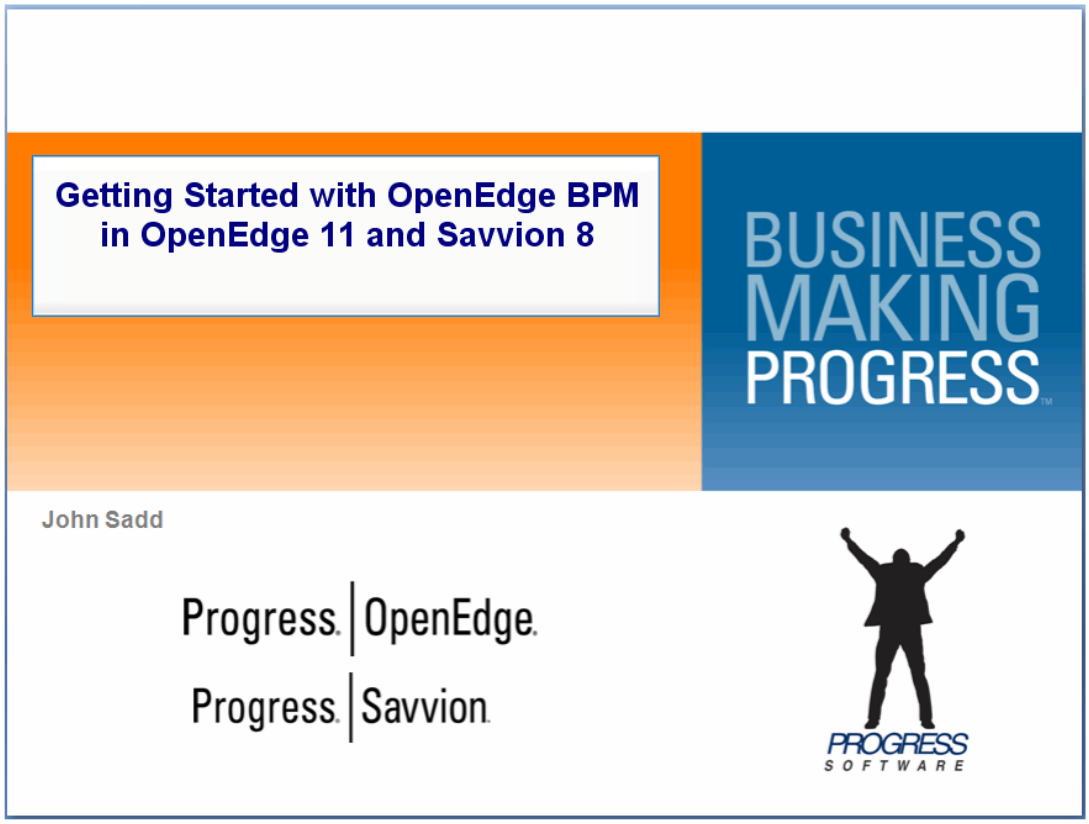
John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
May 2012



DISCLAIMER

This paper accompanies a multi-part presentation on OpenEdge BPM, using OpenEdge 11 and Savvion version 8 together to create process-driven business applications. In many ways this is a follow-on to the series of videos covering the two products using OpenEdge 10 and Savvion version 7, so if you haven't used the products together before, you can start with a look at those materials. This series focuses on what's new when you use the latest versions of the products together.

It is already true in the latest releases of OpenEdge 10 and Savvion 7 that you can use an OpenEdge database as the repository database for Savvion's process information, and I begin by starting a database server for that database. You could configure OpenEdge Explorer or Management to do this, but there's a batch file, **startdb.**bat, created in the **oebpmdb** directory as part of the install to start the database. Principally, it uses a .pf file to specify the startup parameters:

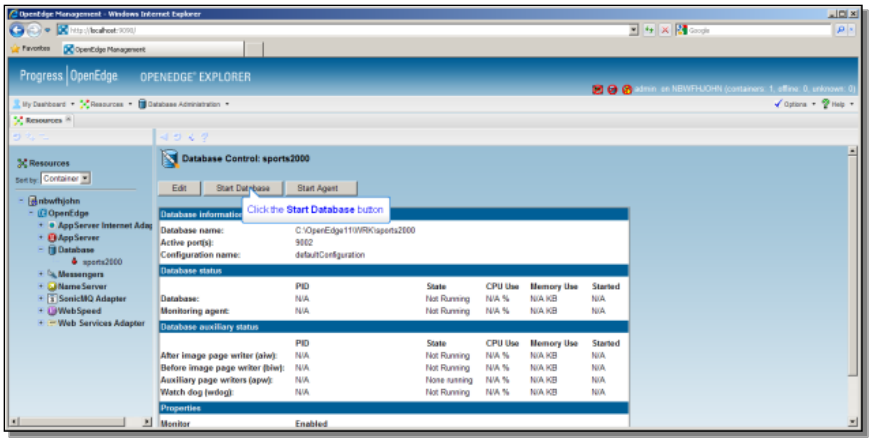
```
@rem
echo off
@rem
@rem Start the oebpmdev database server
@rem Generated on Wed 02/01/2012 at 15:42:55.99
@rem
setlocal
set PATH=;;;;;;;C:\Program Files\Common Files\Microsoft Shared\Windows
Live;C:\oraclexe\app\oracle\product\10.2.0\server\bin;C:\WINDOWS\system32;C:\WINDOW
S;C:\WINDOWS\System32\Wbem;C:\WINDOWS\system32\WindowsPowerShell\v1.0;C:\Program
Files\NTRU Cryptosystems\NTRU TCG Software Stack\bin\;C:\Program Files\Wave Systems
Corp\Gemalto\Access Client\v5\;C:\Program Files\Common Files\Roxio
Shared\DLLShared\;c:\Program Files\Microsoft SQL Server\100\Tools\Binn\;c:\Program
Files\Microsoft SQL Server\100\DTS\Binn\;C:\PROGRA~1\ABSOLU~1;C:\Program
Files\QuickTime\QTSystem\;C:\Program Files\Common Files\Microsoft Shared\Windows
Live;C:\Progress\OpenEdge11\bin;C:\Progress\OpenEdge11\BIN;
set PROSQL_LOCKWAIT_TIMEOUT=302;
call sql_env
call proserve -pf oebpmdev.pf
```

So looking at that **oebpmdev.pf** file, you can see that the database server is started on port 8910, as well as some other parameters that are needed by the way the database is used:

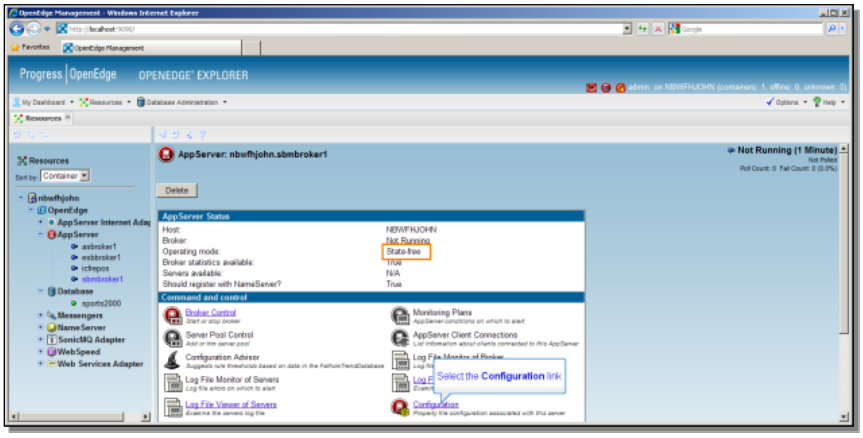
```
#
# Server startup parameters file for the oebpmdev database
# Generated on Wed 02/01/2012 at 15:42:55.96
#
-db oebpmdev
-H 127.0.0.1
-S 8910
-n 75
-bibufs 25
-L 32000
-B 5000
-SQLStmtCache 200
#
#
```

I simple double-click on startdb.bat in the file Explorer to start the database itself.

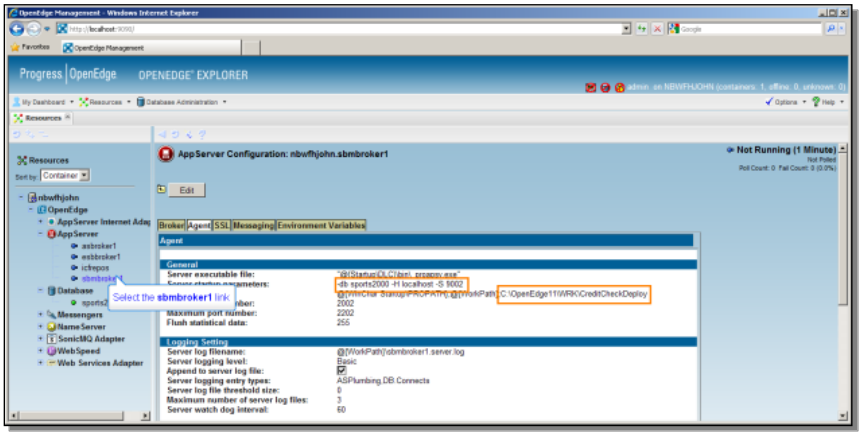
Once it's started, I can go on and start up the other supporting pieces of the environment. In OpenEdge Explorer, I have a sample database configuration that my simple example uses, and an AppServer where ABL business logic runs. I want to focus just on the mechanics of how the new features of the environment work, so my example is a very simple one that uses the familiar **sports2000** database, so I start a server for the database:



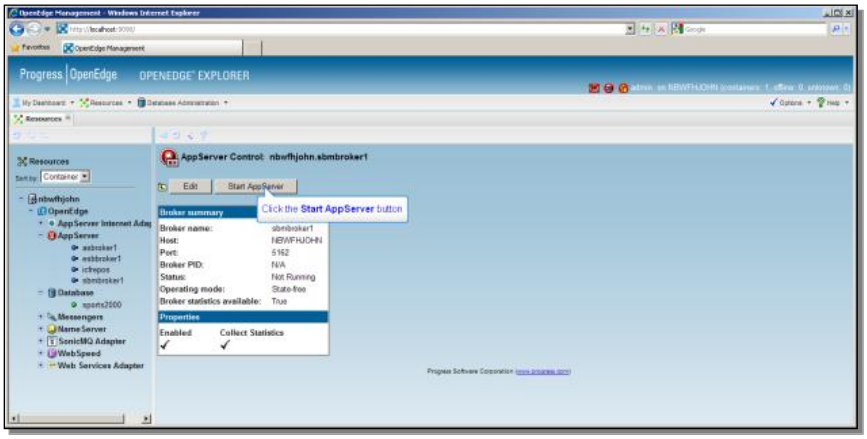
Then I start an AppServer configured to work with the SBM environment. This new AppServer configuration, called **sbmbroker1**, is defined for you as part of the OpenEdge 11 installation. There's no particular reason why you would have to use this one, but it's one of several built in server configurations that come with the OpenEdge product. You can see that it's configured to use the **state-free** operating mode, so that each request is independent of every other request:



Then in the **Agent** tab, I've added startup parameters to connect to the sports2000 database server I just started. In addition, I've added a directory to the end of the propath named **CreditCheckDeploy**. I'm going to define an AppServer for my development workspace, and this is the name of the directory that it will deploy AppServer code to:

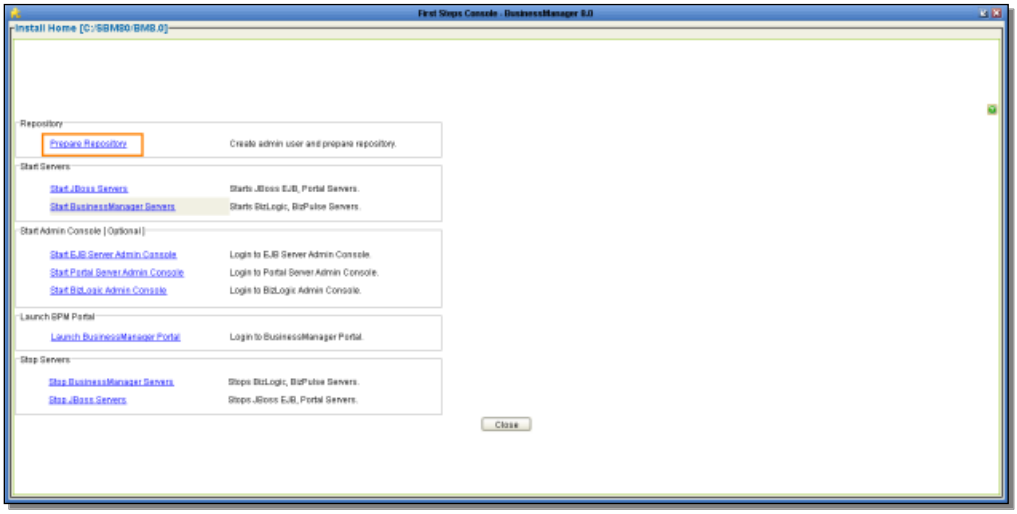


Now I can start the AppServer, and I'm ready to move on:



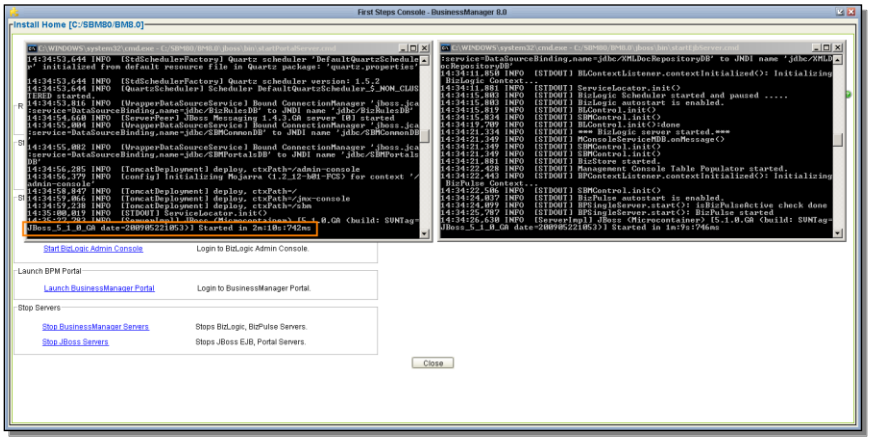
Next, from the **Progress SBM8.0** sub-menu in the Windows Start menu, I can select the **First Steps Console** to bring up a useful set of operations to select from. If you

are starting up your environment for the first time after installing Savvion, you need to click the **Prepare Repository** link and let it configure the repository database:

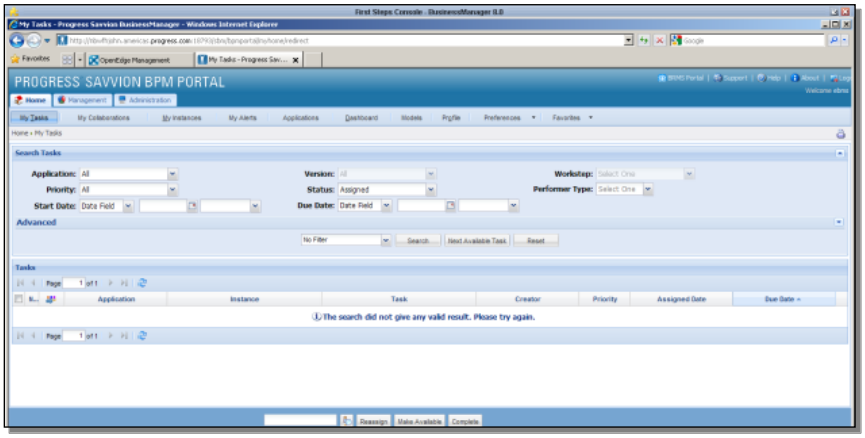


Otherwise, I select the **Start JBOSS Servers** link to start the servers that support the SBM runtime environment.

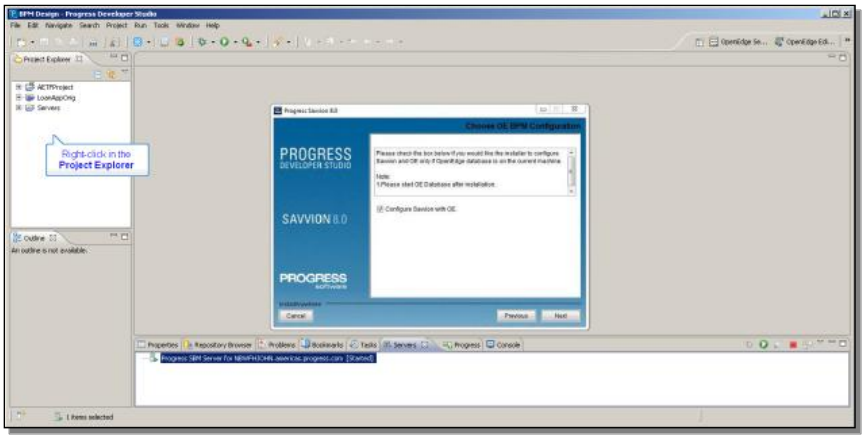
This can take several minutes, and you need to wait until you see the message in each of the two command windows that the **ejb server** and the **portal server** have both started. You'll see a message similar to the one at the bottom of the display in the two command windows shown here:



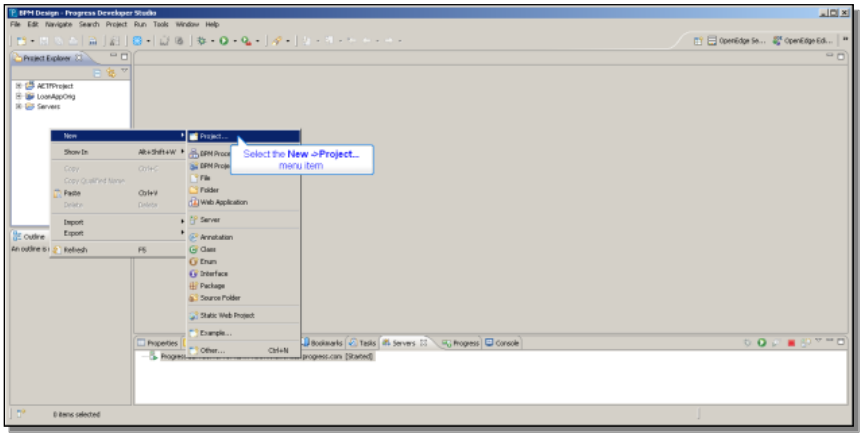
Once that's done, you can go back to the First Steps Console and select the link labeled **Launch Business Manager Portal**. This starts the Savvion runtime environment where you can start and monitor instances of the business processes you've defined. During the product installation process you have an opportunity to set the username and password that will be used for administration access to the SBM Portal. By default it's **ebms** and **ebms**. As you can see below, I haven't deployed any processes yet, so there's nothing much to look at in the Portal:



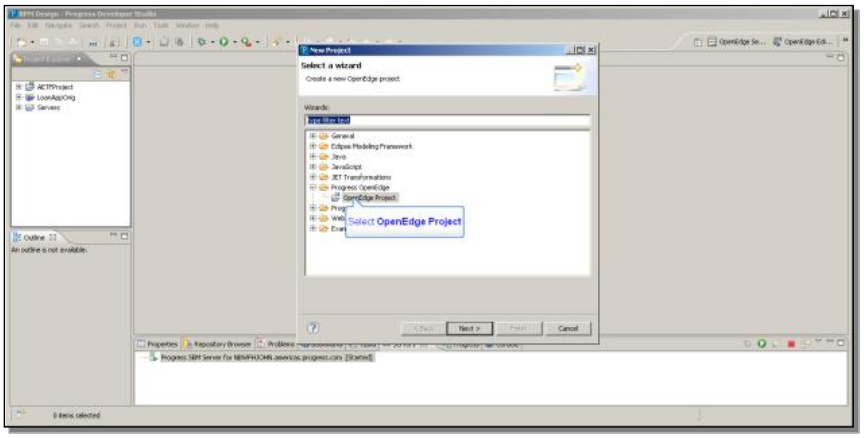
Next I move on to **Progress Developer Studio**. This is a combined development environment that's now supported in OpenEdge 11 and Savvion 8. To review the steps you go through to combine both versions of Developer Studio in one Eclipse-based environment, you should consult the Web paper distributed with the product titled ***OpenEdge Business Process Management Installation and Overview***. In particular, during the Savvion install, select the checkbox labeled **Configure Savvion with OE**, as shown below. This lets you use the same development tool for ABL code and other work related to the OpenEdge side of the application, as well as building and deploying Savvion process models.



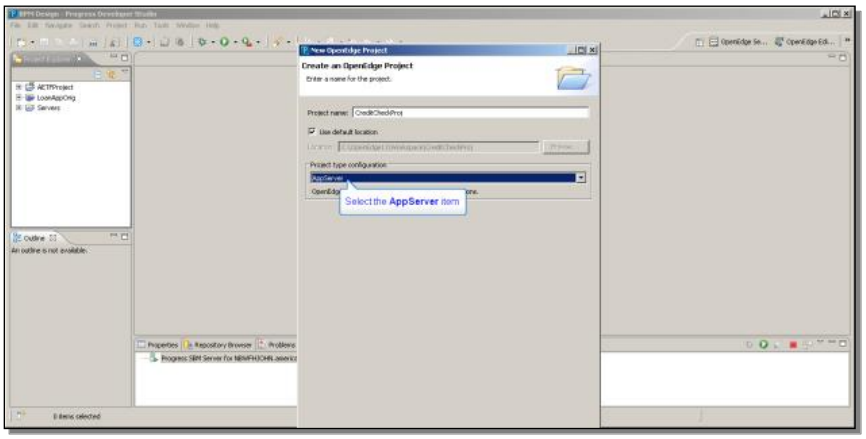
So after starting the combined Developer Studio, I'm going to create a new project:



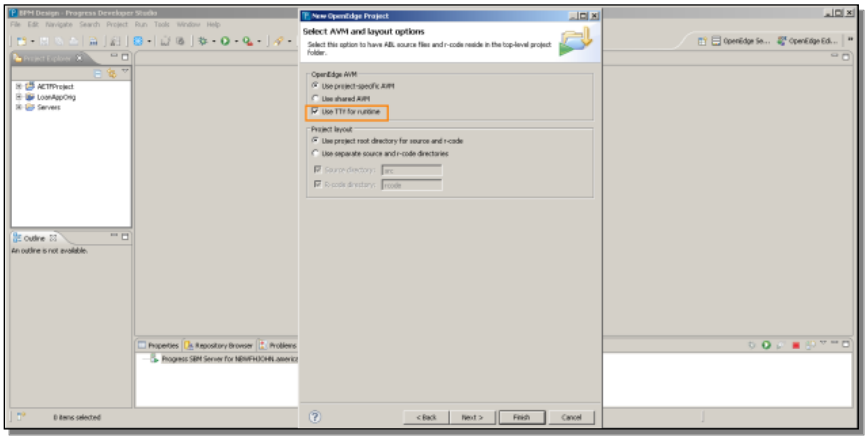
I call it an **OpenEdge Project**, though as you'll see, I'll be adding Savvion capabilities to it shortly:



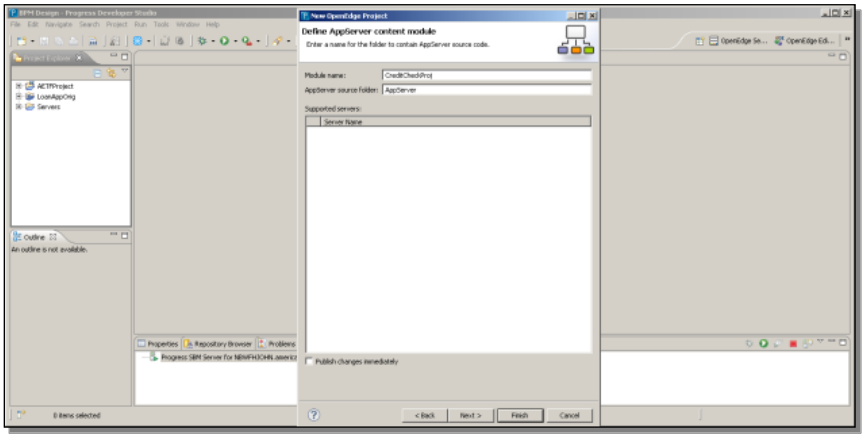
It's going to be the world's simplest credit check application, and this will be a project that deploys to the AppServer. That is, it won't have its own OpenEdge user interface, and the ABL procedures I develop will be business logic run on the AppServer, in this case from the Savvion process:



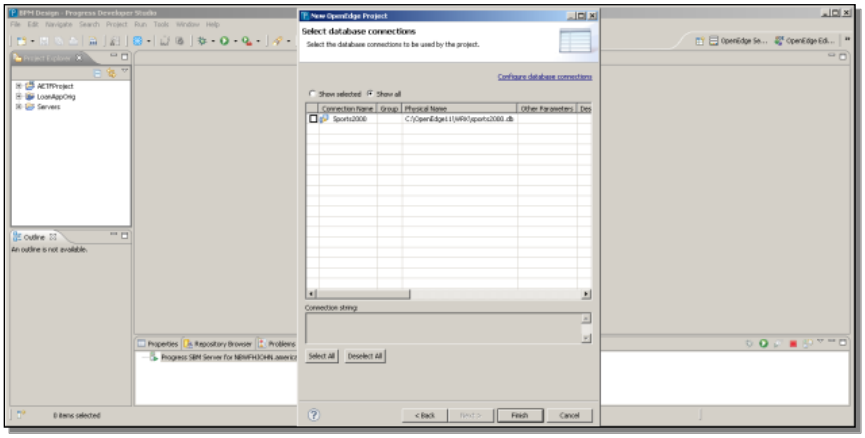
Moving on, you can see that the new project wizard has selected the checkbox labeled **Use TTY for runtime**. This is because I indicated that this is an AppServer project with no UI of its own:



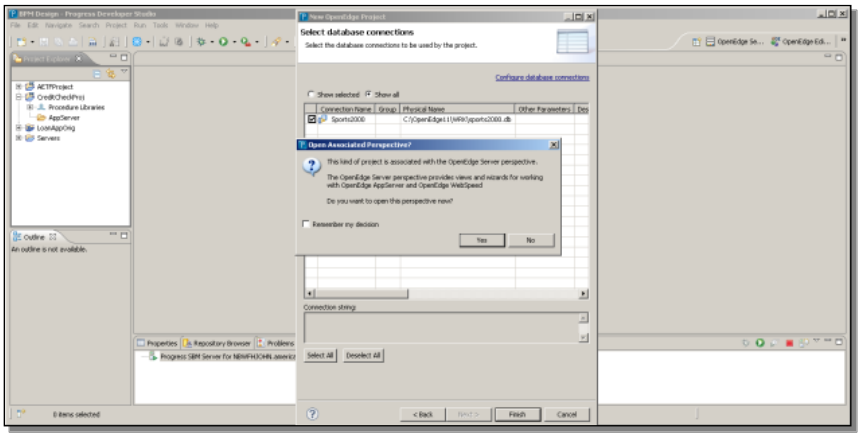
On the next wizard page, I can name the directory that will be used as the source folder for all code that is destined to run on the AppServer. Code that I locate under this directory will be published to the deployment folder I later name, which will be the **CreditCheckDeploy** directory I showed you in the AppServer's Propath when I was starting the AppServer in Explorer. I'll just keep the name **AppServer** as the source folder name. I haven't defined an AppServer in Developer Studio yet, so nothing is displayed in that part of the page. I'll get to that a little later.



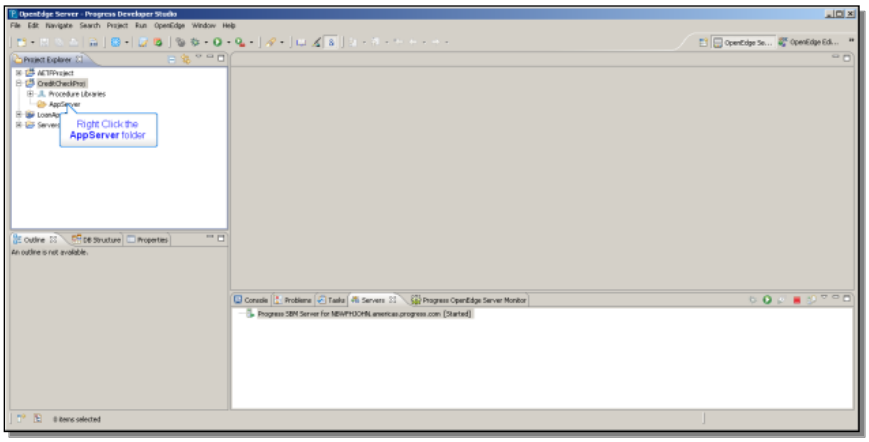
I had already defined a database connection in Developer Studio for sports2000, so I just select that to add to the project, and I'm done:



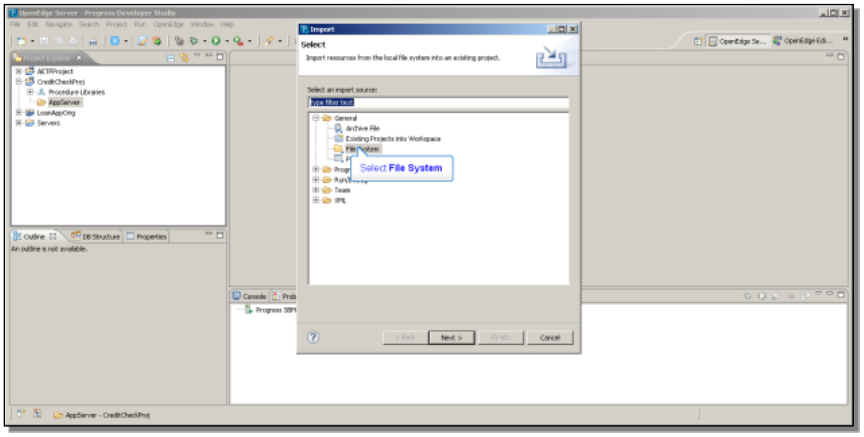
Because I defined an OpenEdge AppServer project, the tool invites me to open the OpenEdge Server perspective for my development, and I just answer **Yes**:



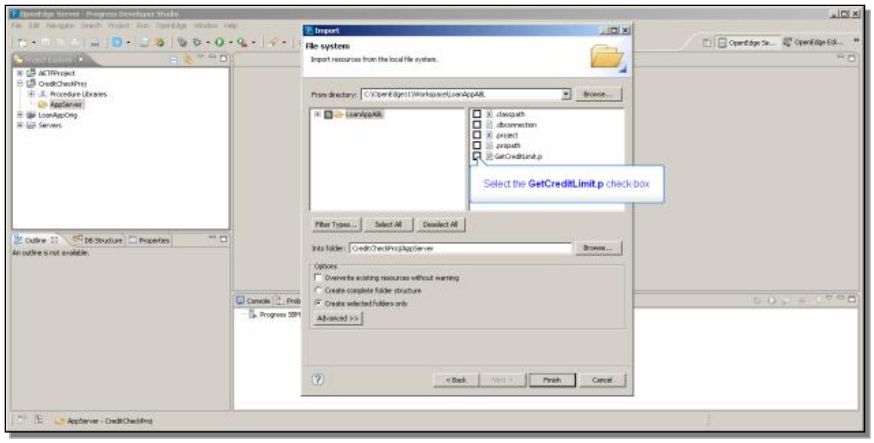
Next I need to set up an ABL procedure to be called from Savvion and define the OpenEdge AppServer connection in Developer Studio. The simple credit check process needs just a single procedure for the Savvion process to call in the OpenEdge AppServer, so I'm going to import that into the project. Remember that Developer Studio will publish the contents of my **AppServer** directory to the **CreditCheckDeploy** target directory, so I start in the AppServer folder:



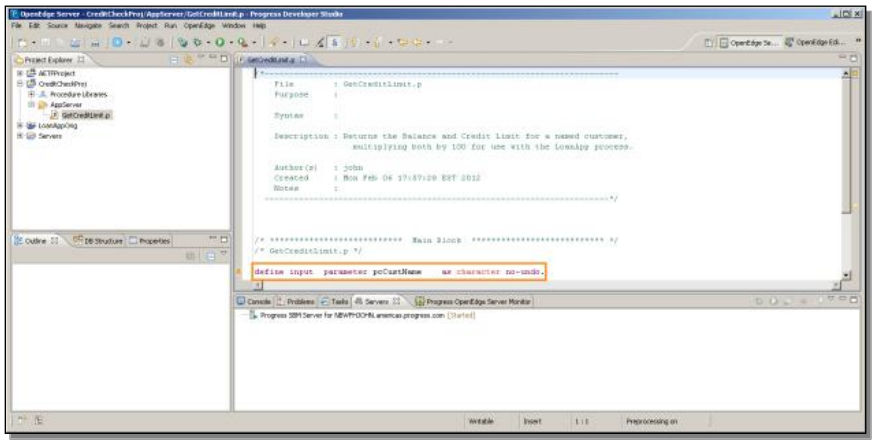
I select **Import**, and **Import** again, and then indicate that I'm importing something from the **File System**:



I select the folder where the ABL procedure is stored away, and select the procedure, **GetCreditCheck.p**:



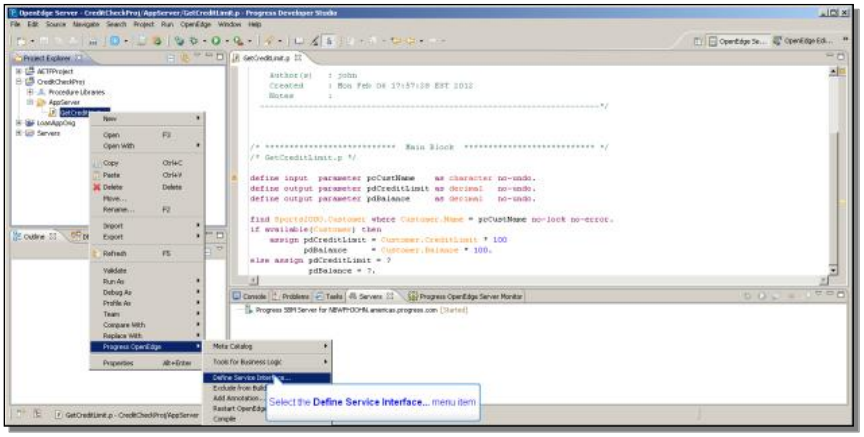
I click Finish, and now I can see the procedure in the AppServer folder. The procedure takes a sports2000 customer name as input:



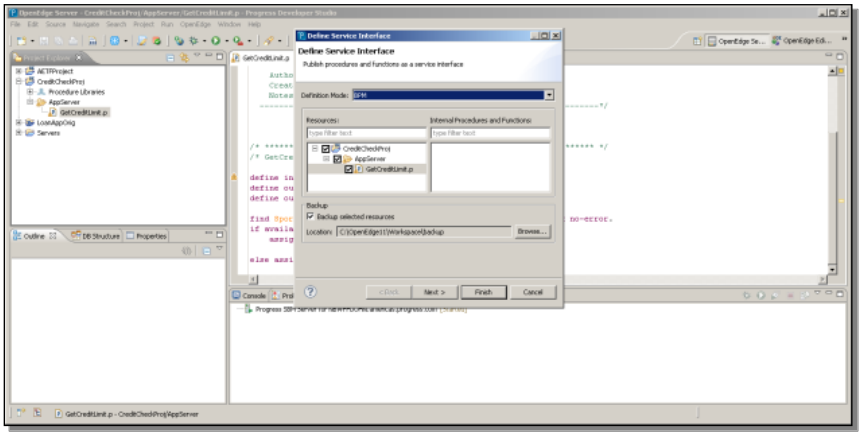
The procedure looks up the Customer name in the database, and returns the credit limit and customer balance values. The procedure has been used in a larger example where higher amounts were called for, so the code multiplies the values by a hundred:

```
/*------------------------------------------------------------------------
    File        : GetCreditLimit.p

    Description : Returns the Balance and Credit Limit for a named customer,
                  multiplying both by 100 for use with the LoanApp process.
  ----------------------------------------------------------------------*/



/* ***************************  Main Block  *************************** */
/* GetCreditLimit.p */

define input  parameter pcCustName    as character no-undo.
define output parameter pdCreditLimit as decimal   no-undo.
define output parameter pdBalance     as decimal   no-undo.

find Sports2000.Customer where Customer.Name = pcCustName no-lock no-error.
if available(Customer) then
    assign pdCreditLimit = Customer.CreditLimit * 100
           pdBalance     = Customer.Balance * 100.
else assign pdCreditLimit = ?
            pdBalance = ?.
```

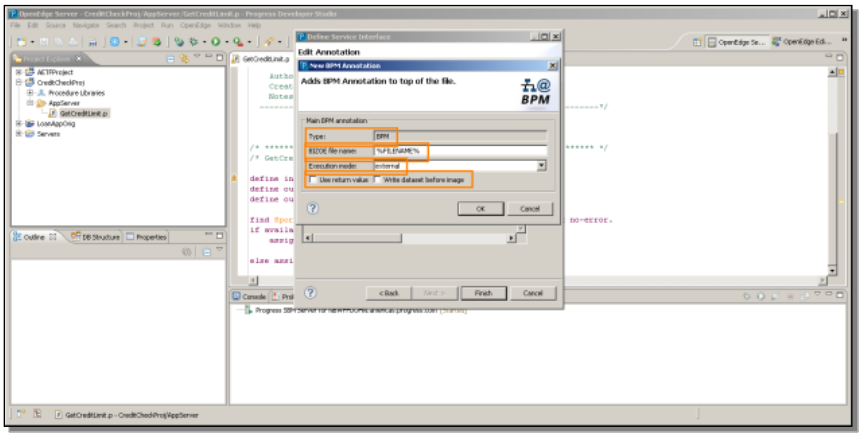
In order to be able to run this procedure from the Savvion process, I need to generate an annotation that goes at the top of the sourse code to define a service interface. So I right-click the procedure, and under **Progress OpenEdge**, select **Define Service Interface**:



The path to the procedure I want to generate the interface for is selected for me. I confirm this and click **Next** to preview the text of the annotation:



I don't need to change anything in the annotation, but I can take a look at it just to point out what some of the components of the annotation are:



The type is **BPM**. In my environment, with OpenEdge and Savvion installed, that's the only option for the type.

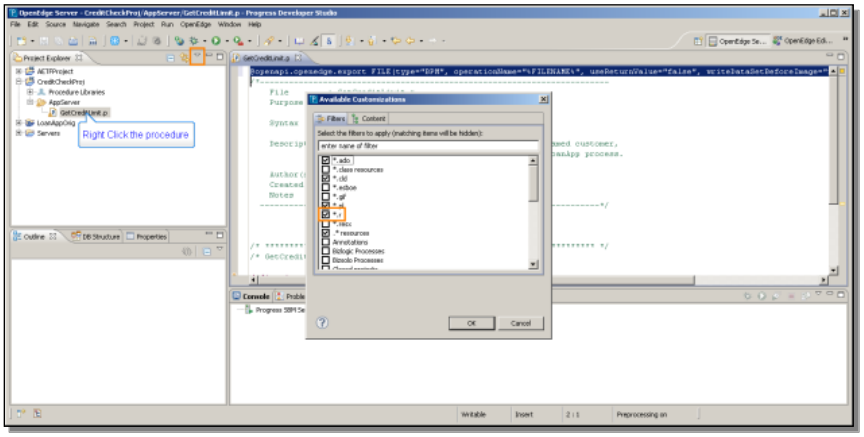
The end result of what I'm doing is that Developer Studio is going to generate a special file containing a description of the parameters of this procedure, which will act as an intermediary between the Savvion environment and the OpenEdge AppServer which will actually execute the ABL procedure. The new file extension for that type of file is **bizoe**, since it connects a BizLogic process to OpenEdge. Think of this file as being much like the WSDL file you would need in order to invoke the ABL procedure as a Web service, as you would have done in OpenEdge 10. By default the .bizoe file will have the same base name as the procedure file, with a different filename extension. That's what **%FILENAME%** indicates, so I leave that alone.

Next, this is an **external** procedure, that is, a stand-alone ABL procedure that I want to run, not an internal procedure inside a persistent procedure instance, so I leave that as well.

Finally, there are a couple of special case checkboxes if the ABL RETURN-VALUE is significant, or if there will be an updated ProDataSet passed back. I don't need those either, so I'm done. If I look at the top of the procedure file itself, I can see the generated annotation in the source code:

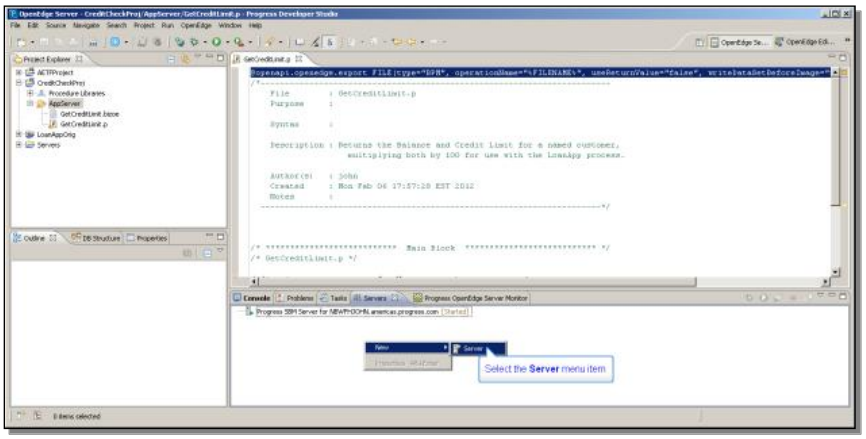
```
@openapi.openedge.export FILE(type="BPM", operationName="%FILENAME%",
useReturnValue="false", writeDataSetBeforeImage="false", executionMode="external").
```

When I save the procedure, it is compiled with the annotation information in the .r file. Note that you can specify which types of filename extensions you want to see in the Project Explorer. If you drop down the Project Explorer **View** menu, and select **Customize View**, you see a list of all the possible filters that can be applied to filenames. Because I have .r checked, they're filtered *out* and I don't see .r files in the Project Explorer. You could uncheck this if you wanted to see .r files. I leave it checked, since the .r file isn't one I would normally want to select in the Explorer:



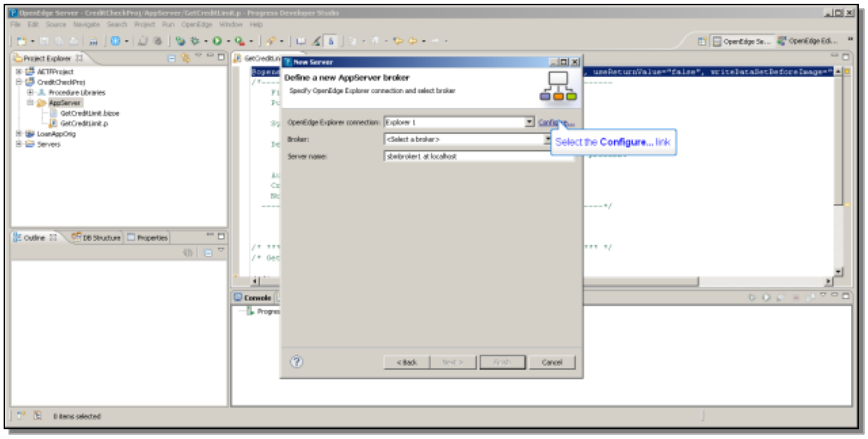
The next step is therefore to generate the bizoe file itself. I right-click on the procedure, and once again, under **Progress OpenEdge**, I select **Generate BPM Invocation Files**.

With the bizoe file generated, the next thing I have to do is to define a connection from Developer Studio to the AppServer that will run the GetCreditCheck procedure. I right-click in the Servers view, and select **New Server**. What I'm defining here is actually not a new server, but a new connection from Developer Studio to that server, so keep that in mind:

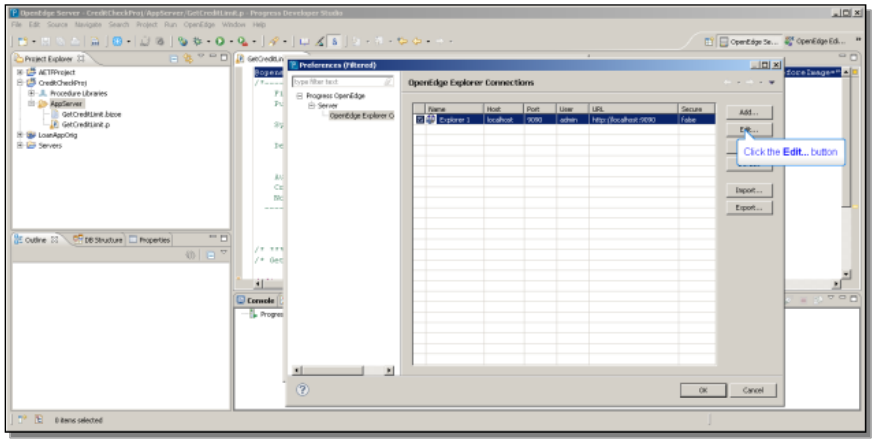


It's an Open Edge AppServer, and I adjust the string that serves as an identifier for it. I put sbmbroker into the name, so that it's clear which AppServer is used for the connection. Next I need to identify the instance of **OpenEdge Explorer** that the

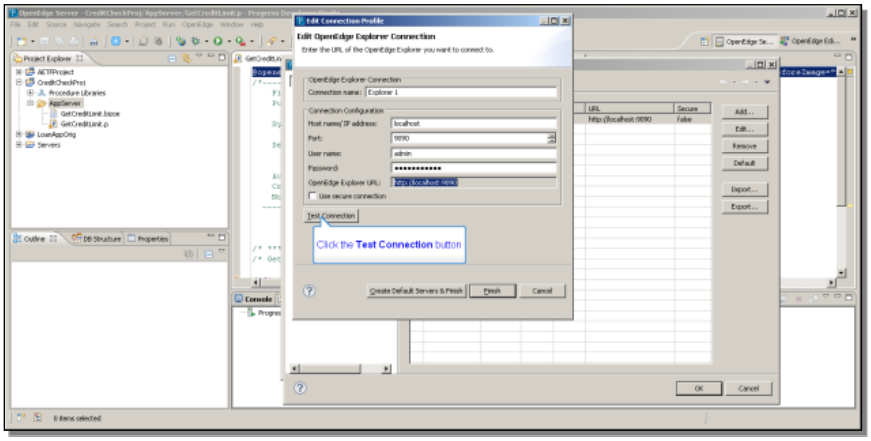
actual server is defined on. There is only got one instance running, but I select **Configure** just so you can see what things are available to be defined:



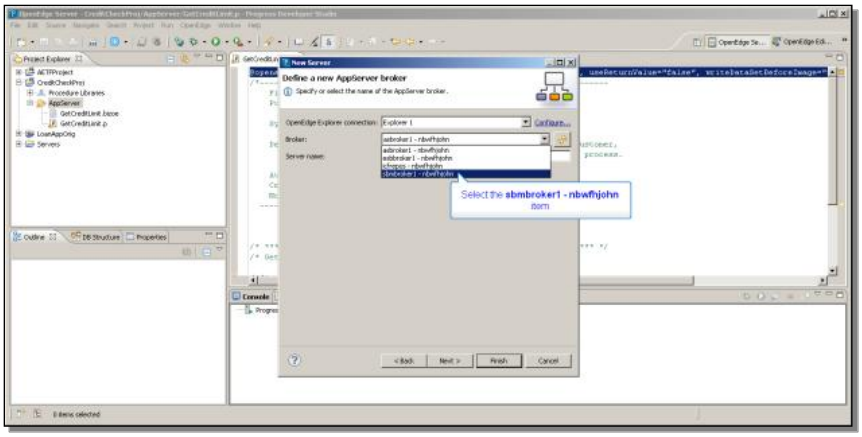
In the Configuration, I select the one instance, **Explorer 1**, and then **Edit**:



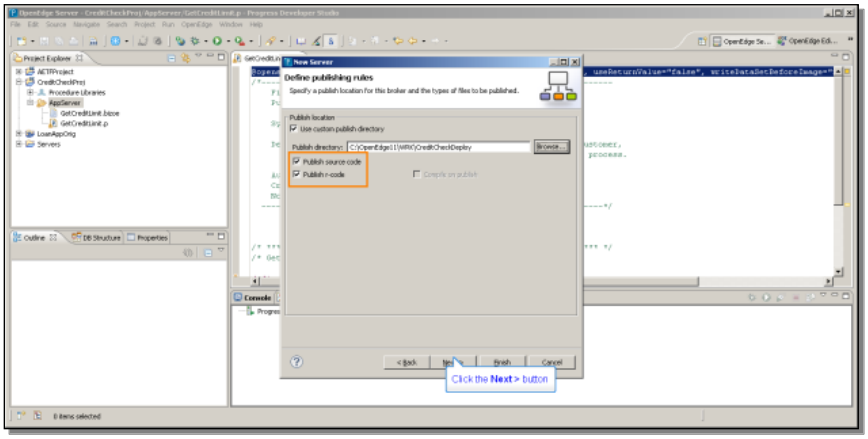
Port number 9090 is the default for OpenEdge Explorer. Then I need to make sure that the administrator username and password are valid. The connection wizard assumes **admin** and **admin** by default, so I select the password field, and replace that with my own admin password. That's the only thing that would be different from the defaults for my environment. Finally, I click **Test Connection** to make sure that Studio can talk to Explorer:



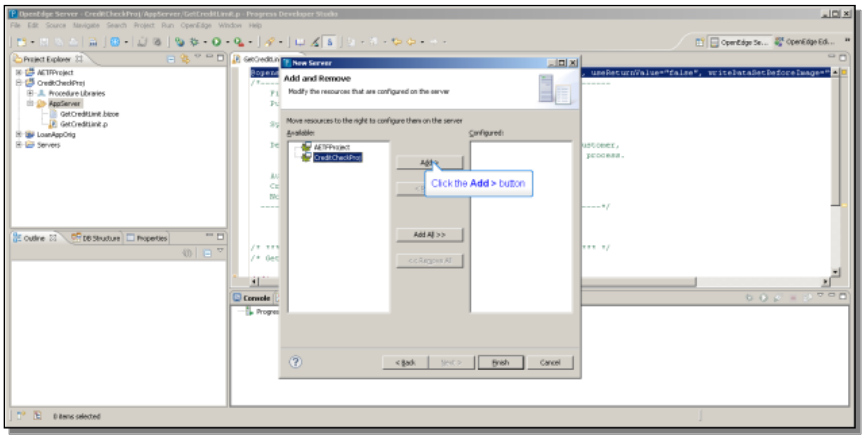
That succeeds, so I'm finished with the Explorer connection definition. Having established that connection, the **New Server** wizard populates the **Broker** list with a list of the AppServer brokers defined in my instance of Explorer. The one I want to use is **sbmbroker1**:



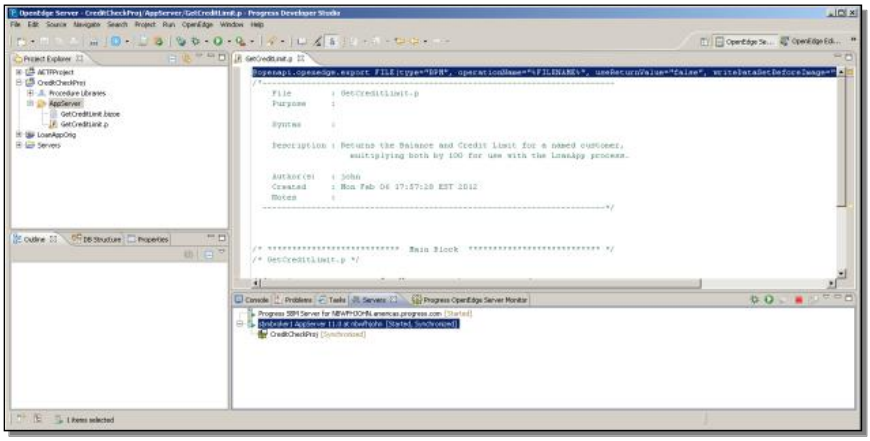
Next I need to define the **publish** directory. This means, when I change ABL code in the local AppServer sub-directory, which I designated as being the source for code destined for the AppServer, where should Developer Studio copy that code to be found for testing? The answer is the directory that I added to the Propath for the sbmbroker1 AppServer, called **CreditCheckDeploy**. Note that you can configure whether you want both source code and r-code published or not. I leave both those options checked, and continue on:



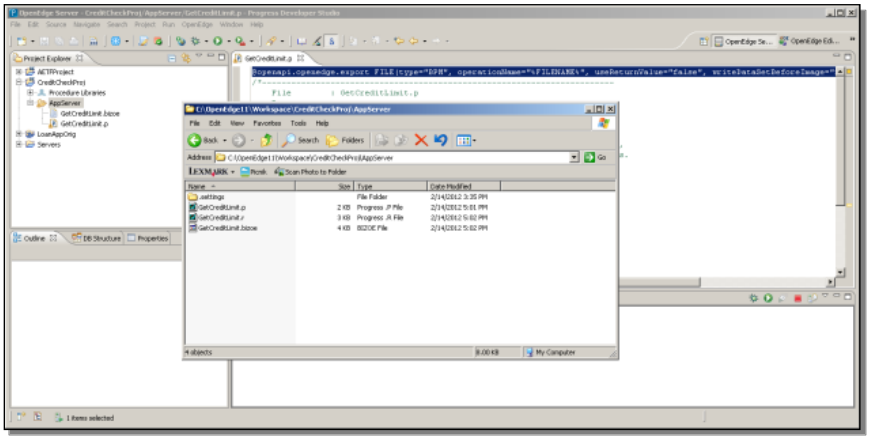
Next I select the sample project to **Add** to the configured projects in Developer Studio, so that code is automatically published as I do development work:



In the Servers view, the broker is shown as stopped, so to synchronize this connection definition with the AppServer I started in Explorer, I right-click on it, and select **Start**. If I expand the broker now, I can see that the CreditCheck server connection is started and synchronized:

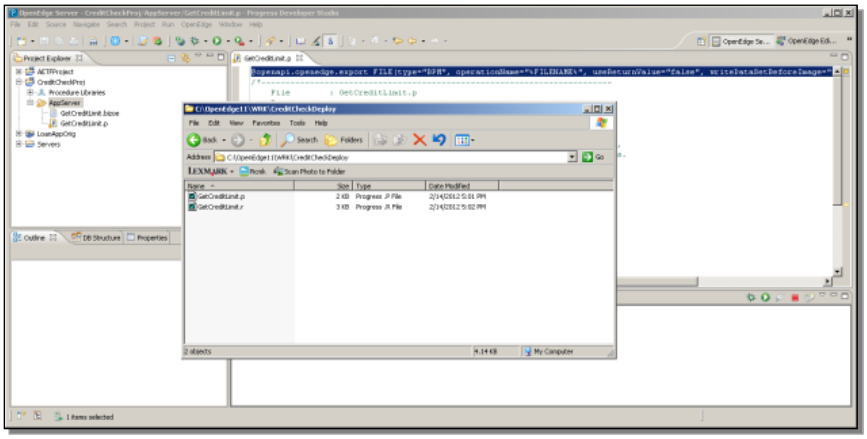


What does it mean that it's synchronized? Here's the AppServer directory under the CreditCheck project:



This is the source directory for the AppServer code. You can see the GetCreditLimit procedure that was imported into the directory, compiled, and the bizoe file that was generated for it to define how it can be run from Savvion.

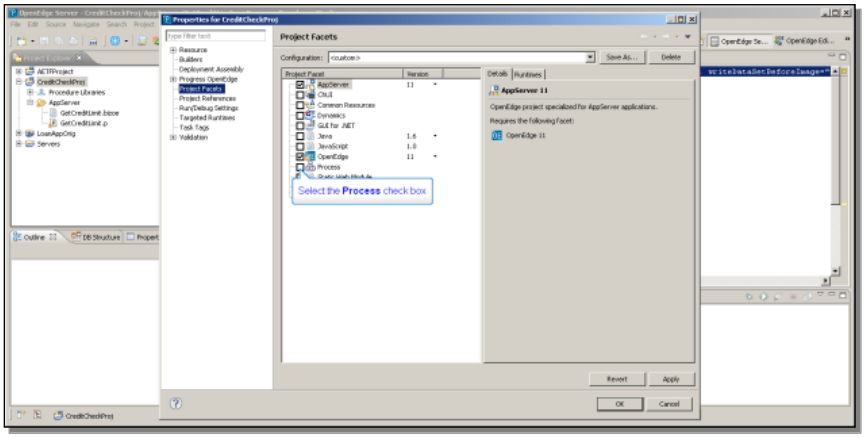
If I switch to the **CreditCheckDeploy** directory, which is the target directory for the project, you can see that the source procedure and the compiled r-code have been copied there. This is the effect of configuring this project in Developer Studio and having it synchronize the code. This directory is in the Propath for the AppServer agent, so it will be found when I do a test that runs the procedure from the Savvion process. Note that the bizoe file is not deployed, because it is used on the client, in this case by the Savvion process, to understand how to run the procedure on the AppServer:



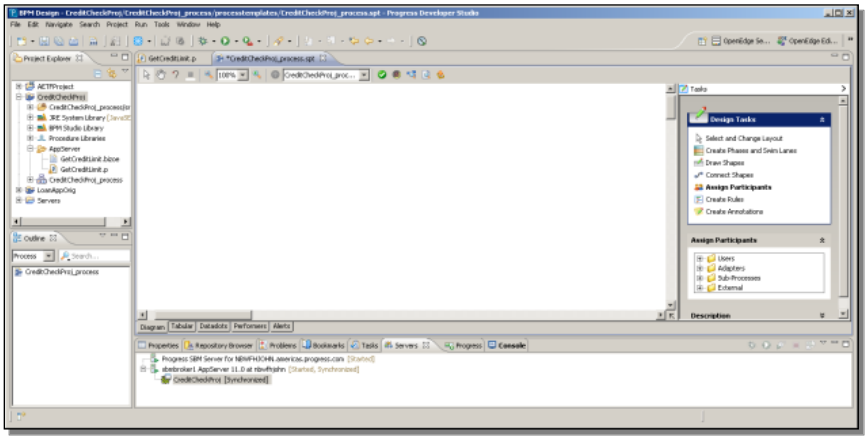
This finishes the configuration work for the project. In the next section of this paper I define the Savvion project and define a call out to the GetCreditLimit procedure using the new Savvion OpenEdge Adapter.

Next I have to turn this OpenEdge project into one that can support a Savvion process design as well. This is the key capability that is enabled by creating a combined Developer Studio environment during the product installation.

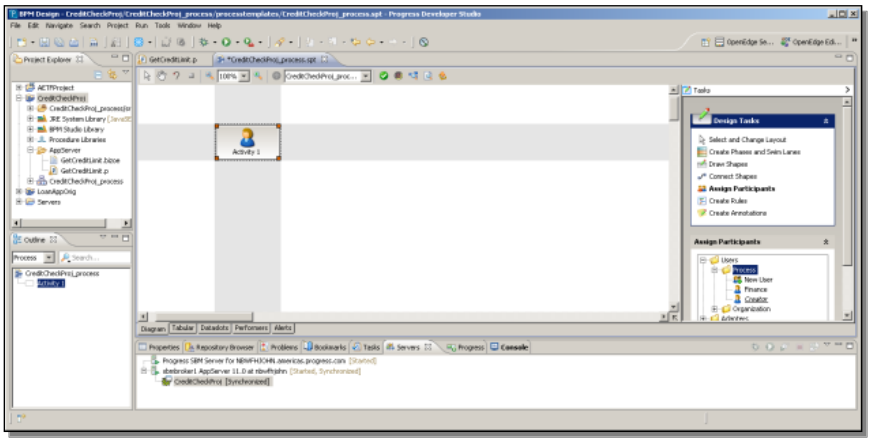
I right-click on the project, and in the **Project Properties**, I select **Project Facet,** and check on the **Process** facet, which adds the capability to define Savvion processes to the project:



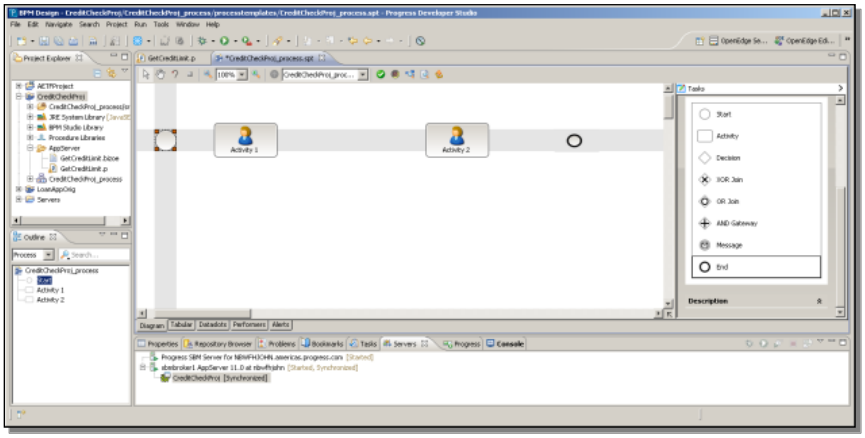
Checking on the Process facet also selects the **Java** facet, which Savvion requires. When I apply this change, Developer Studio opens a new diagram for a Savvion process, represented as a Savvion **process template file** with the extension **spt**:



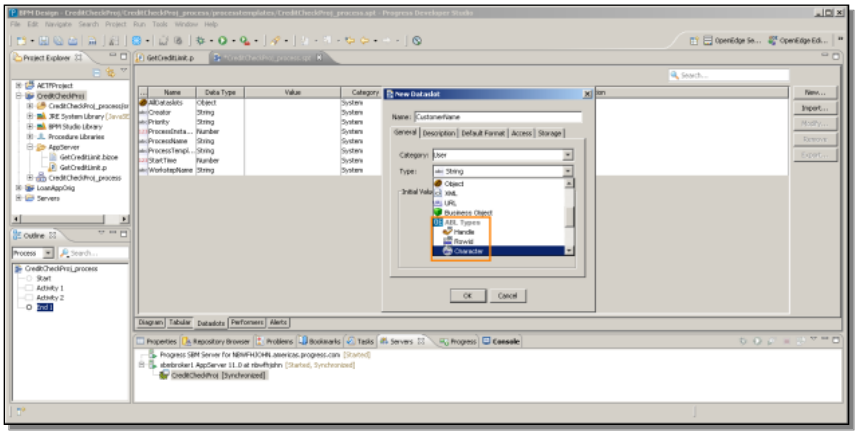
I create one user by dragging **New User** onto the diagram. When I'm prompted, I name the user **Finance**. This creates an Activity for the Finance user:



I then create a second Activity step for the Finance user.These two steps will execute just before and just after the process calls the ABL procedure on the AppServer. I also need a Start step and an End step to make the diagram complete.

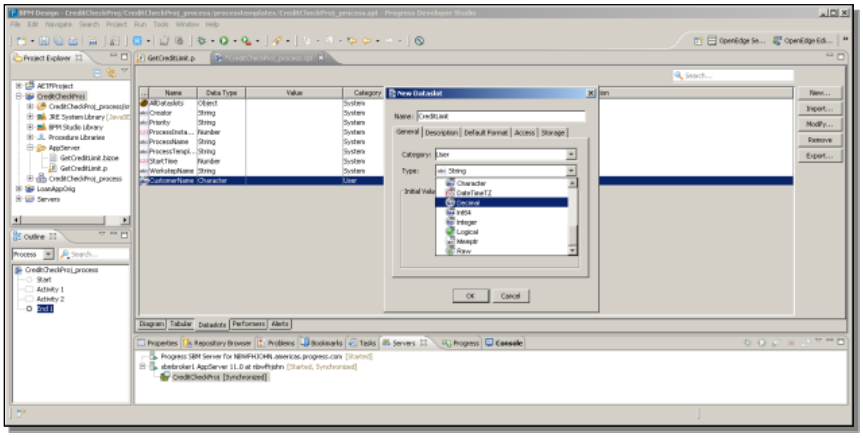


Next the process needs three dataslots, which will map to the three parameters to the procedure GetCreditLimit.p. The first I call **CustomerName**, and it shows part of what is new with the new OpenEdge Adapter support in Savvion 8. In addition to the native Savvion datatypes, there are now dataslot datatypes specific to calling out to OpenEdge. They're listed under the header **ABL Types**. I can select **Character** for this one:

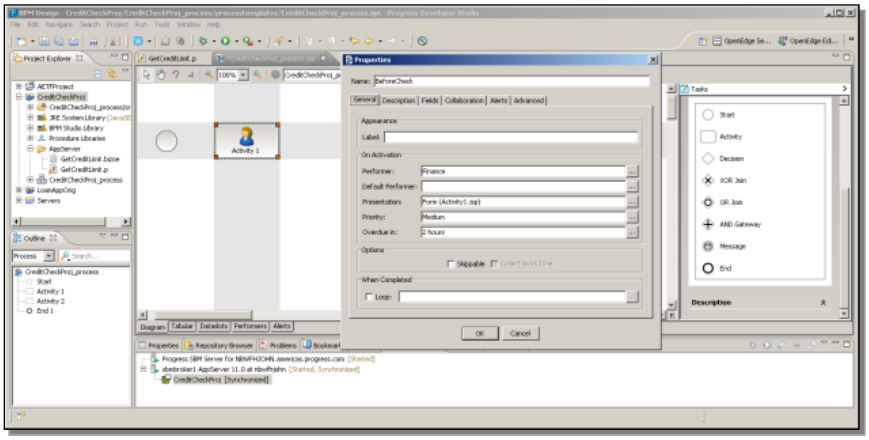


Most of the ABL types map to Savvion types, and in many cases can be used interchangeably, but you should use ABL types for dataslots that will map to ABL procedure parameters to assure type compatibility, and to allow the use of the Null value passed to or from ABL, which the Savvion types don't support.

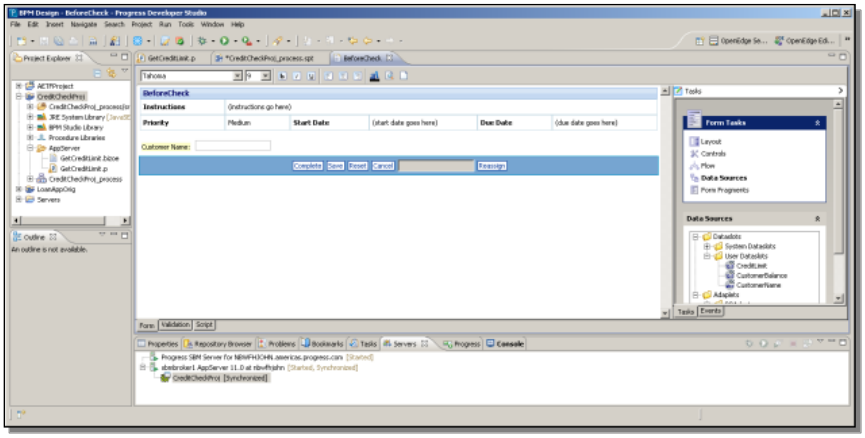
To continue, I create a second dataslot with an ABL Datatype. This one is for the **CreditLimit** output parameter. Looking through the ABL types, I select **Decimal**:



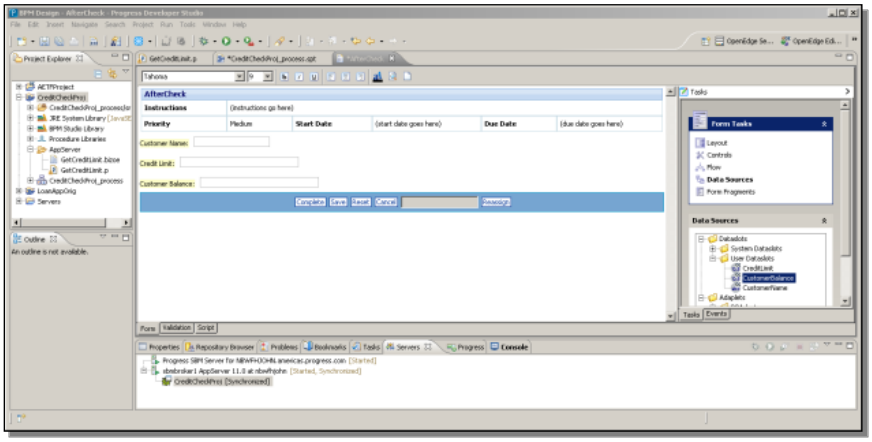
I need one more dataslot for the **CustomerBalance** output parameter. and that is **Decimal** as well. After I create that, I return to the diagram. I want to give more meaningful names to the two Activity steps, so I call the first one **BeforeCheck**:



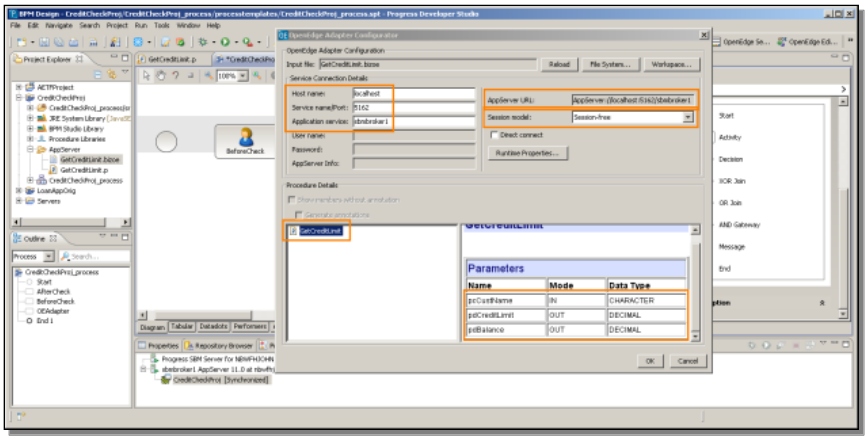
I rename the form for the step as well. Now I need to go into the form and add a dataslot to the form. Under the new User dataslots I select the **CustomerName**. This step will run before I run GetCreditCheck, so I need to be able to enter a customer name that I can then pass into GetCreditCheck.p:



I need to do the same for the other Activity. This will be the **AfterCheck** step. In its form I want to add all three of the ABL dataslots. The **Customer** Name won't change from the Before step, but I let the form display it just to confirm that it has been entered properly. The **CreditLimit** and the **CustomerBalance** will be returned from the procedure, so I need to display those to make sure the call worked properly.



Now I need to create an instance of the new **OpenEdge Adapter** that Savvion 8 supports. The easiest way to do that is just to drag the bizoe file from the Project Explorer onto the process diagram. As soon as I do that, the **OpenEdge Adapter Configurator** opens:

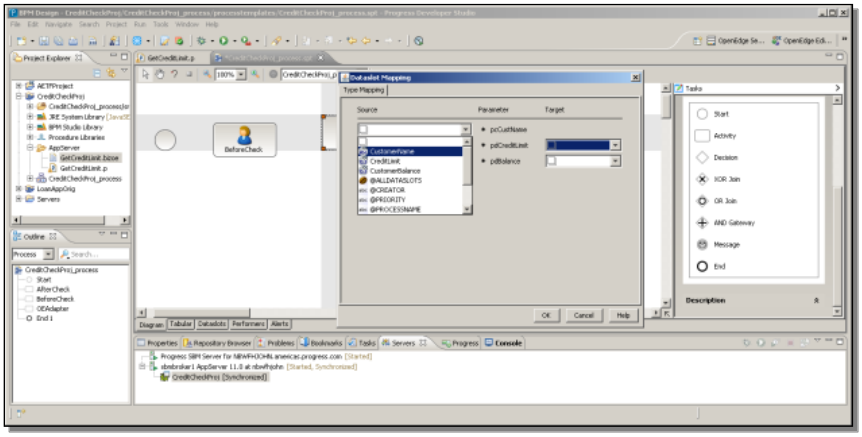


The great thing about this is that all the information needed for the call is embedded in the bizoe file that was generated from the procedure. You can see the **host name**, **port**, and the **AppServer name** displayed. Putting these together gives Savvion the **URL** it needs to use to call out to the AppServer.

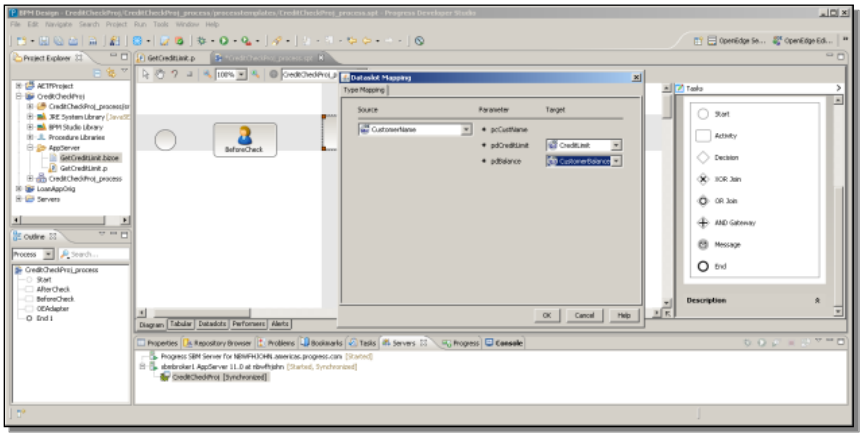
The **session model** is properly displayed as **Session-Free**. The procedure name that it will call is displayed, as are the three parameters to the procedure. Thus, unless I want to change the default configuration that was specified when Developer Studio generated the service interface annotation and the bizoe file, I don't need to change anything here. All the work is done for me.

When I **OK** the configurator, I'm dropped into the dataslot mapping. The one input parameter name is displayed, **pcCustName**, and from the **Source** dropdown on the

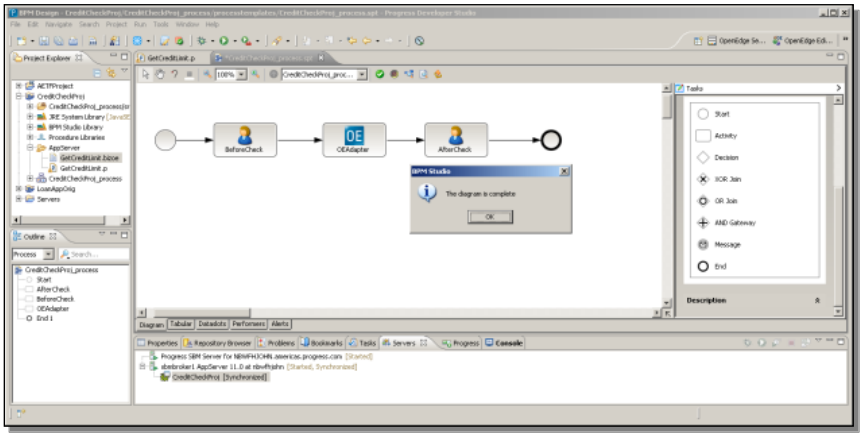
left, I can select the **CustomerName** dataslot, which was defined as type **ABL Character**, to map to the input parameter:



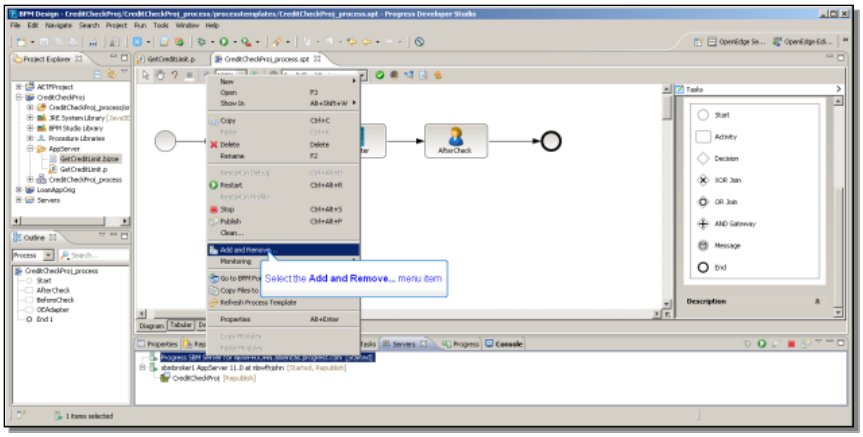
For the output parameters, I map **pdCreditLimit** to the **CreditLimit** dataslot, which is of type ABL Decimal, and for **pdBalance**, I select the **CustomerBalance** Decimal dataslot. Now all the input and output parameters are mapped to the call, so the call is integrated right into the rest of the Savvion process steps:



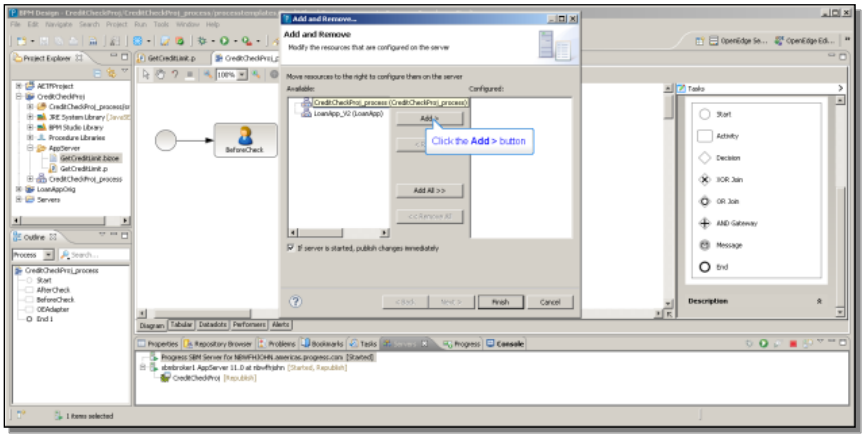
Back in the process diagram, the new step created by dragging the bizoe file onto the diagram has been transformed into an **OpenEdge Adapter** activity in the diagram. All I have to do to finish up is to connect all the steps together. The **Start** step, which just displays the instance name when the process is run, to the **BeforeCheck** step, where I can enter a Customer Name; the **BeforeCheck** step to the **Adapter** step, which will take the CustomerName as input and return the two decimal values; the **Adapter** output to the **AfterCheck** step, which displays the dataslots that are mapped to the output parameters; and the **AfterCheck** step to the **End** step:



Selecting **Check diagram** from the diagram's context menu confirms that the diagram is valid at least, so I save the process. Next I need to tell the SBM server about the process, so I right-click on that server in the Servers View and select **Add and Remove**:

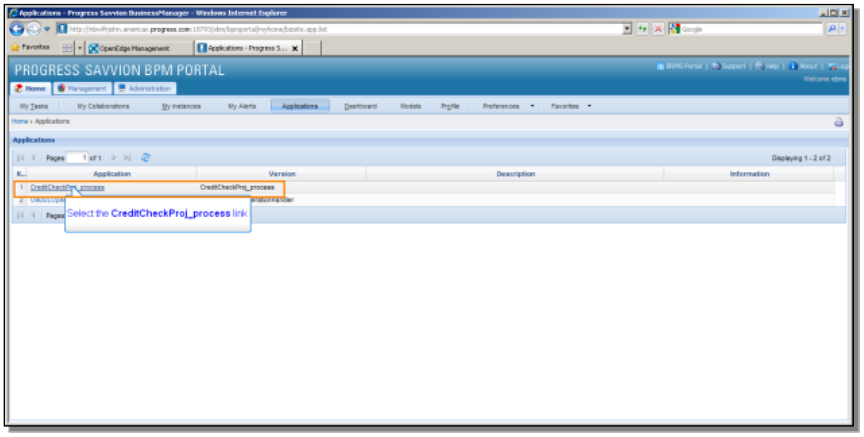


I select the **CreditCheckProj** process and add it to the resources for the SBM server:

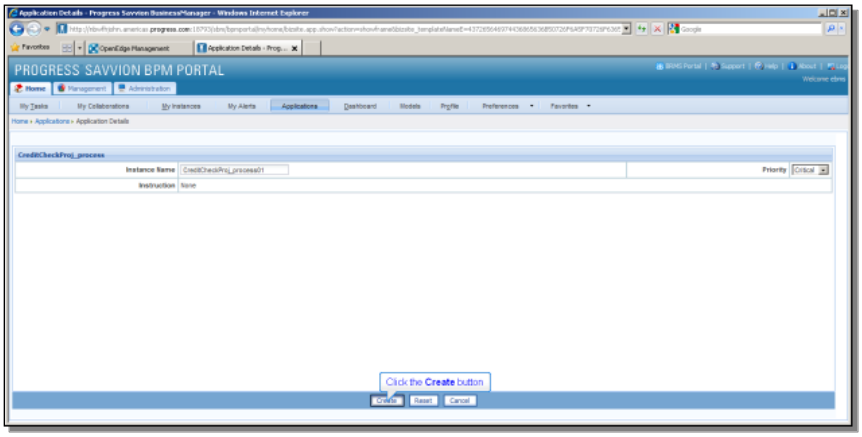


When I click **Finish**, Developer Studio synchronizes the SBM server, which means that the process is automatically deployed to the server.

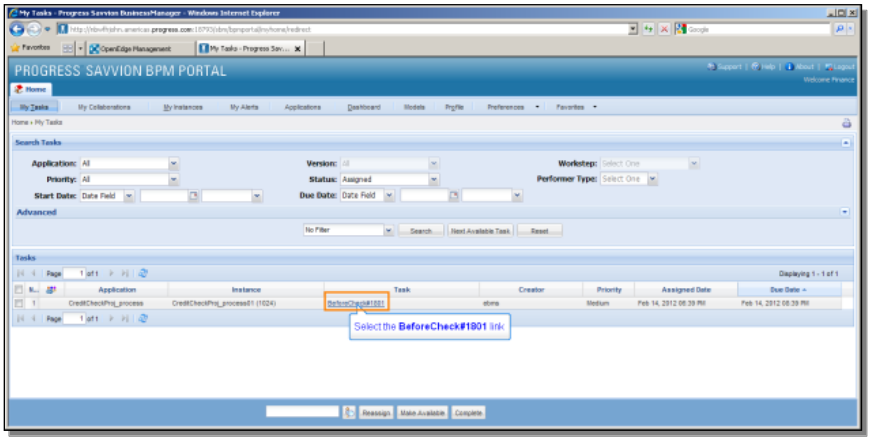
Let's take a look in the SBM Portal to see if it's there. The **CreditCheckProj_process** is installed on the server:



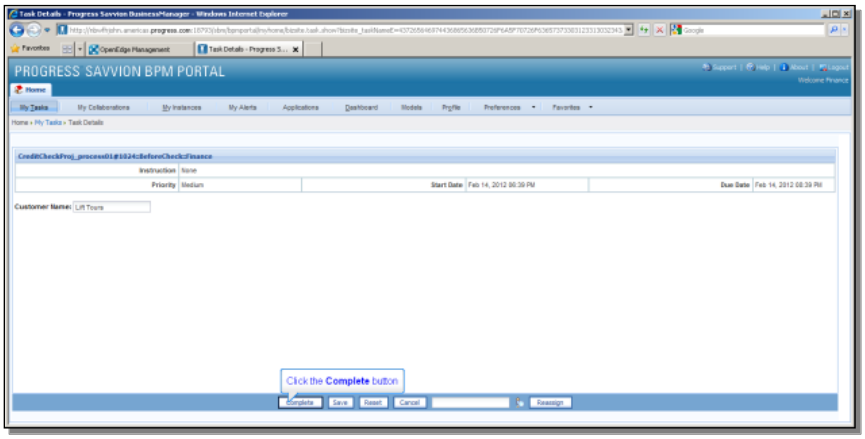
If I select the link for the process, I start up an instance of the new process. The Start step displays a default form with the Instance Name and an Instruction field. I can add a distinctive number to the end of the Instance Name, and **Create** the new instance:



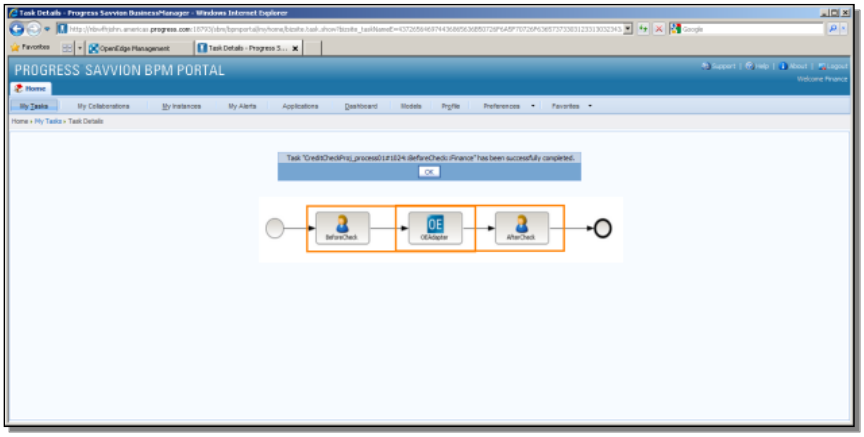
Remember that the activities in the process are assigned to the Finance user, so I have to log out and log back in to the SBM Portal as Finance, to see the first task assigned to Finance, the **BeforeCheck** step:



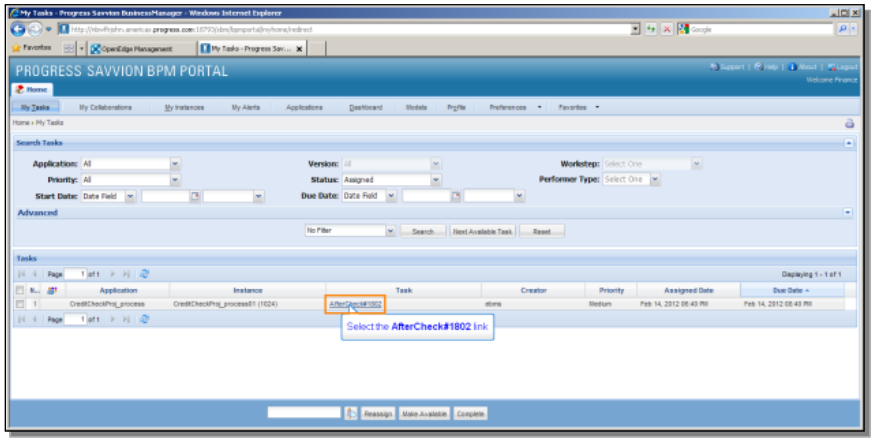
I can enter a valid customer name from the sports2000 database, and complete the step:



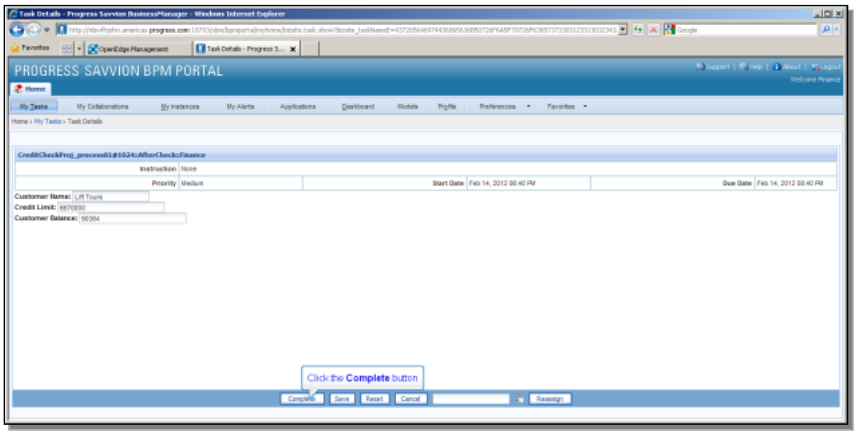
Now what should happen next? The **BeforeCheck** step should proceed to the **OpenEdge Adapter** step, which uses the customer name dataslot to populate the input parameter to GetCreditCheck.p. The procedure is run and returns the two decimal values, which are then transferred to the ABL Decimal dataslots. Finally **AfterCheck** should run to display them:



The Finance user now has an AfterCheck task to run, so I select that:



This form displays the data values returned from OpenEdge for Lift Tours. (Remember that they've been multiplied by 100.) I complete the step and my process is done:



That completes this exercise in combining the OpenEdge and Savvion Developer Studio environments, and defining a Savvion process that uses the new Savvion OpenEdge Adapter to make a call out to an ABL procedure, making the definition of the call as simple as it could be.