance of it:



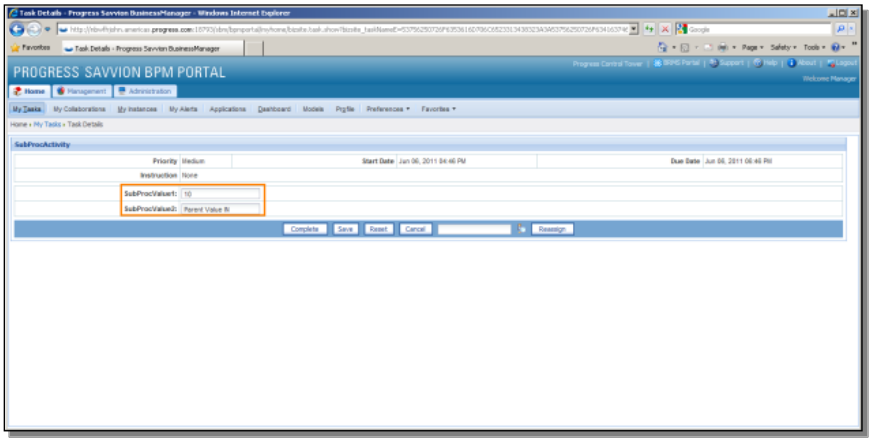
Here's the default form for the Start step:



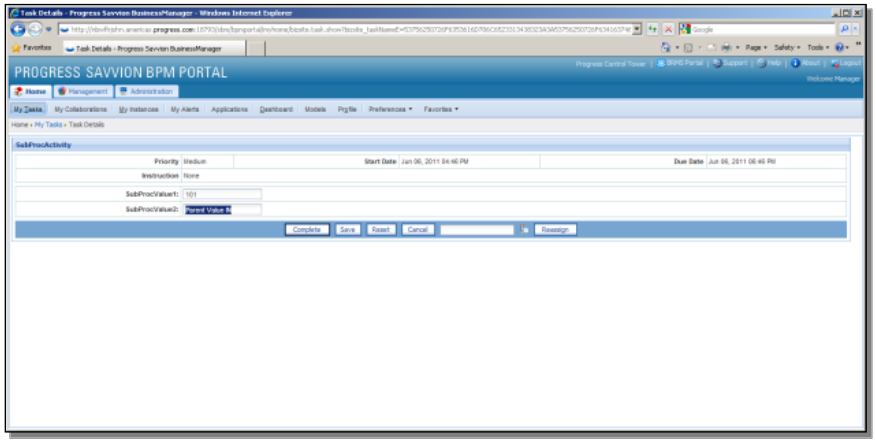
I set both the parent dataslots, the numeric one to **10**, and the string value to **Parent Value IN**, to confirm that this is going to be input to the subprocess. The **Create** button completes the creation of the parent process instance. Since I'm the Creator, the process tasks will be assigned to me. Here you can see that the parent process has started an instance of the subprocess, in its first activity step. I can click on the subprocess activity to access it:



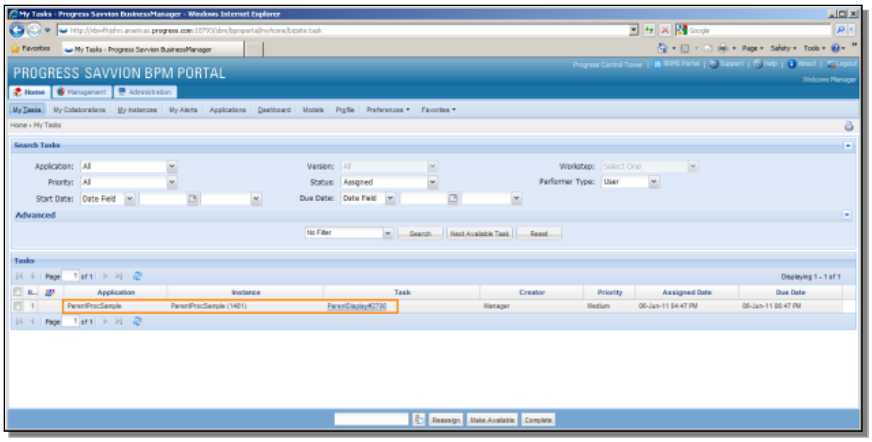
The form confirms that the two parent Dataslot values were successfully passed in and mapped to **SubProcValue1** and **SubProcValue2**:



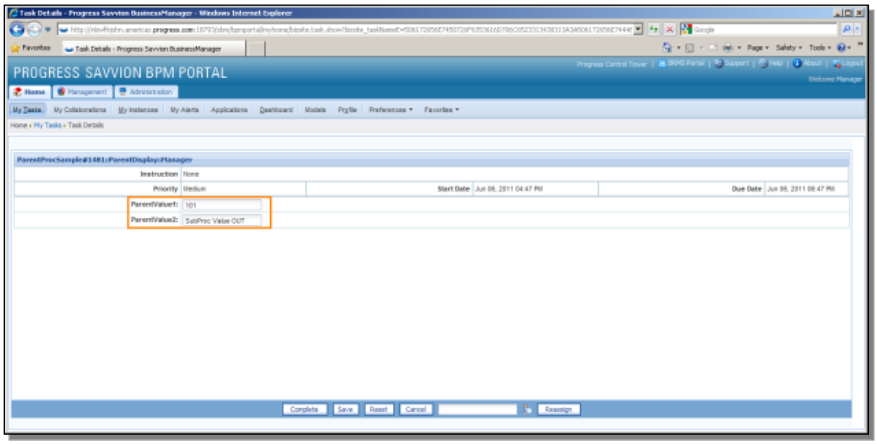
Let's see what happens on the way out when I modify them. When I click **Complete**, the subprocess is done and it returns to the parent process.



Back in **My Tasks**, I see the **ParentDisplay** activity in the parent process is now waiting for me to attend to it:



Here you can see that the two modified subprocess Dataslot values were returned as output to the parent process. That completes the parent process activities:



And that also completes this presentation. I've shown you how to define both internal and external subprocesses, and how external subprocesses are defined and deployed independently of any parent process that wants to use them. And I showed you how to pass Dataslot values as parameters from one process to another. Now you can break up a complex Business Process Application into as many reusable sub-components as you need to.