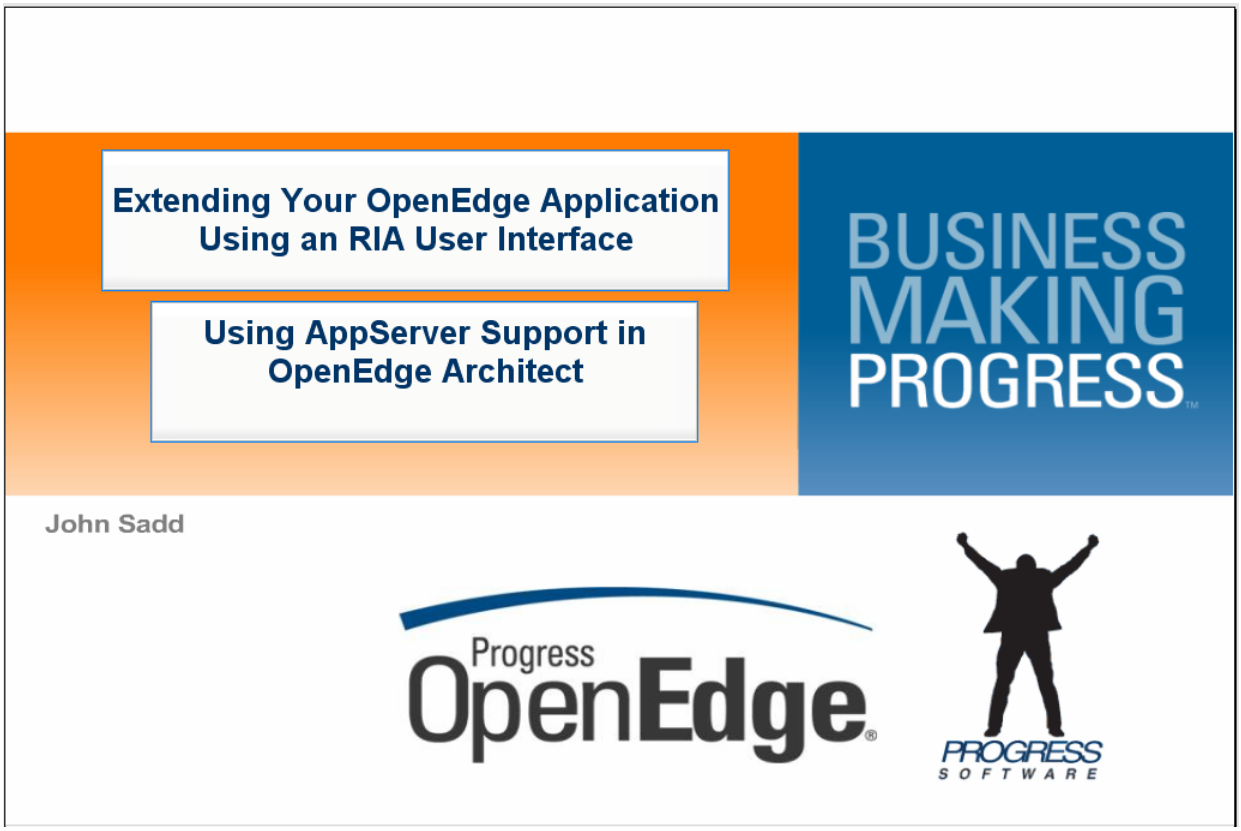
John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
August 2010



DISCLAIMER

This paper accompanies a two-part video that continues the AppServer thread in the series on building the foundation for a Rich Internet Application connection to your server-side OpenEdge business logic. The previous session on this topic – ***Starting OpenEdge and Configuring a Sample AppServer*** -- showed how to get into OpenEdge Explorer and use it to configure and enable an AppServer broker. This paper and its videos show you how to use the support for AppServer in OpenEdge Architect to develop and test distributed applications.

I start with a project named **`WebSamples`** that is used in several presentations in this series. The code below is a very simple test procedure that I want to be able to run on an AppServer:

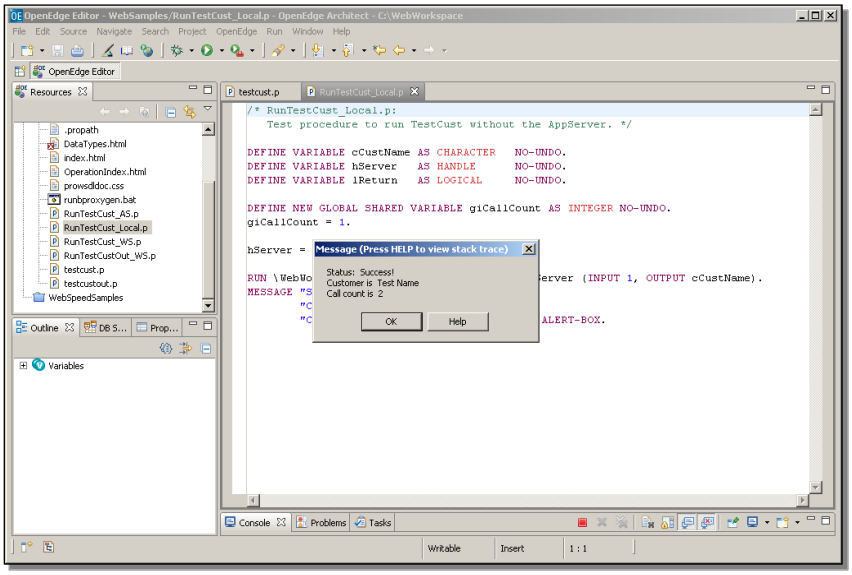
```
/* TestCust.p:
   Test procedure for AppServer and WebServices call. 10/07 */

DEFINE INPUT  PARAMETER pCustNum  AS INTEGER   NO-UNDO.
DEFINE OUTPUT PARAMETER pCustName AS CHARACTER NO-UNDO.

DEFINE NEW GLOBAL SHARED VARIABLE giCallCount AS INTEGER NO-UNDO.

giCallCount = giCallCount + 1.
IF pCustNum = 1 THEN
DO:
    pCustName = "Test Name".
    RETURN "Success!".
END.
ELSE DO:
    pCustName = "No Name".
    RETURN "Failure!".
END.
```

It returns a name as a character string, and a success or failure return value depending on the integer value passed in. Importantly, it also defines a **`GLOBAL SHARED VARIABLE`** that it increments, and you'll see in a moment why this is revealing.

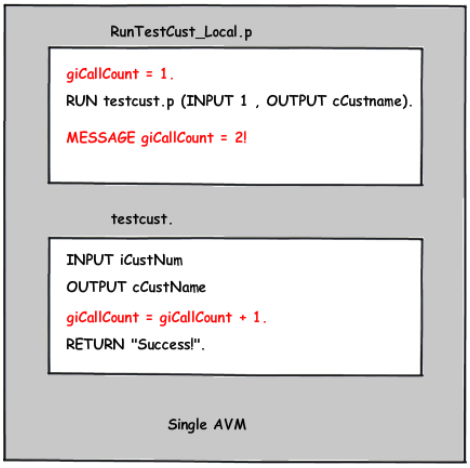
Next is a procedure that runs **`testcust.p`** and displays a message with the values that come back:

```
/* RunTestCust_Local.p:
   Test procedure to run TestCust without the AppServer. */

DEFINE VARIABLE cCustName AS CHARACTER   NO-UNDO.
DEFINE VARIABLE hServer   AS HANDLE      NO-UNDO.
DEFINE VARIABLE lReturn   AS LOGICAL     NO-UNDO.

DEFINE NEW GLOBAL SHARED VARIABLE giCallCount AS INTEGER NO-UNDO.
giCallCount = 1.

hServer = SESSION.

RUN \WebWorkspace\WebSamples\TestCust.p ON hServer (INPUT 1, OUTPUT cCustName).
MESSAGE "Status: " RETURN-VALUE SKIP
        "Customer is " cCustName SKIP
        "Call count is " giCallCount VIEW-AS ALERT-BOX.
```

Setting the server handle to **SESSION** means that the **RUN** statement will execute locally. Notice that it also defines the same global variable and initializes it to 1. I first run this locally in the project's AVM:

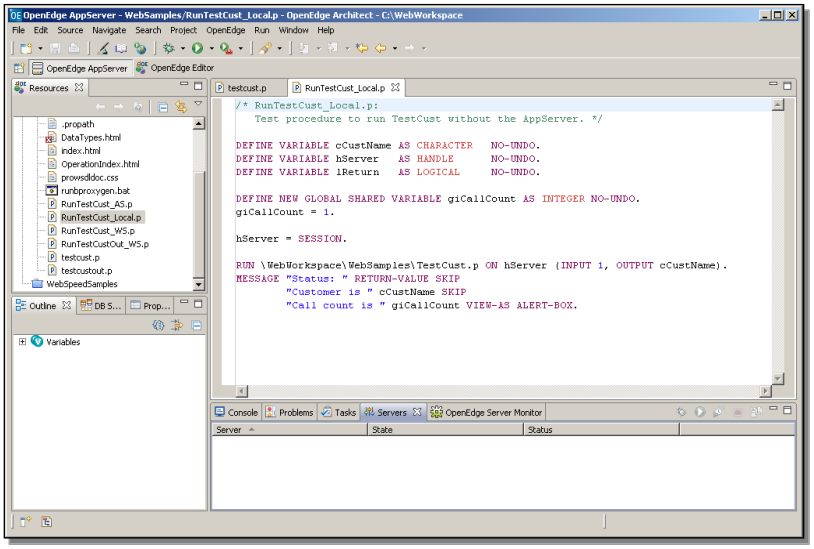


The called procedure **testcust.p** returned the "Success!" return value, got a name back, and the global variable **giCallCount** is equal to 2. The calling procedure set it to 1, and the called procedure incremented it, as illustrated here:

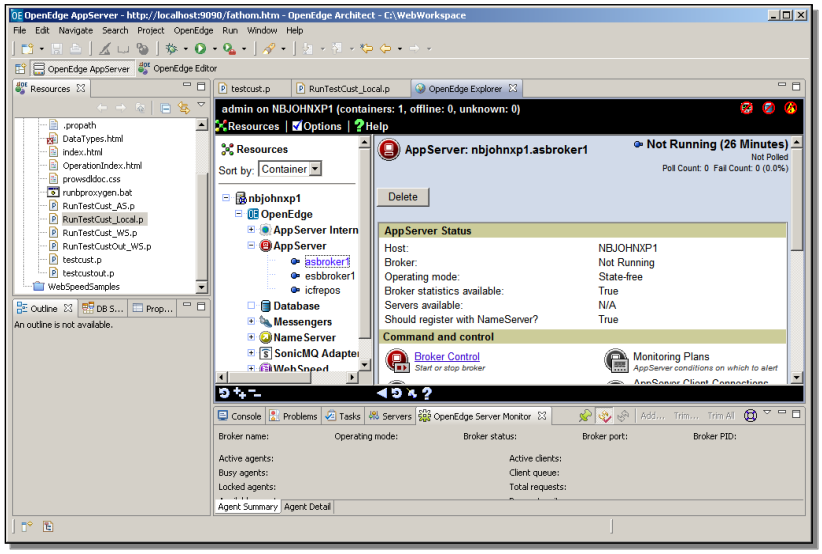


This value just confirms that these two procedures are running in the same OpenEdge session. If the intention is to deploy this as a distributed application, with **`testcust.p`** running on an AppServer with the rest of the business logic, then it's important to integrate the AppServer into the development process and initial testing. The global variable is a very simple example of the kinds of behavior differences you can get when an application runs in multiple OpenEdge sessions. Across an AppServer connection, there will be two separate instances of **`giCallCount`**, so the result returned should not be equal to 2. More realistically, if there are AppServer procedures that pass a temp-table or a ProDataSet **`BY-REFERENCE`**, or if the application runs persistent procedures and expects to be able to access them through their handles, then the behavior can be very different when the application is actually run with the AppServer. This is why we provide support in Architect for using the AppServer as part of the development process.

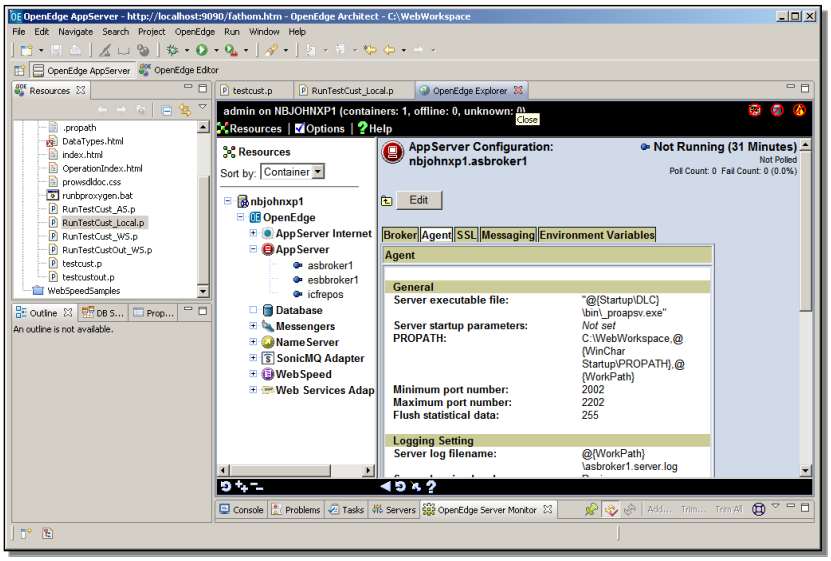
Here is how you enable AppServer support for a project in Architect. Under the **Window** menu, you can open the **AppServer** perspective. What you see when you do this is two new tabs for additional Views, by default down in this pane on the bottom right. The first is the **Servers View**.



This shows you connections that have been defined in Architect to servers such as the OpenEdge AppServer. None are displayed initially, even though there are several AppServers that are defined as part of the Openedge install. That's because you need to define connections to them from Architect. There's also a **Server Monitor View**, which shows details for any AppServers that the session is connected to. Even though it's not displaying anything useful yet, you can access OpenEdge Explorer directly from this View. If you log in to Explorer, you see it come up in the editor pane. And you can drill down into the list of AppServers to see the basic configuration information on, for instance, **`asbroker1`**.



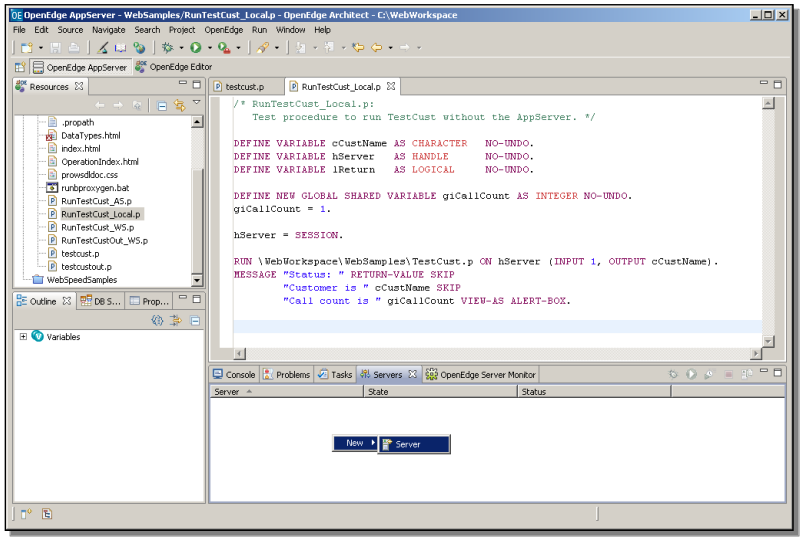
If you go back into **Configuration** information, there are settings for the broker, including its working directory, and information for the agents that run in that broker, including their ProPath.



This shows you that Architect provides you with access to a lot of useful information before you even configure anything for a specific project. But what this paper describes is how to provide AppServer support as part of a project, such as the **`WebSamples`** project, so that you can run procedures such as **`testcust.p`** in an AppServer while doing development. To do that, you need to associate an AppServer with a project, and tell Architect which code in the project is going to be deployed to the AppServer.

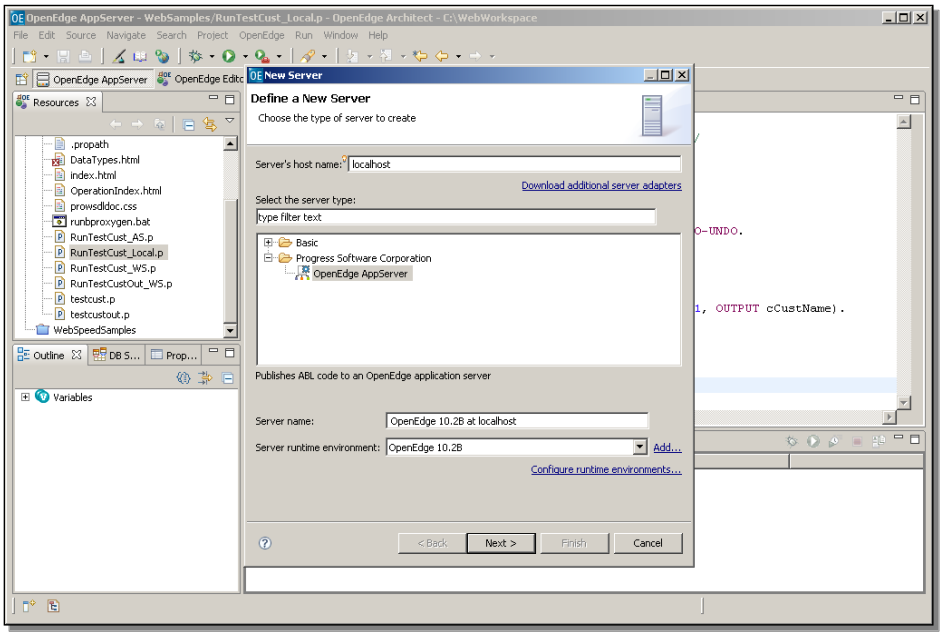
This requirement results in a bit of a chicken-or-the-egg problem. You can't name an AppServer connection as the one to use for a project, because there aren't any server connections until you define them. But if you define a connection, you can't yet name the Architect projects that should use it, because you haven't told architect that they should be AppServer-enabled. So you need to start at one end of the process or the other and then go back and fill in the missing pieces.

Start by defining a new Server in the **Servers View**. Right-click in the View, and select the only option: **New -> Server**.



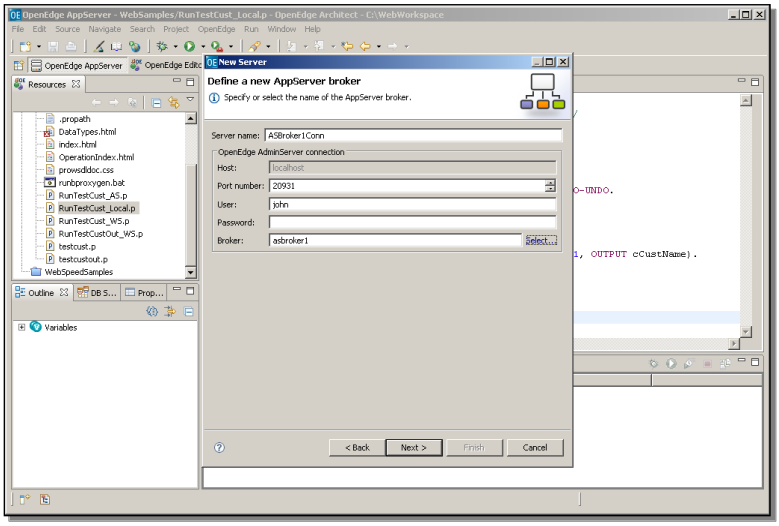
The use of the word *server* here is somewhat confusing. You can't in fact define a new AppServer in Architect, though you can access and even modify a lot of its properties. What's called a server here is really a *connection* from Architect to an existing AppServer. Just keep that in mind.

The **New Server** wizard comes up:

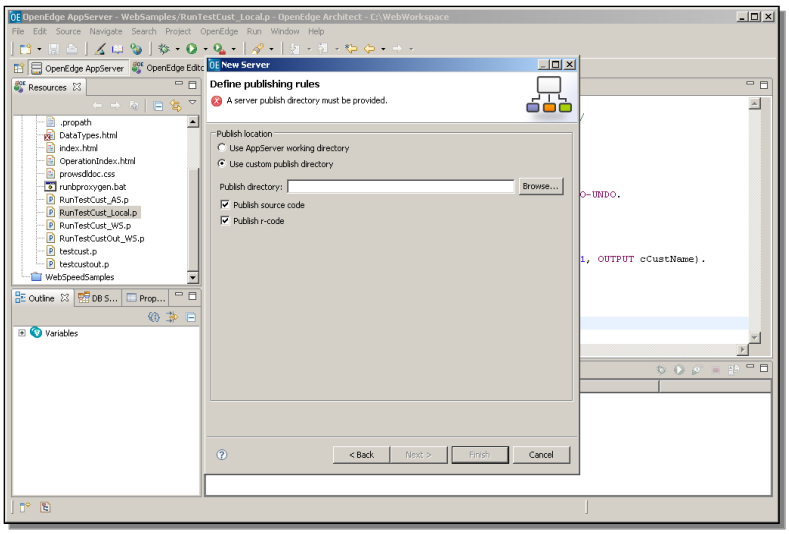


On the first page of the **New Server** wizard, the only host name you can use is **`localhost`**. Architect doesn't yet support publishing code to an AppServer on another machine. Keep in mind that Architect is primarily a development tool, not a deployment management or performance testing tool, and **`localhost`** is all you need to test the critical elements of application logic that depend on operations being run across the AppServer boundary.

The **server type** filter is set up by default to let you define an AppServer – that's the only kind of server connection supported in OpenEdge 10.2B, at least. Provide the server with an appropriate name. Since this is a connection to **`asbroker1`**, this example uses the name **`ASBroker1Conn`**. Continue on to the next page in the wizard.

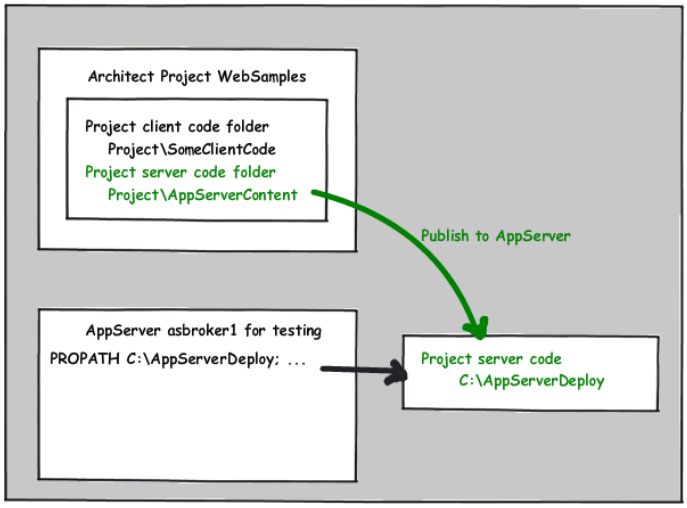


Next press the **Select...** button to locate the AppServer broker. This brings up a dialog to display the brokers on your machine. For the example I select **`asbroker1`**. The **Port number** is the port that the AdminServer runs on, not the AppServer. That should be initialized correctly, so it can just be left at this value. Continue on to the next page of the wizard.



Here you tell Architect where to publish code. This is one of the elements that's incomplete until you define both the server connection on the one hand and the properties for an AppServer project on the other. What this setting means is: for any project that you associate with this AppServer connection, that project is going to be prepared to publish -- to copy -- ABL procedures or classes from a designated set of directories in the project to a target set of directories that the AppServer will see. This is part of the separation of the project's client OpenEdge session and the AppServer session. For my example I could select the AppServer's working directory as the target, which is **`OpenEdge102B\WRK`**. But that's not where I want the code to go. I've created a folder on my machine where I want published code to go, so I enter its name here: **`C:\AppServerDeploy`**. I need to make sure this is in the ProPath of the AppServer, which is a setting I can make a little later.

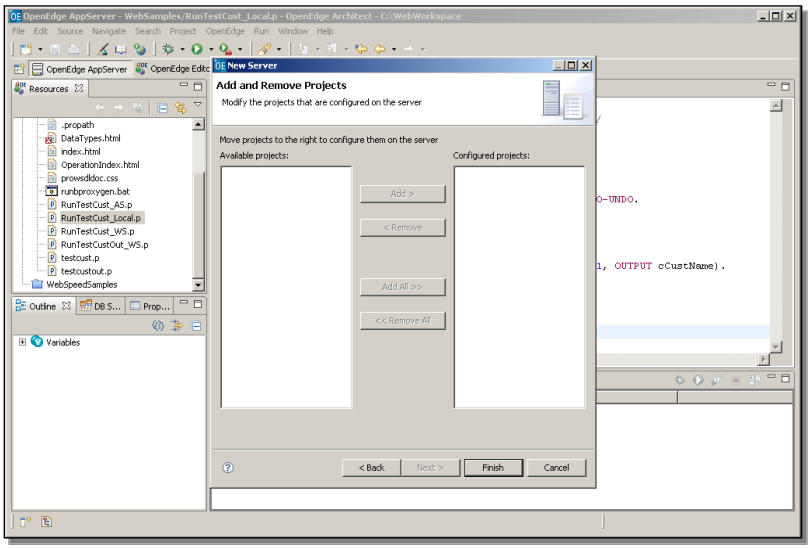
There's another important point to make here which this diagram illustrates:



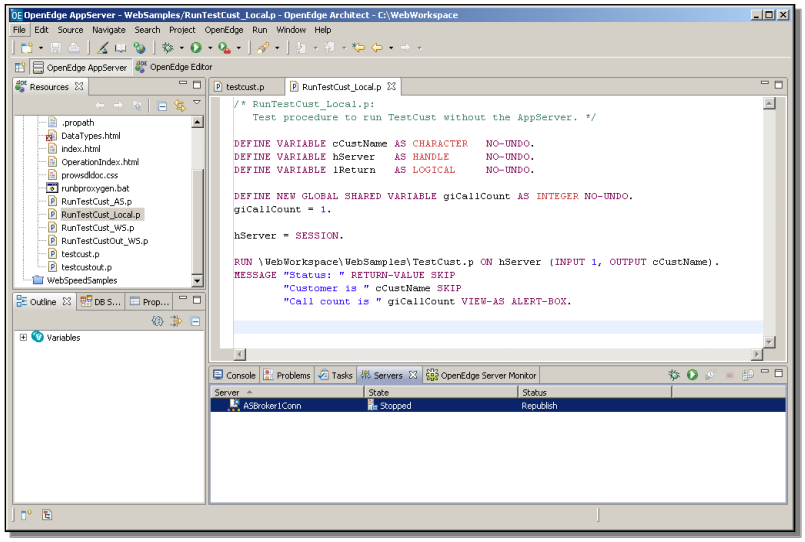
The *publish directory* is where code gets copied to after you create or make changes to it. It's not where you make those changes. So you shouldn't put this folder somewhere in the project or the workspace the project is in. In fact, you'll get an error if you try to do that. It needs to be independent of the project itself.

Finally, in the wizard you can choose whether you want both a copy of the source code and the compiled r-code to be published.

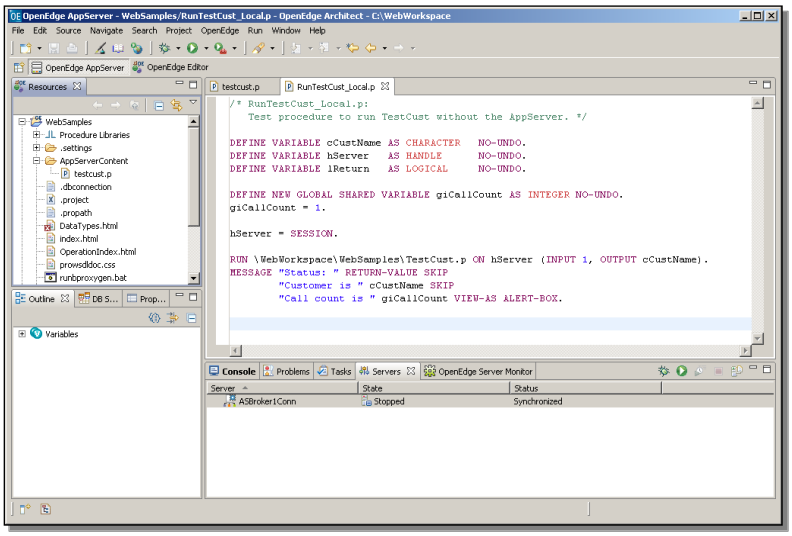
Continue on to the last page of the **New Server** wizard.



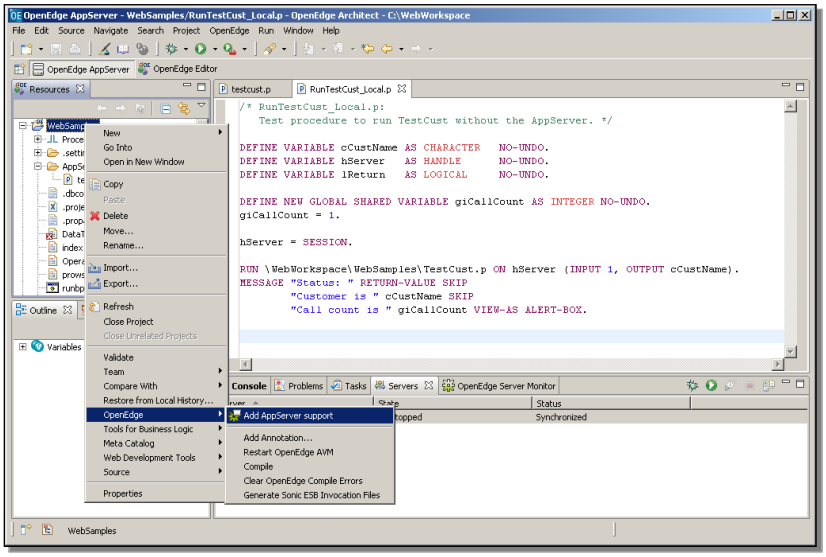
Here's where you see how the process of defining the first connection isn't complete until you go into the project properties as well. There are no available projects to be associated with this server connection, because you have to turn on AppServer support in one or more projects for them to be eligible to show up on this list. So just leave this alone for the moment, and press **Finish** to complete the definition of the new server connection. Below you can see the new server connection in the **Servers** view.



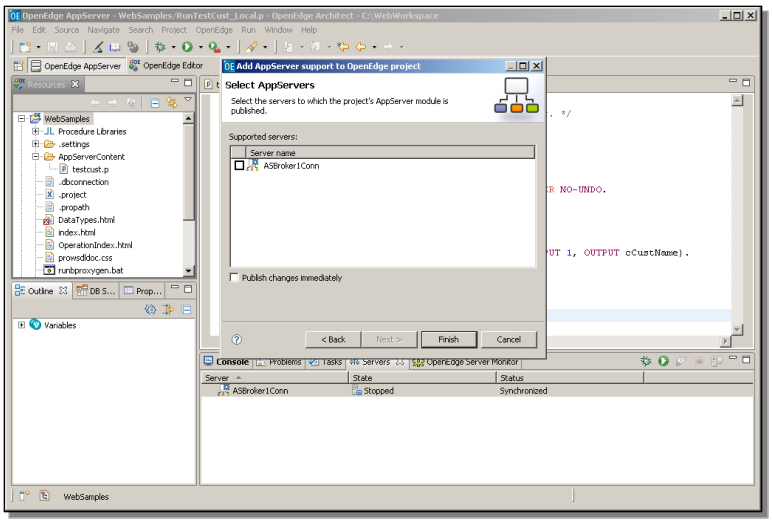
To complete the connection you must make the needed changes to one of your projects. For this example, the first thing I want to do is define a new folder in the project where code to be published to the AppServer should go. This is where I will develop those procedures or classes, so they are part of the project. Right-click on the project, and create a new folder, named for example **`AppServerContent`**. For starters, I want to move my little procedure **`testcust.p`** to that new folder, so that it will get published to the AppServer folder. So here it is in the new **`AppServerContent`** folder.



Once again, this is the source directory for publishing code. The **`AppServerDeploy`** folder that I defined as part of the server connection is the target directory. Next you need to take the step I've been referring to, enabling AppServer support for this project. Right-clicking on the project, select **OpenEdge**, and **Add AppServer support**.

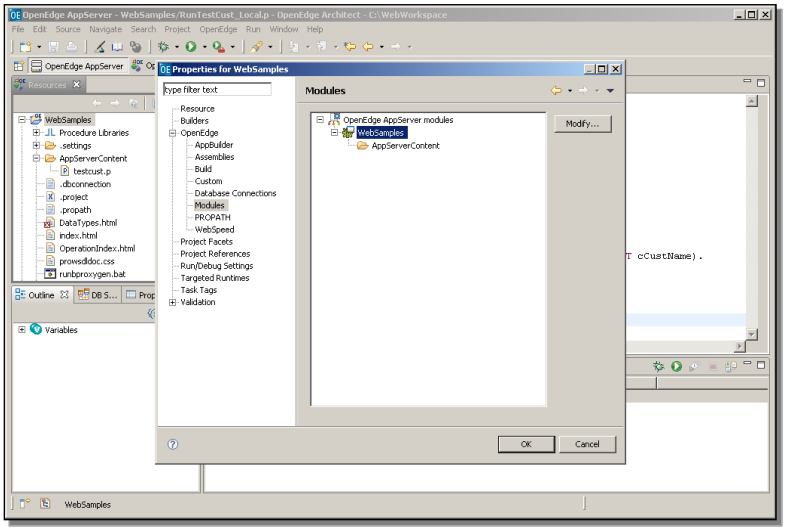


In the example, **`WebSamples`** is the project. Select the parent source folder to publish content from. This is all the code that will be published to the AppServer for testing as it's developed. In the example that's the **`AppServerContent`** folder I created in the project. Now that you've defined an AppServer connection for Architect, you can select it as the one to use for your project.

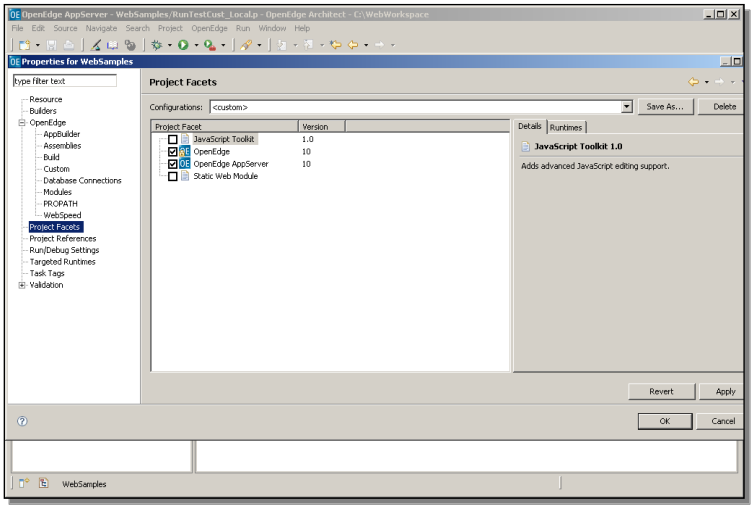


You can ask it to publish any content that is already there, and you're done. That's all there is to it. Now this project is AppServer aware. Architect will automatically publish procedures, classes, and their r-code to a separate directory in the AppServer's propath, so that you can test client-side ABL running those procedures on the AppServer.

Now I will explain how to make changes to these new properties. In the project properties, expand the **OpenEdge** node, and then expand the node labelled **Modules**. A module is a set of related files organized under a directory, in this case all the files that you want to publish to the AppServer. For example, if I select the WebSamples module, I can then modify the pointer to the top-level folder that holds all the AppServer code.

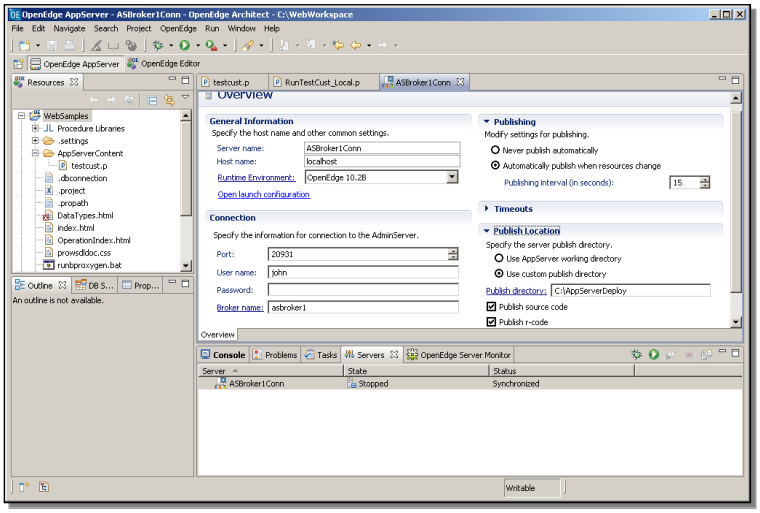


You can also select **Project Facets**. A facet is in effect a marker that is used to identify a project as being of a certain type.

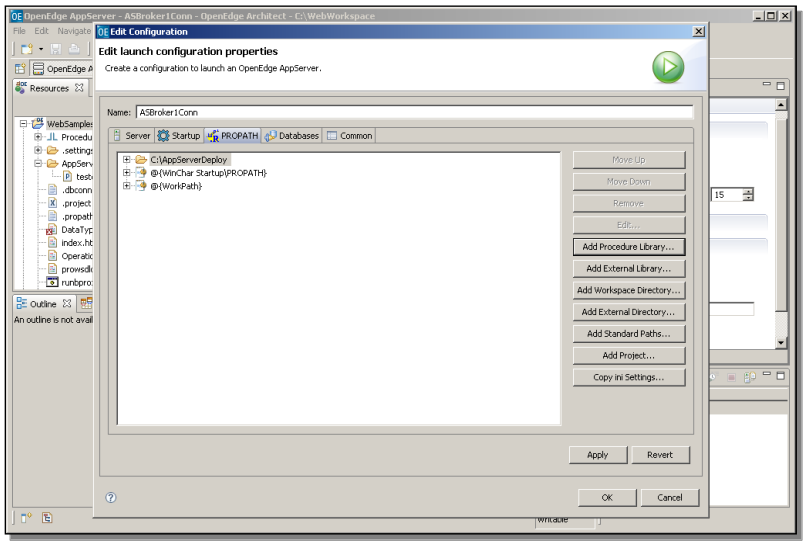


Here you can see **OpenEdge AppServer** as one of several facets that define this project. Adding AppServer support to the project checked this facet on. You could remove AppServer support from the project by checking the facet off. Any OpenEdge project has the **OpenEdge** facet set, and you can't turn it off.

Now we need to complete the testing process and show some more ways to manage AppServers from within Architect. First, if you just double-click on the New Server connection, that brings up an **OverView** display with a lot of information. It confirms the information supplied when you defined the connection. If you expand the **Publishing** item, you can see and modify the options for when code changes are published, and under **Publish Location**, you can likewise confirm or modify where files get published to in order for the AppServer to find them.

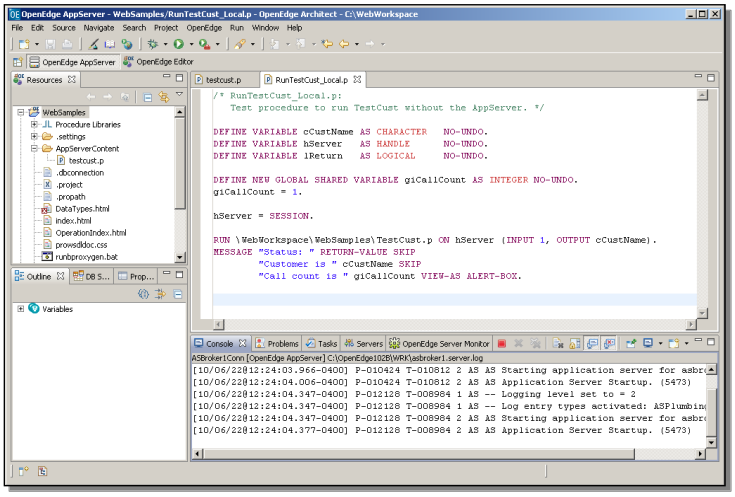


Importantly, over on the left is a link labeled **Open launch configuration**. Remember that in Architect, a Launch Configuration or Run Configuration holds all the options for how code is run: what the top-level procedure is to invoke, the ProPath, database connections, and so forth. So here as well, you can customize the details of how the AppServer is accessed from a project that's AppServer enabled. You can edit the ProPath the AppServer agents use. The tab shown below for instance shows ProPath for **`ASBroker1Conn`**. I can change that to place the **`AppServerDeploy`** folder that I have now specified as the publish location at the head of the ProPath.

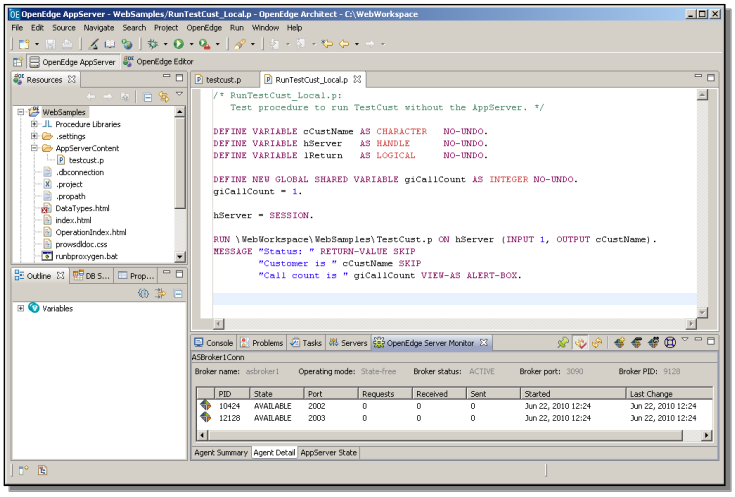


This is one example of the AppServer properties that you can now set directly from Architect as well as from within Explorer. You can customize the database connections the AppServer uses, as well as many other options. That's all from the **OverView** you access by double-clicking on the connection in the **Servers View**.

There are some more things you can do from the **Servers View**. When you select a server connection, there are a number of buttons in this View's toolbar and in the **Server Monitor** to control the AppServer. You can start the AppServer from here, and the **Console View** will pop to the top to display the details of the startup process. One of the other buttons lets you override that behavior of the **Console View** coming up, depending on your preference.

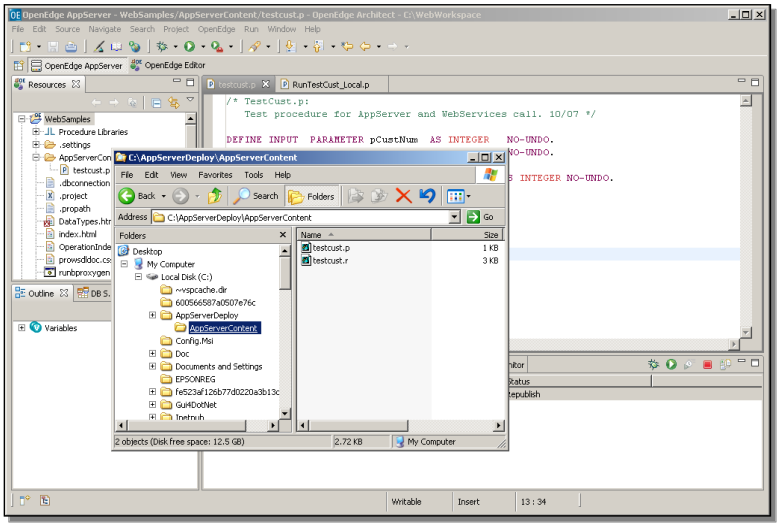


In the **Server Monitor View**, now that there's a connection to select, all the broker details are displayed. If you select the **Agent Detail** tab, you see the state of each of the AppServer agents.

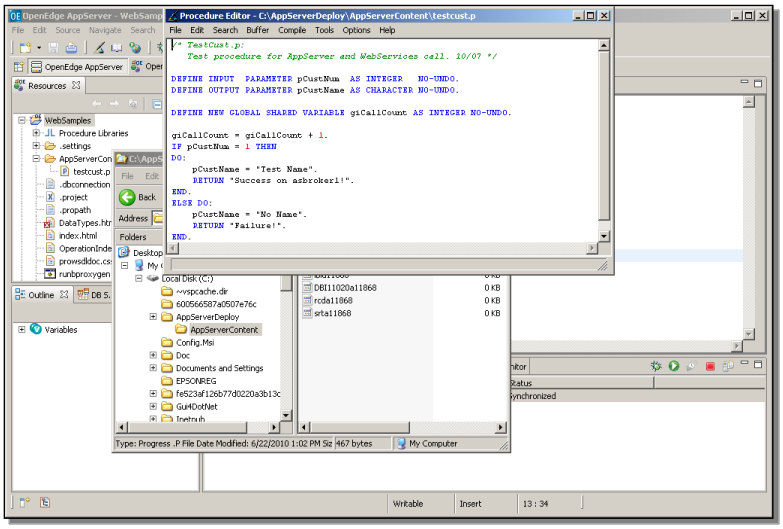


The **Add** button in the **Server Monitor View** toolbar would let me add agents, and the **Trim** button would let me trim the number of agents running in the broker. So you can carry out almost all of the essential operations on an AppServer directly from here in Architect, in addition to being able to fire up OpenEdge Explorer from Architect as well.

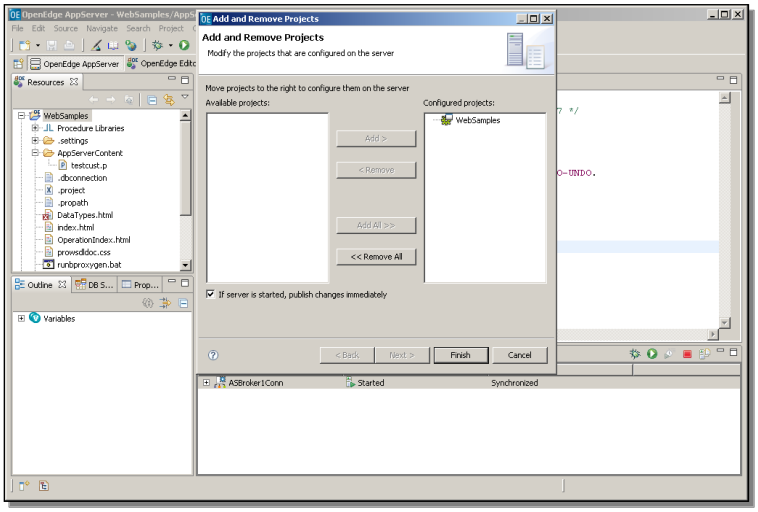
To confirm the behavior of the publish operation and the source and target folders I defined earlier, I can make a simple change in my example to **`testcust.p`** and save that change, which now in addition to compiling the file will automatically publish it to the **`AppServerDeploy`** folder. I can confirm that this worked by taking a look at the file in the target folder. Below you can see the **`AppServerDeploy`** folder. Remember that I set this as the publish location for the server connection. Any AppServer-enabled project that uses this server connection will publish changes from its **`AppServerContent`** folders to this destination. The **`WebSamples`** project has an **`AppServerContent`** folder that's defined as the source for publishing, so a copy of that has been created here as a subdirectory under **`AppServerDeploy.`** Here you can see the source file and r-code for **`testcust.p`**:



If I open the source file in this publish directory I can see the change that I made to change the first **RETURN** statement to return the string "Success on asbroker1!". This confirms that the files were copied over as soon as the change was saved and compiled.



Remember that when I first defined this server connection, the last screen in the wizard came up to let me associate one or more AppServer projects with the connection. And there weren't any, because I had to add AppServer support to a project before it would show up. Here you can see now that after selecting **Add and Remove Projects** for the server, the one project that I've added AppServer support to shows up in the list.

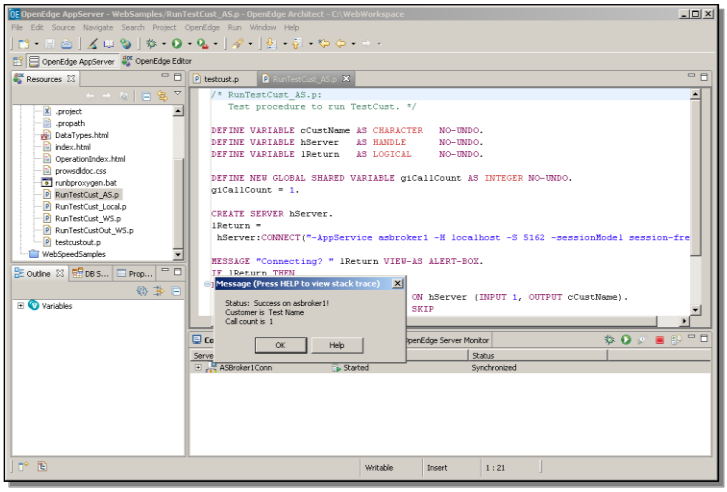


It's on the right-hand side, of course, under **Configured projects**, because I made the association to this server connection when I added AppServer support to the project. But I came back in here just to reinforce the notion that once you have defined one or more server connections, and AppServer-enabled one or more projects, then you will be able to make associations between them in the server properties or the project properties.

Now that I've made all the right connections and started the AppServer, it's time to test whether what I've done in my example works the way I expect. I open a version of my runner procedure, which represents the client side of the application. I've added a statement to connect to **asbroker1**, and a message to confirm that the initial AppServer connect succeeds.

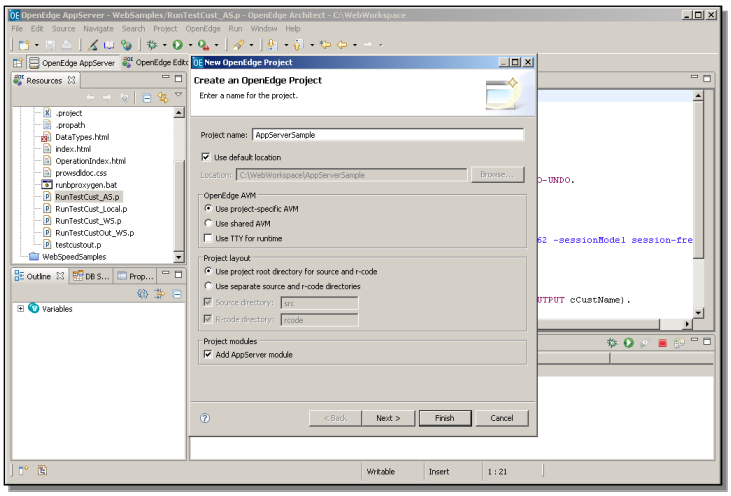
```
/* RunTestCust_AS.p:
   Test procedure to run TestCust. */

DEFINE VARIABLE cCustName AS CHARACTER   NO-UNDO.
DEFINE VARIABLE hServer   AS HANDLE      NO-UNDO.
DEFINE VARIABLE lReturn   AS LOGICAL     NO-UNDO.

DEFINE NEW GLOBAL SHARED VARIABLE giCallCount AS INTEGER NO-UNDO.
giCallCount = 1.

CREATE SERVER hServer.
lReturn =
 hServer:CONNECT("-AppService asbroker1 -H localhost -S 5162 -sessionModel session-
free").

MESSAGE "Connecting? " lReturn VIEW-AS ALERT-BOX.
IF lReturn THEN
DO:
    RUN AppServerContent\testcust.p ON hServer (INPUT 1, OUTPUT cCustName).
    MESSAGE "Status: " RETURN-VALUE SKIP
            "Customer is " cCustName SKIP
            "Call count is " giCallCount VIEW-AS ALERT-BOX.
    hServer:DISCONNECT().
END.
DELETE OBJECT hServer.
```

When I run this procedure, the Connecting message is displayed, and then the message that shows the values that came back.

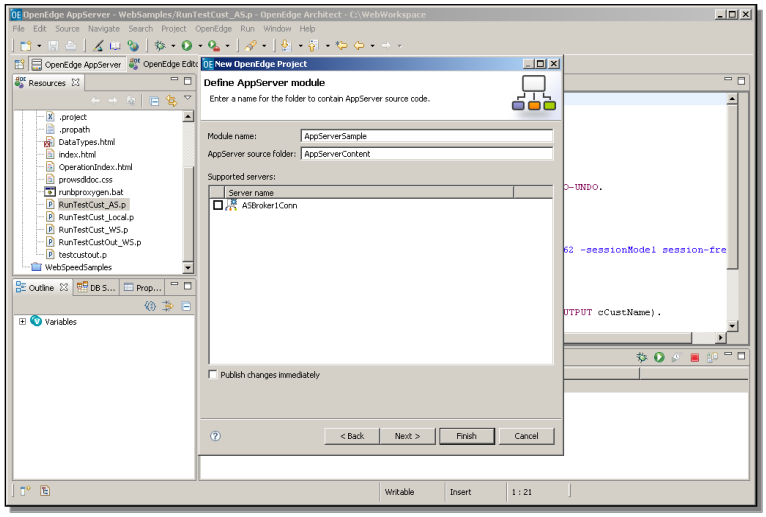


Note that the Call Count, that global variable I defined, is now equal to one. That confirms that the calling procedure and the called procedure are in fact running in different sessions. Once again, this is just a very simple example of why it's important to set up your Architect projects so that you can really test with an AppServer, because your application behavior may be different in ways much more significant than this.

Finally, let me quickly make sure you know how to provide AppServer support for a new project when you first create it. If you define a new OpenEdge project, then right on the main wizard page, there's a checkbox to **Add AppServer module**.

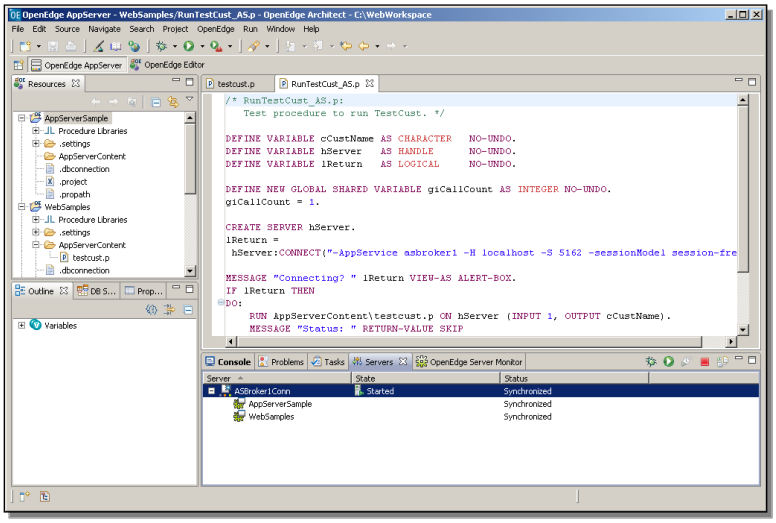


If you check that on, then you'll be prompted to identify the server connection names to associate it with:



You can pick more than one if there's more than one server connection defined, and your project will publish to all of them. Architect will by default add an **`AppServerContent`** directory to the head of your project's ProPath. Then as for any new project you can select database connections, and other project properties. A new session starts up for your new project, if you're not using the shared AVM, and you get a message prompting you to switch to showing the project initially in the Editor perspective.

Back in the AppServer perspective, you can expand the server and see that you now have two projects associated with this server. When you create and modify code in either of those projects, they will publish the code to be runnable on this AppServer.



That completes the two-part session on enabling AppServer support for projects in OpenEdge Architect. In other sessions I look at how you can use the AppServer as the channel for making business logic requests from a rich desktop UI such as the GUI for .NET interface you can now build right in OpenEdge.