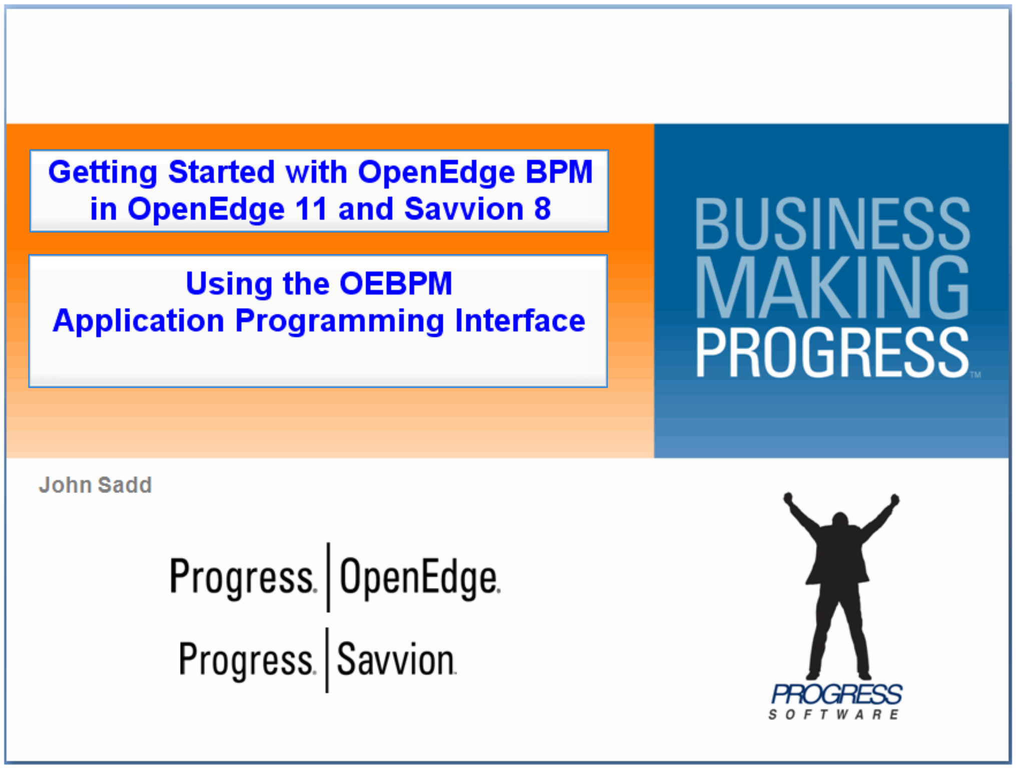
USING THE OEBPM API

John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
May 2012



Getting Started with OpenEdge BPM in OpenEdge 11 and Savvion 8

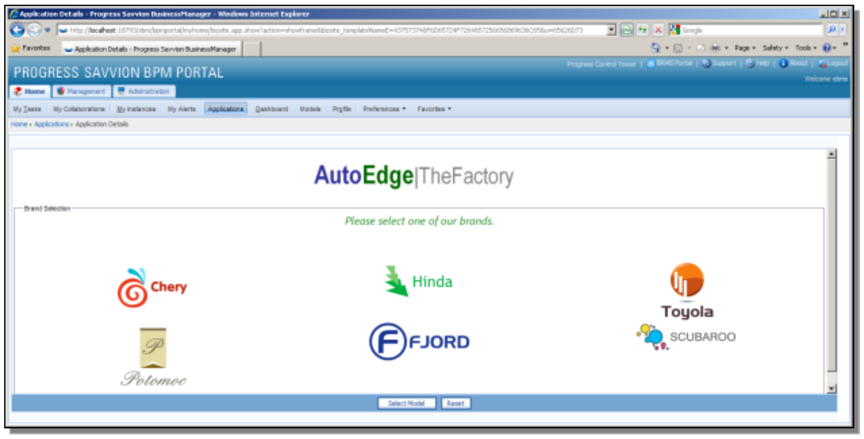
Using the OEBPM Application Programming Interface

BUSINESS MAKING PROGRESS™
John Sadd
Progress. | OpenEdge.
Progress. | Savvion.
PROGRESS SOFTWARE

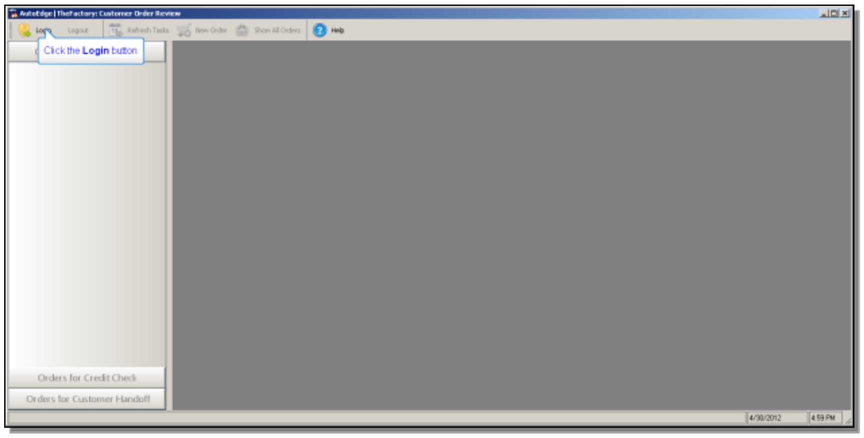
DISCLAIMER

This paper accompanies the final two parts in a series of presentations on ***Getting Started with OpenEdge BPM using Progress OpenEdge release 11 and Savvion version 8***. There's a new Application Programming Interface in OpenEdge 11 that lets you use methods in a set of built-in ABL classes to make calls to Savvion from an OpenEdge application, to create process instances, retrieve and set DataSlot values, and other operations. This paper introduces you to that API.

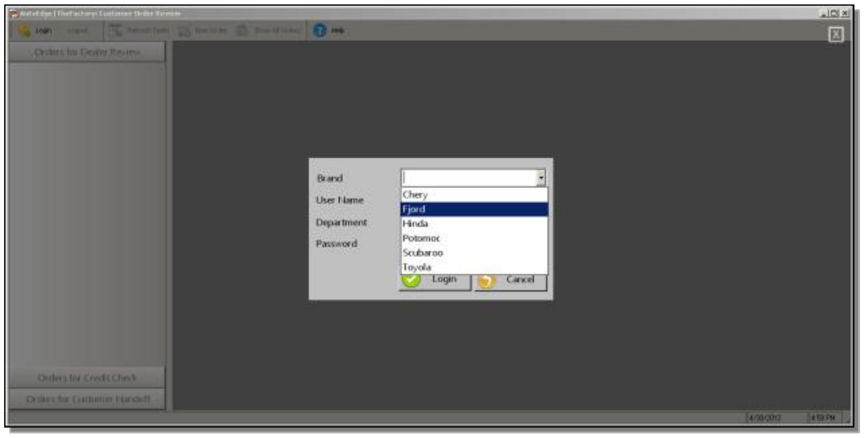
You've seen the AutoEdge The Factory application, which is the AutoEdge module for showing OpenEdge BPM. In that application, a customer walks through a series of web forms to request a new car, and a Savvion process is created to support the delivery of the car. All the forms are defined in Savvion. The customer web forms can be run in the SBM Portal or run over the Internet:



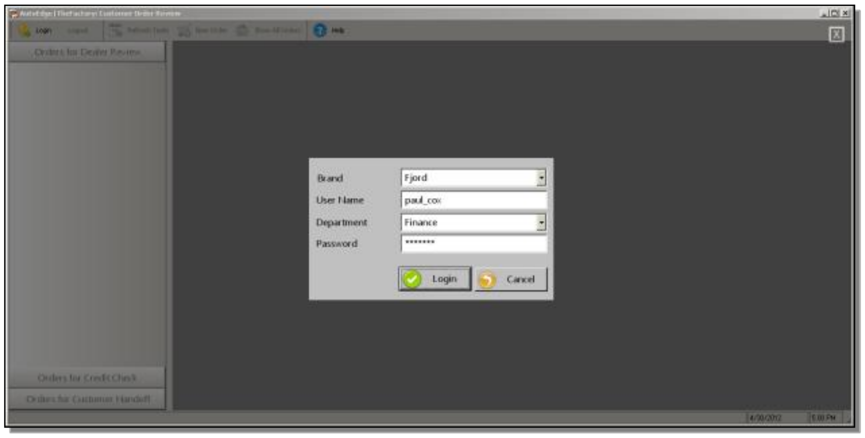
The support forms used by the finance, car sales, and support staff can also be run in the Savvion BPM Portal. However, you may want to create a user interface independent of Savvion, integrated into your OpenEdge application, and control a Savvion process from there. That's what the new API allows you to do. Below is a window built using the OpenEdge support for GUI for .NET. Peter Judge, of the OpenEdge Best Practices group, who created the Auto Edge The Factory module, has expanded the sample GUI for .NET screen in that application into a very nice application that runs the entire process from an ABL GUI for .NET user interface. Let's take a look at how that works. First I log in.



Every car brand has its own support staff, so I choose one of the recognized brands, Fjord.



Then I enter the user name. I start out logged in as a member of the Finance department, because the first task assigned by the Savvion process will be assigned to Finance. I enter a password for the user, and I log in:



Brand	Fjord
User Name	paul_cox
Department	Finance
Password	*******

Because I logged in as a finance person, **Orders for Credit Check** is highlighted, since that’s the Savvion process task assigned to finance people. This is where Paul Cox can look to see what credit check tasks have been assigned to him. But there are no active orders at the moment, so someone has to create one. I let Paul do that next by clicking on the **New Order** button.

Before I do that, I want to show some selected bits of the code that lies behind this application module to show how it's using the API to communicate with Savvion. The ABL procedure that starts the GUI for .NET user interface is called **start_desktop_ui.p**. Even this one example has a substantial amount of support code, so I'm not going to make any attempt to show all of what it's doing. I just point out some examples of API calls that talk to Savvion. The startup procedure creates an instance of the **DealerReview** class, and runs its **Initialize** method, passing in localhost as a parameter:

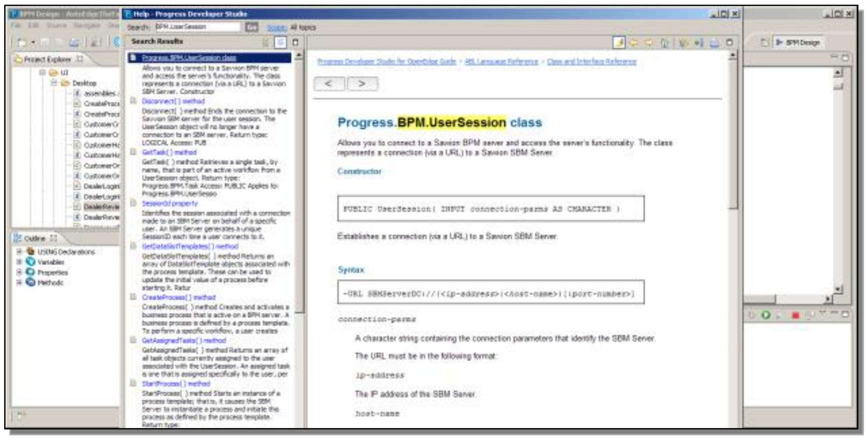
```
oDealerReview = new DealerReview().
oDealerReview:Initialize('localhost').
```

The name DealerReview is really a holdover from the original sample application, where that's the one business function that's implemented in ABL. Here, as you'll see, all the work is done from the ABL app for users in different departments.

DealerReview.cls and the other user interface components are good examples of object-oriented programming in ABL. First you see the new classes that support the interface to Savvion, Progress.BPM.UserSession, Process, Task, and DataSlot:

```
using Progress.BPM.UserSession.
using Progress.BPM.Process.
using Progress.BPM.Task.
using Progress.BPM.DataSlot.
```

You can learn about the specifics in the Reference documentation and the online Help. For example, if you look for **BPM.UserSession**, you can see information on the **UserSession** class, which allows you to connect to a BPM server:



Progress.BPM.UserSession class
Allows you to connect to a Savvion BPM server and access the server's functionality. The class represents a connection (via a URL) to a Savvion SBM Server. Constructor

Disconnect( ) method
Disconnect( ) method Ends the connection to the Savvion SBM server for the user session. The UserSession object will no longer have a connection to an SBM server. Return type: LOGICAL Access: PUB

GetTask( ) method
GetTask( ) method Retrieves a single task, by name, that is part of an active workflow from a UserSession object. Return type: Progress.BPM.Task Access: PUBLIC Applies to: Progress.BPM.UserSessio

SessionId property
Identifies the session associated with a connection made to an SBM Server on behalf of a specific user. An SBM Server generates a unique SessionID each time a user connects to it.

GetDataSlotTemplates( ) method
GetDataSlotTemplates( ) method Returns an array of DataSlotTemplate objects associated with the process template. These can be used to update the initial value of a process before starting it. Retur

CreateProcess( ) method
CreateProcess( ) method Creates and activates a business process that is active on a BPM server. A business process is defined by a process template. To perform a specific workflow, a user creates

GetAssignedTasks( ) method
GetAssignedTasks( ) method Returns an array of all task objects currently assigned to the user associated with the UserSession. An assigned task is one that is assigned specifically to the user, per

StartProcess( ) method
StartProcess( ) method Starts an instance of a process template; that is, it causes the SBM Server to instantiate a process and initiate the process as defined by the process template. Return type:

Progress Developer Studio for OpenEdge Guide > ABL Language Reference > Class and Interface Reference

Progress.BPM.UserSession class

Allows you to connect to a Savvion BPM server and access the server's functionality. The class represents a connection (via a URL) to a Savvion SBM Server.

Constructor

```
PUBLIC UserSession( INPUT connection-parms AS CHARACTER )
```

Establishes a connection (via a URL) to a Savvion SBM Server.

Syntax

```
-URL SBMServerDC://[<ip-address>|<host-name>][:port-number>]
```

`connection-parms`

A character string containing the connection parameters that identify the SBM Server.

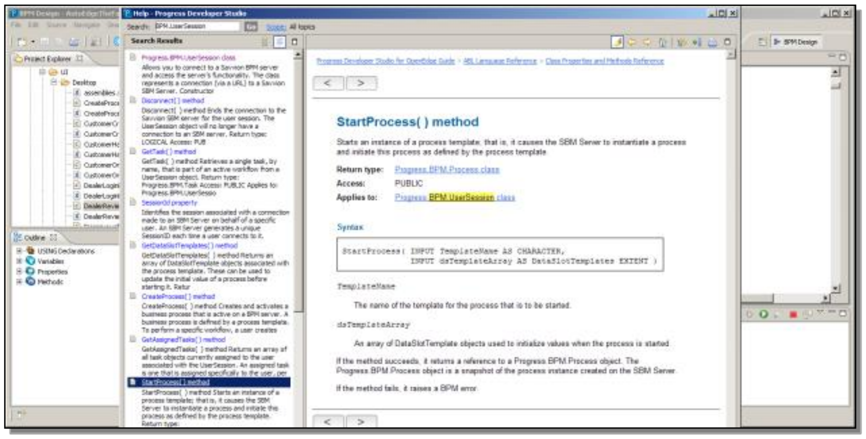
The URL must be in the following format:

`ip-address`

The IP address of the SBM Server.

`host-name`

And you can look at the specific methods within a class, like **StartProcess**, which starts an instance of a process template.



Progress Developer Studio for OpenEdge Guide > ABL Language Reference > Class Properties and Methods Reference

StartProcess( ) method

Starts an instance of a process template; that is, it causes the SBM Server to instantiate a process and initiate this process as defined by the process template

Return type:	Progress.BPM.Process class
Access:	PUBLIC
Applies to:	Progress.BPM.UserSession class

Syntax

```
StartProcess( INPUT TemplateName AS CHARACTER,
              INPUT dsTemplateArray AS DataSlotTemplates EXTENT )
```

TemplateName

The name of the template for the process that is to be started.

dsTemplateArray

An array of DataSlotTemplate objects used to initialize values when the process is started.

If the method succeeds, it returns a reference to a Progress.BPM.Process object. The Progress.BPM.Process object is a snapshot of the process instance created on the SBM Server.

If the method fails, it raises a BPM error.

Consult the documentation and online Help for all the details on what I show in overview form in this paper.

Let's look at that **Initialize** method that the start procedure runs. I just highlight individual lines of code as I go here. Initialize creates an AppServer and connects to it:

```
create server AppServer.
AppServer:connect(substitute('-AppService asAutoEdgeTheFactory -H &1 -sessionModel
  session-free', pcServerHost)).
```

On the one hand, this client OpenEdge application is going to run business logic on the AppServer. The same service procedures that you've seen before being run as Web services or as OpenEdge Adapter steps from Savvion, can now be run from the ABL client application itself. Then the code creates a new **UserSession** object, just as you saw documented in the Help topic we just looked at:

```
this-object:UserSession = new UserSession(substitute('-URL SBMServerDC://&1:18793',
    pcServerHost)).
```

The special URL **SBMServerDC**, which stands for Direct Connect, connects to the server on the designated port. So the ABL client application is able to call out to OpenEdge using an AppServer connection on the one hand, and to Savvion using a UserSession object on the other.

Now let's look at the **LoginUser** method. It first runs an ABL serveice called **service_employeelogin.p**, which validates the user credentials against a database and creates a Client-Principal object for the user:

```
run service_employeelogin.p on AppServer (
                    input  pcBrand,
                    input  pcUserName,
                    input  pcDepartment,
                    input  pcPassword,
                    output UserContextId,
                    output DealerCode,
                    output DealerName,
                    output SalesrepCode).
```

Then the method sets the **BpmUserName** property of the DealerReview object to the department username passed in.

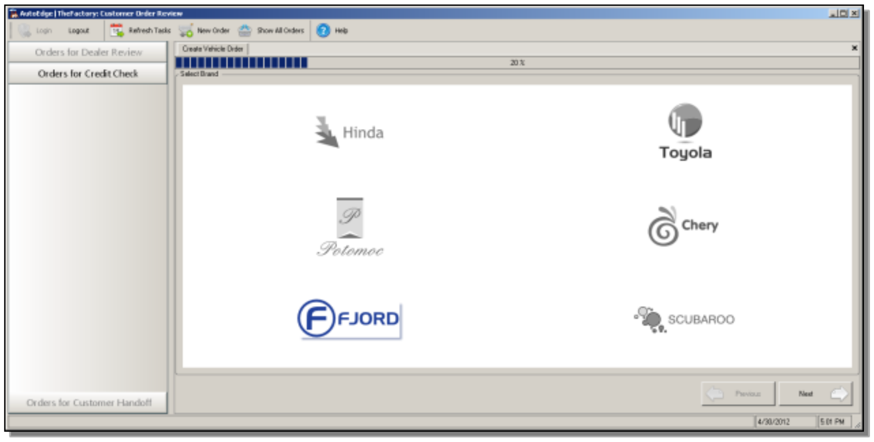
```
                case pcDepartment:
                    when 'sales'   or
                    when 'support' or
                    when 'manager' or
                    when 'finance' then this-object:BpmUserName = lc(pcDepartment).
                    when '_system' then this-object:BpmUserName = pcUserName.
                    otherwise           this-object:BpmUserName = ''.
                end case.
```

Then it connects to Savvion through the UserSession object with that username and password. Now the method has a connection to Savvion just as if a user had logged into the BPM Portal.

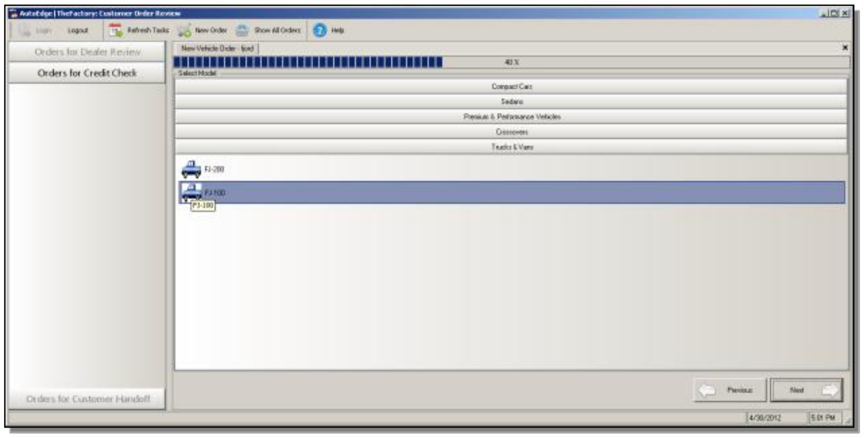
```
if this-object:BpmUserName ne '' then
    this-object:UserSession:Connect(this-object:BpmUserName, pcPassword).
```

That's a summary of the code that got executed when I logged in as a Finance user.

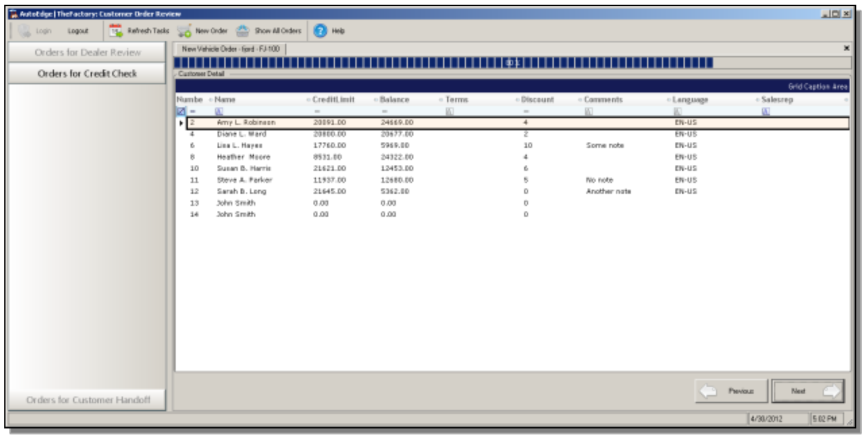
Now in order for that user to have any tasks to complete, someone has to create an order. This operation is really independent of which department and user is logged in, but the user interface requires a login before the New Order button is enabled, so I start there. Here you can see that the web forms user interface that was originally defined as a Savvion bizsolo sequence has been redefined using a .NET GUI as part of the ABL application UI, all contained within this one window:



As always, I like Fjords. Next I choose a vehicle model, just as the customer would in the bizsolo interface:

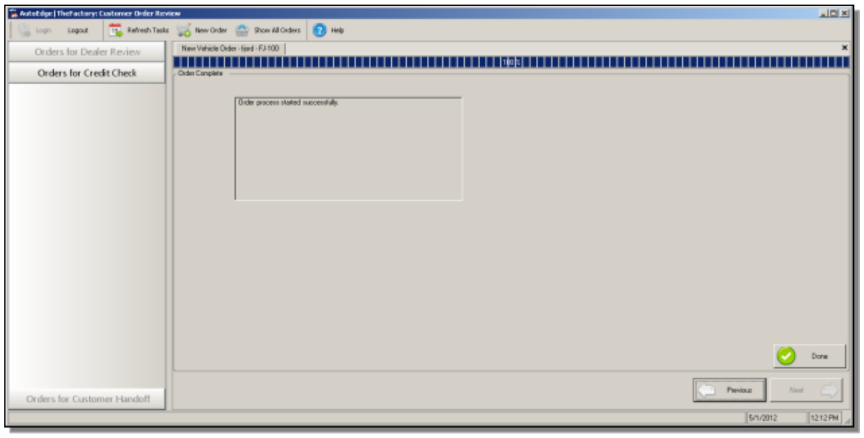


I can also choose some options. Next, the presumption is that someone associated with the business is taking the order; that's why this is executing as a desktop application UI. So I choose a user from the list of known customers:



Numbe	Name	CreditLimit	Balance	Terms	Discount	Comments	Language	Salesrep
2	Amy L. Robinson	20891.00	24669.00		4		EN-US	
4	Diane L. Ward	20800.00	20677.00		2		EN-US	
6	Lisa L. Hayes	17760.00	5969.00		10	Some note	EN-US	
8	Heather Moore	8531.00	24322.00		4		EN-US	
10	Susan B. Harris	21621.00	12453.00		6		EN-US	
11	Steve A. Parker	11937.00	12680.00		5	No note	EN-US	
12	Sarah B. Long	21645.00	5362.00		0	Another note	EN-US	
13	John Smith	0.00	0.00		0			
14	John Smith	0.00	0.00		0			

Now the order definition is complete. The ABL application has created a Savvion order process. Let's take a look at how that was done.



Order process started successfully.

The code for pretty much the entire set of forms for placing an order is in **SubmitOrderForm.cls**. The **Activate_OrderComplete** method handles the final step of getting the process started after all the data has been entered. The important thing this method does is to run another ABL service to start a vehicle order process.

```
run service_startorderprocess.p on hAppServer (
                        this-object:VehicleBrand,
                        cSelectedVehicleModel, /*mcVehicleModel,*/
                        mcModelName,
                        eCustomer.CreditLimit,
                        cCustomerEmail,
                        string(eCustomer.Number),
                        eCustomer.Name,
                        this-object:DealerCode,
                        uxPaintColour:Value:ToString(),
                        uxMoonroof:Value:ToString(),
                        uxWheels:Value:ToString(),
                        cSelectedInteriorAccessories,
                        uxTrimMaterial:Value:ToString(),
                        uxTrimColour:Value:ToString(),
                        output cError).
```

This might seem a little indirect, because the ABL procedure is going to turn around and send a call back to Savvion to start the process, but if some of the data being passed in these parameters is also updating an OpenEdge database, then this could be an appropriate way to get things started.

Let's take a look at that procedure. It's in the AppServer folder, of course.

You can see the reference to **Progress.BPM.UserSession** at the top:

```
using Progress.BPM.UserSession.
```

This sample procedure just uses a hardcoded username and password to simplify the example, so the process is started on behalf of the **manager** user, not the user who happened to be logged into the GUI application. The name of the Savvion process is **VehicleOrderProcess**. You've seen this before. This is the same Savvion process that runs with Savvion forms in the BPM portal. The code creates another UserSession object. Remember that the first new UserSession we saw is for the session the finance person, Paul Cox, is logged into, and which he will use to take care of his Credit Check tasks. This is a separate session used to start the Savvion process that generates those tasks. Then the code connects to the Savvion session using manager and the manager's password:

```
cBPMProcessName = 'VehicleOrderProcess'.
cBPMPassword = 'letmein'.
cBPMUserName = 'manager'.

oUserSession = new UserSession('-URL SBMServerDC://localhost:18793').
oUserSession:Connect(cBPMUserName, cBPMPassword).
```

Next the code uses the **GetDataSlotTemplates** method in the API to get back an object that holds a list of all the DataSlots for that process. Think of this as being rather like an OpenEdge buffer, one set of values with their own individual names and datatypes:

```
oDataSlotTemplate = oUserSession:GetDataSlotTemplates(cBPMProcessName).
```

The iLoop block loops through the list, and assigns each Dataslot the right initial value from all the parameters passed in, which of course represent the data that was filled in on behalf of the customer ordering the vehicle.

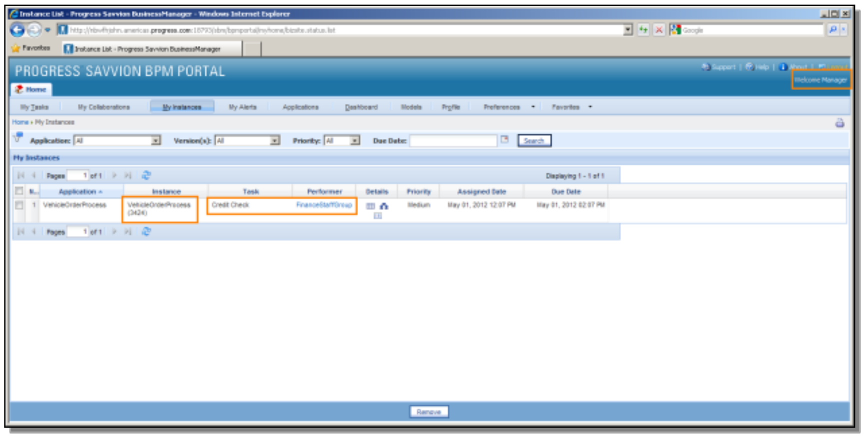
```
do iLoop = 1 to iMax:
    case oDataSlotTemplate[iLoop]:name:
        when 'CustomerCreditLimit' then oDataSlotTemplate[iLoop]:Value  =
                  pcCustomerCreditLimit.
        when 'CustomerEmail' then oDataSlotTemplate[iLoop]:Value  =
                  pcCustomerEmail.
        when 'CustomerId' then oDataSlotTemplate[iLoop]:Value  = pcCustomerId.
        when 'CustomerName' then oDataSlotTemplate[iLoop]:Value  = pcCustomerName.
        when 'DealerCode' then oDataSlotTemplate[iLoop]:Value  = pcDealerCode.
        when 'ModelName' then oDataSlotTemplate[iLoop]:Value  = pcModelName.
        …
        when 'VehicleBrand' then oDataSlotTemplate[iLoop]:Value  = pcVehicleBrand.
        when 'VehicleModel' then oDataSlotTemplate[iLoop]:Value  = pcVehicleModel.
        /* only pass values we know about. */
        otherwise
            oDataSlotTemplate[iLoop] = ?.
    end case.
end.
```

Finally, the procedure uses the **StartProcess** method to create an instance of the Savvion vehicle order process, passing in the DataslotTemplate object to initialize all the Dataslot values.

```
oUserSession:StartProcess(cBPMProcessName, oDataSlotTemplate).
```

This gets us through the starting of the vehicle order process.

Let me bring up the BPM Portal for a moment to show you the result of what just happened. I'm logged in as manager. You can see that there's now an instance of the VehicleOrderProcess started by the manager, its instance number is **3424**, the next assigned task is Credit Check, and it's assigned to a member of the Finance group:



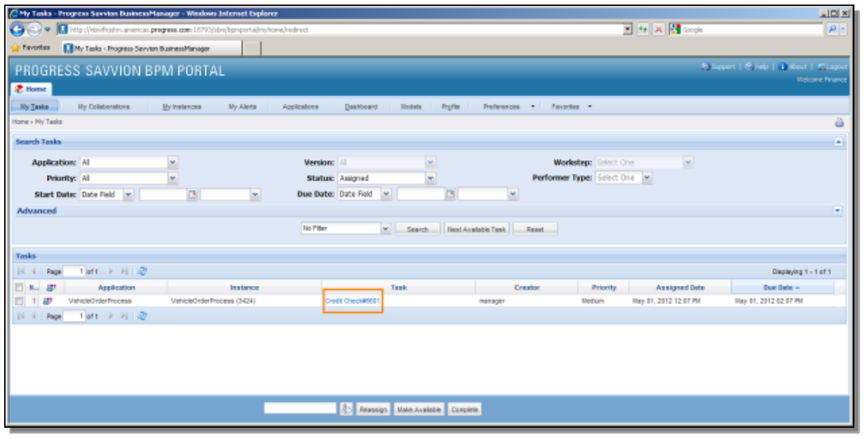
Application: All Version(s): All Priority: All Due Date:

My Instances

Displaying 1 - 1 of 1

N..	Application	Instance	Task	Performer	Details	Priority	Assigned Date	Due Date
1	VehicleOrderProcess	VehicleOrderProcess (3424)	Credit Check	FinanceStaffGroup		Medium	May 01, 2012 12:07 PM	May 01, 2012 02:07 PM

I log back into the Portal as Finance, and here I see the Credit Check task assigned to the finance user.



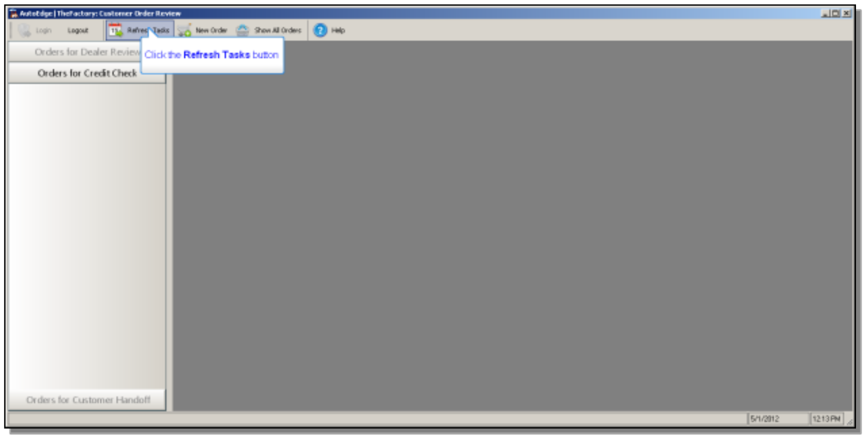
Search Tasks

Application: All Version: All Workstep: Select One
Priority: All Status: Assigned Performer Type: Select One
Start Date: Date Field Due Date: Date Field

Tasks — Page 1 of 1 — Displaying 1 - 1 of 1

N...	Application	Instance	Task	Creator	Priority	Assigned Date	Due Date
1	VehicleOrderProcess	VehicleOrderProcess (3424)	Credit Check#6681	manager	Medium	May 01, 2012 12:07 PM	May 01, 2012 02:07 PM

Back in the GUI application, in order to see the first task associated with that process, Paul Cox clicks the Refresh Tasks button:



Let me show what happens when he presses that button.

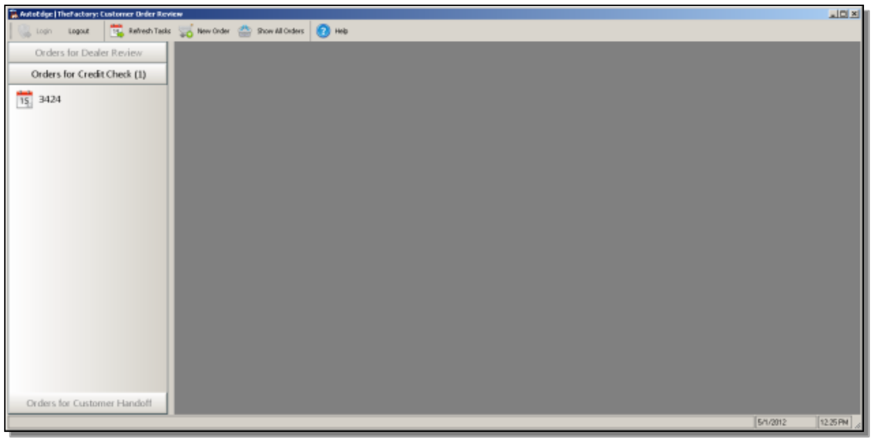
A method named **RefreshTasksList** gets run in DealerReview. Below you can see that the method runs first **GetAssignedTasks** in the UserSession, and then **GetAvailableTasks**. GetAssignedTasks will return an object containing a set of all the tasks that have been assigned to this individual user, and the second call will return a set of tasks that have been allocated to a group the user belongs to, so they are available to be handled by that user.

```
method protected void RefreshTasksList(input pcActivity as character):
        session:set-wait-state('general').

        /* Empty all the existing orders */
        ClearUI(pcActivity).


        PopulateTaskList(this-object:UserSession:GetAssignedTasks(), pcActivity).
        PopulateTaskList(this-object:UserSession:GetAvailableTasks(), pcActivity).

        finally:
            session:set-wait-state('').
        end finally.
    end method.
```

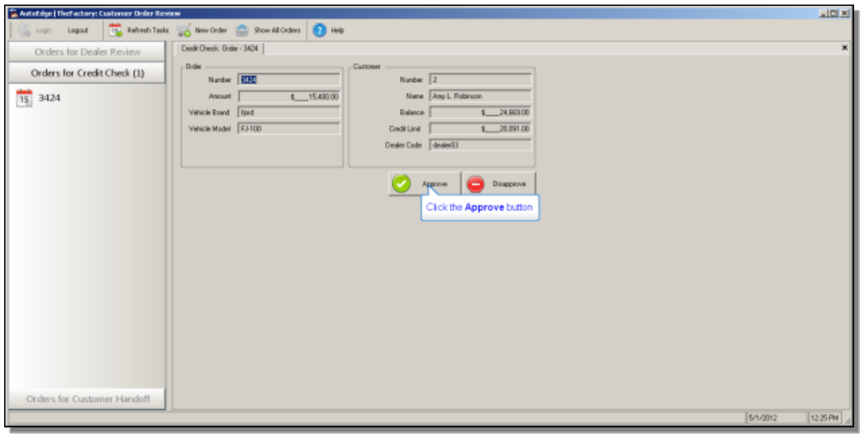
Together, these lists are received by **PopulateTaskList**, and for all VehicleOrderProcess tasks – just in case there might be tasks for some other process that would be handled by another user interface – the code uses the API's **GetDataSlots** method to retrieve a list of the process instance Dataslot values, and walks through those getting specific values out such as the DealerCode and OrderNum. Note that there's no way to directly access a single Dataslot value by name. Currently, you have to walk through the set of values in a Dataslot object to find the ones you're looking for:

```
if poTaskList[iTaskLoop]:name matches 'VehicleOrderProcess*::' + pcActivityName
then
    do:
        oDataSlot = poTaskList[iTaskLoop]:GetDataSlots().
        iDataSlotMax = extent(oDataSlot).
        lAddItem = false.

        do iDataSlotLoop = 1 to iDataSlotMax:
            case oDataSlot[iDataSlotLoop]:Name:
                /* only get tasks for this dealer */
                when 'DealerCode' then
                  lAddItem = (oDataSlot[iDataSlotLoop]:Value eq
                      this-object:DealerCode).
                when 'OrderNum' then
                    assign cOrderNum = string(oDataSlot[iDataSlotLoop]:Value)
                           lAddItem = not (cOrderNum eq ? or cOrderNum eq '')
                when not lAddItem. /* all add conditions must be true */
            end case.
        end.
```

That's what happened when Paul clicked **Refresh Tasks**, so back in the client application, he sees the Credit Check task for instance number 3424, which we saw in the Portal just a moment ago.


AutoEdge | TheFactory: Customer Order Review
Login
Logout
Refresh Tasks
New Order
Show All Orders
Help
Orders for Dealer Review
Orders for Credit Check (1)
3424
Orders for Customer Handoff
5/1/2012
12:25 PM

When Paul selects task 3424, he can see the relevant Order and Customer values for the task. Paul can make changes if needed, and then approve the credit check or not. He'll approve this one.



3424

Order

Number: 3424
Amount: $15,400.00
Vehicle Brand: fjord
Vehicle Model: FJ-100

Customer

Number: 2
Name: Amy L. Robinson
Balance: $24,669.00
Credit Limit: $20,891.00
Dealer Code: dealer03

Now let's go back into the code again, and look at the **CustomerCreditCheck** class. The Initialize method runs a service on the AppServer to retrieve order information from the OpenEdge database. Remember that the client user interface has to communicate and coordinate both with OpenEdge via the AppServer calls, and with Savvion.

```
run service_orderbynumber.p on hAppServer
                (input  pcBrand,
                 input  iOrderNum,
                 output dataset dsOrder).
```

Then it runs another service to get customer information, and assigns the values that come back to the UI.

```
run service_getcustomer.p on hAppServer
                (input  pcBrand,
                 input  iCustNum,
                 output dataset dsCustomer).
```

When Paul clicks the Approve button, the method that handles that event runs the **UpdateCreditApproval** method, and that first runs another service to update the approval status in the database:

```
run service_updatecreditapproval.p on AppServer (
            input uxVehicleBrand:Text,
            input integer(uxOrderNumber:Text),
            input plApproved).
```

Then, after retrieving Dataslot values for the current task from Savvion…

```
        oDataSlot = this-object:CurrentTask:GetDataSlots().
```

…the method updates the CreditApproval Dataslot value:

```
        do iLoop = 1 to iMax:
            case oDataSlot[iLoop]:Name:
                when  'CreditApproval' then
                do:
                    oDataSlot[iLoop]:Value = plApproved.
                    leave.
                end.
            end case.
        end.
```

Remember that the user sees a list of both tasks assigned to him specifically, and tasks assigned to his group which are available to him. If the current task was one available to the group, which the task **Status** value shows, then the code assigns it to Paul:

```
if this-object:CurrentTask:Status eq 'I_AVAILABLE' then
    this-object:CurrentTask:Assign(this-object:BpmUser).
```

Finally, it tells Savvion to mark the task as complete, just as you would do in the Savvion forms by clicking the Complete button at the bottom of the form:

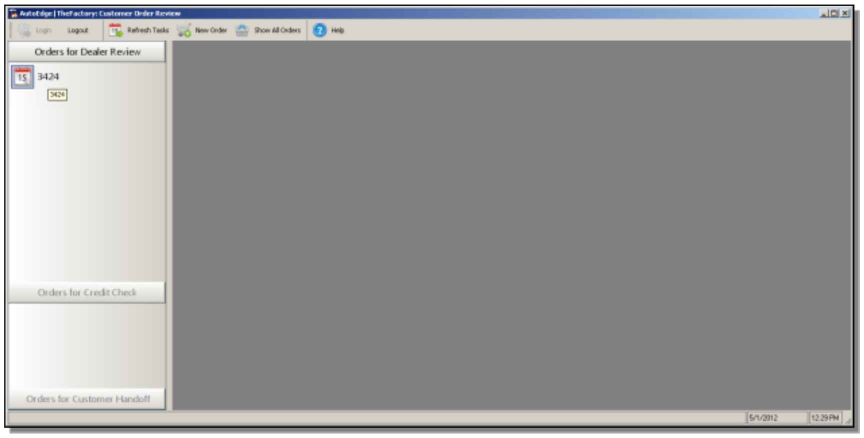
```
        this-object:CurrentTask:Complete().
```

That's what happens in the background when Paul reviews and approves Amy's order. When he approves the order, it disappears from his task list, and he can wait for more orders to come in. Instead, I have Paul log out so that I can show you the rest of the steps in completing the order process.

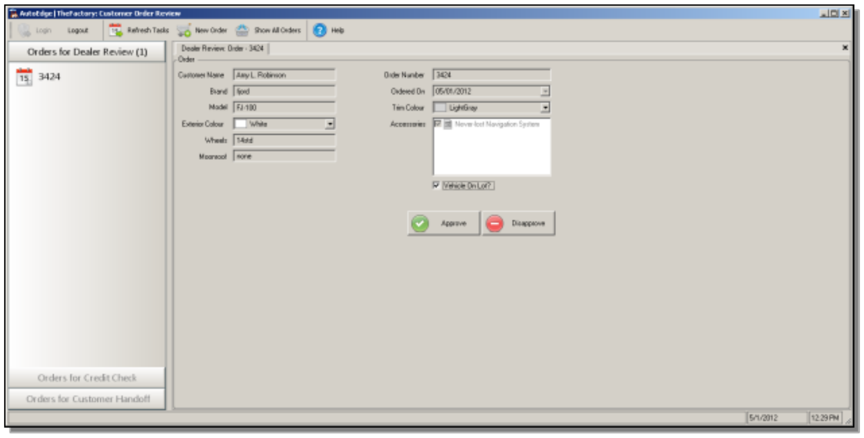
First a quick look at what happens when he logs out. In the DealerReview class, I look at the **DealerLogout** method. (Once again, some of the method names here are dealer specific because that's what the original version of the GUI application handles. Now the method applies to all users.) After running a procedure on the AppServer to register there that the user is logging out, the code runs the **Disconnect** method in the UserSession object. The parameter value of true tells Savvion to log out of the process instance, just as I could do in the Portal.

```
run service_userlogout.p on AppServer (input UserContextId).
        if this-object:UserSession:Connected then
            this-object:UserSession:Disconnect(true).
```

That's as much as I'm going to show you in the code. So that you can see how the rest of the application works, the remainder of the paper walks through the remaining screens and tasks. Just as I have logged in and out of the same BPM Portal instance in other videos to show what different users would see, I use this one instance of the user interface for all the types of users who would be running this application on their desktops and keeping track of their own assigned tasks. As always, the brand is Fjord. I log in as a sales user this time. Here the application knows to gather assigned and available task instances for the Dealer Review task, and you can see that after Paul Cox completed the Credit Check task for Amy's pickup truck, the next task in the process instance number 3424 has been assigned to Sales:



The Sales person sees the order information, checks whether the Vehicle is in stock or not, and approves the order for delivery:

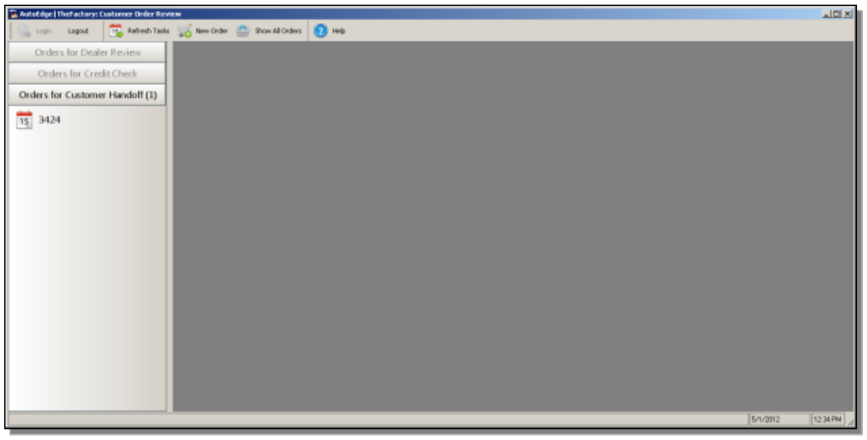

AutoEdge | TheFactory: Customer Order Review
Login
Logout
Refresh Tasks
New Order
Show All Orders
Help
Orders for Dealer Review (1)
3424
Dealer Review: Order - 3424
Approve
Disapprove
Orders for Credit Check
Orders for Customer Handoff
5/1/2012
12:29 PM

Order

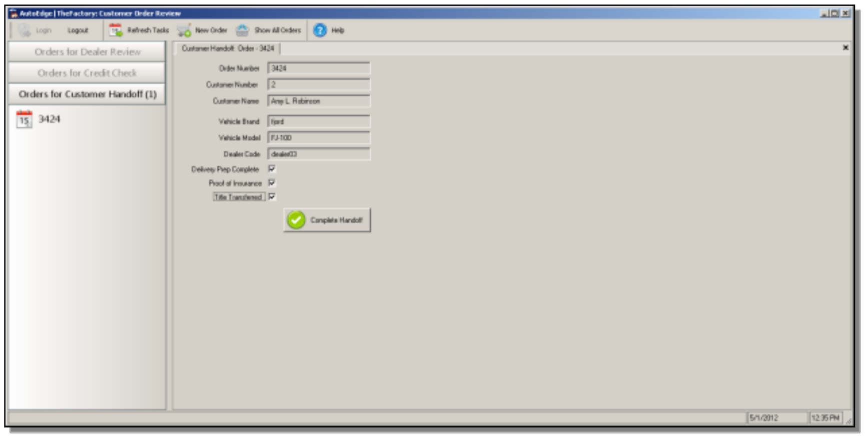
Customer Name	Amy L. Robinson
Brand	fjord
Model	FJ-100
Exterior Colour	White
Wheels	14std
Moonroof	none
Order Number	3424
Ordered On	05/01/2012
Trim Colour	LightGray
Accessories	☑ Never-lost Navigation System

☑ Vehicle On Lot?

Now there’s just one more step in the process. This time Alice does the job of the Support organization, and prepares the truck for delivery:

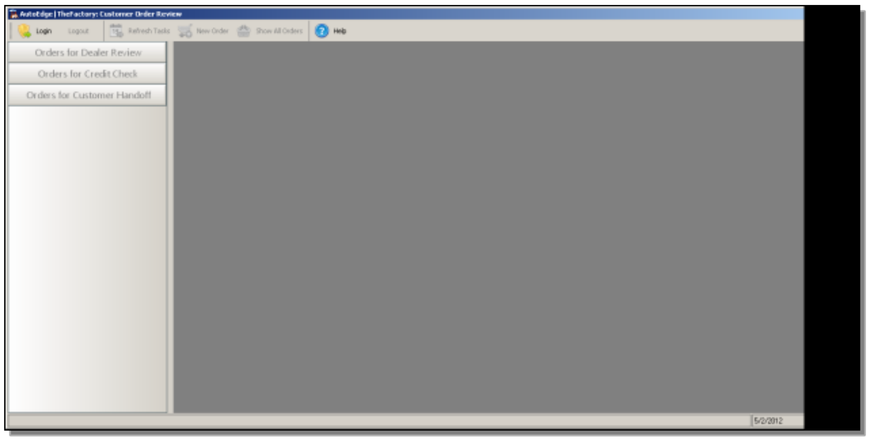


Everything’s complete on prepping the vehicle, so the process instance is completed:



Order Number	3424
Customer Number	2
Customer Name	Amy L. Robinson
Vehicle Brand	fjord
Vehicle Model	FJ-100
Dealer Code	dealer03
Delivery Prep Complete	✓
Proof of Insurance	✓
Title Transferred	✓

Alice can log out or wait for more tasks to come her way.



This is a very nice example of how you can build an ABL user interface of any kind, using the OpenEdge support for GUI for .NET, or WebSpeed, or an RIA user interface, or a native ABL user interface, to manage part or all of a Savvion process from within your ABL application. Once again, consult the documentation for the full details on using the new BPM Application Programming Interface in OpenEdge 11.