Chapter 2

Progress Modernization Framework for OpenEdge

The PMFO Wizard

Purpose

There are many moving parts within a modernized application: business logic, a REST adapter, an AppServer, a web server, authentication logic, UI frameworks, front-end libraries, application code, and HTML layouts. Starting from scratch to get all of these work in concert right out of the box with little to no knowledge about their inner workings is difficult enough, if not fully time-consuming initially. Therefore, a "wizard" approach has been taken to aid developers in generating all the necessary pieces of a working application, while illustrating the connections and interactions between the front-end and back-end.

In this material, the stage will be set for two modes of operation, either starting with the project as-is from the Spark repository or building a new project and incorporating the wizard components on your own. Ideally this should give you enough information to get a known-good configuration up and running, as well as illustrating the underpinnings for starting a brand-new modernization project.

NOTE

At this time, the PMFO framework is available from the public Progress GitHub repository. There are 3 distinct repositories with code related to the framework, though for our purposes we are only interested in the demo applications. Therefore, you will only need to perform a "**Git Clone**" of the following repository location:

https://github.com/progress/Spark-Demo

NOTE

As part of this training material, we will focus on use of a PASOE instance with PDSOE. These instances are copies of a Tomcat installation from your Progress installation directory at **%DLC%/servers/pasoe** and may be referred to as **CATALINA_HOME**. For reference, the home directory for each of your instances will be referred to as **CATALINA_BASE**.

Glossary of Terms

Throughout this material several terms will be used, many of which may be new to ABL developers. In order to provide context please consult the definitions below as needed.

Annotations: Special comments within a Business Entity class that are understood by PDSOE. These are used to generate the metadata to expose the class methods to the REST adapter, and to produce the necessary data for the catalog definition.

Authentication: The act of establishing an identity via use of credentials (aka. AuthN).

Authentication Model: The method by which a user will attempt to authenticate with a remote REST service. The current and typical options are Anonymous (no security), Form (utilizing a POST of username/password credentials), or Basic (sending of an authentication token in an HTTP header). Note: This is different from the security model, which dictates how the credentials will be validated.

Authorization: The act of establishing permission to an identity through use of Roles (aka. AuthZ).

Business Entity: An ABL class, which contains methods to be exposed to a REST adapter instance.

Catalog: A document in JSON format that describes a REST service, including all of its resources, operations, and parameters. Used in conjunction with the Progress JSDO features.

Client-Principal Object (CPO): A sealed (read-only) token that describes the identity of a user. Produced via the authentication process and sent along to the AppServer from Tomcat on the session request handle.

JSDO: Shorthand for the Progress JavaScript Data Object, a client-side library for accessing remote services generated by PDSOE and provided via the OE REST Adapter.

Management Console: Shorthand for the Progress Management Console, which is a web-based replacement for the OpenEdge Explorer tool. Accessible at http://localhost:9090 (u/p: admin/oeadmin)

PASOE: Shorthand for the Progress Application Server for OpenEdge, the next-gen AppServer.

PDSOE: Shorthand for the Progress Developer Studio for OpenEdge, the Eclipse-based IDE.

Roles: Arbitrary names that represent actions or behavior that a user may perform. Typically used in conjunction with authorization processes to determine what URI's a user may access within a REST service. Users are assigned to 1 or more roles, and URI's are granted access by 1 or more roles.

Security Model: The type of security scheme applied to a REST adapter instance via Tomcat. Typical options include Anonymous (no authentication), Local (username/password via flat file), and OERealm (username/password via remote AppServer). Note: In the case of OERealm the AppServer providing the authentication for the user may utilize a database in order to read from an existing user/profile table.

Service: A particular transport service by which data is accessed. Each service may contain 1 or more resources (URI's), which may respond to 1 or more operations (HTTP Verbs), which may contain 1 or more input/output parameters (including, but not limited to, some type of schema object).

Spring Security: An industry-standard Java EE security framework implemented within Tomcat.

Transport: How you intend to access your WebApp (e.g. by REST, Web, SOAP, or APSV).

WebApp: Any one of (potentially) many distinct "web applications" deployed to Tomcat.

WebHandler: A new construct available in PAS that supports WebSpeed functionality. It can also provide generic access to any HTTP request, providing programmatic control over all aspects of your request and response as ABL objects. Translation: it's a class that can provide REST support.

Creating a New Project

To show usage of the wizard, we first need a new project. The steps for creating our project are very similar to the "**Project Setup**" instructions as outlined in **Appendix A**, but focuses on the process of creating a project from quickly via scripting and incorporating the specific components necessary from the PMFO framework. **For all work going forward in this material we will utilize this new project.**

Before we begin, it should be noted that in OpenEdge terms, a "Data Object" provides the necessary components suited to exposing your ABL business logic in a manner that is expected by the JSDO on the client side. And in order to make use of this type of code in a PASOE environment you must utilize an "ABL Web App" project type which will expose these objects via an "ABL Service".

NOTE

Before proceeding, please see the sections **"Obtaining Source Code"** and **"Preparing the PDSOE Workspace"** in **Appendix A**. These steps will ensure a consistent working directory for your projects and ready your development space in the Progress Developer Studio for OpenEdge. If working on a provided virtual machine, these steps should have been performed for you already.

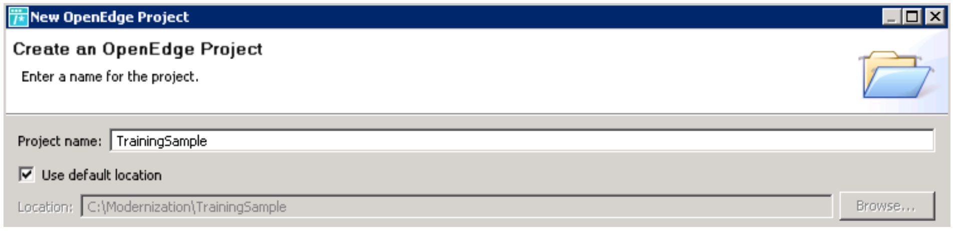
NOTE

As of OpenEdge 11.7 the wizard process for creating a new project changed significantly and selecting the necessary options at the very start will alter the dialogs shown later in the process. Therefore, please take special care to follow the steps exactly, to ensure that you are prompted with the expected dialogs. Screenshots will be provided to allow you to check your work.

The ABL Web App Project

Let us create a new project to implement a Data Object Service via PMFO.

1. Start the Progress Developer Studio by selecting **Start → Developer Studio**
2. Within PDSOE, to go **File → New → OpenEdge Project**
3. For the project name, use "**TrainingSample**".
   a. You may leave the "*Use default location*" option checked, provided the workspace is still **C:/Modernization** (See "**Preparing a PDSOE Workspace**" in **Appendix A**.)



New OpenEdge Project

Create an OpenEdge Project

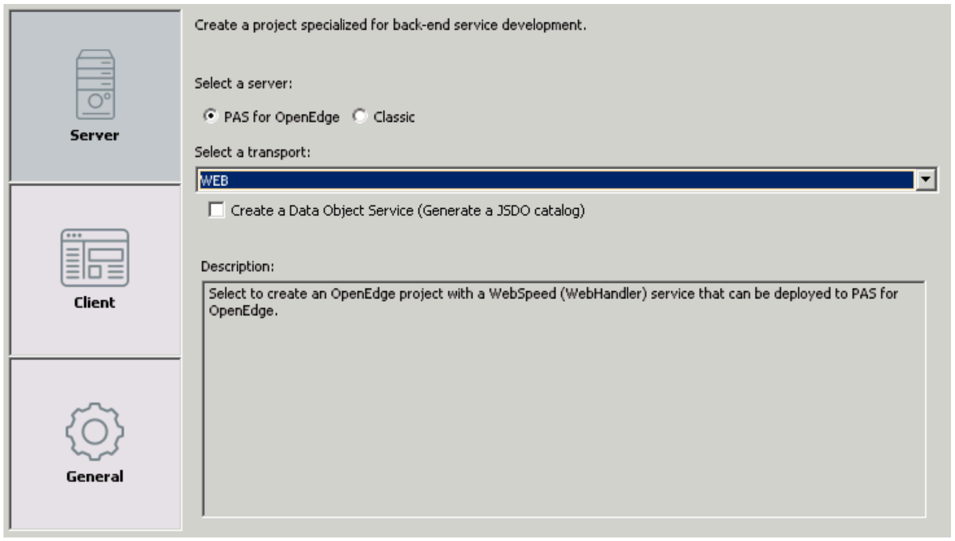
Enter a name for the project.

Project name: TrainingSample

[✓] Use default location

Location: C:\Modernization\TrainingSample

4. Select the "*Server*" option on the left, as we want to create web-based services.
5. Select "*PAS for OpenEdge*" as we want to use the next-generation AppServer.
6. Select "*WEB*" as the Transport as we will want to utilize a WebHandler class.
7. Leave the option "*Create a Data Object Service*" unchecked, as we will be using PMFO to provide our Data Object Service and to generate any catalog data.



Create a project specialized for back-end service development.

Select a server:

[● PAS for OpenEdge] ○ Classic

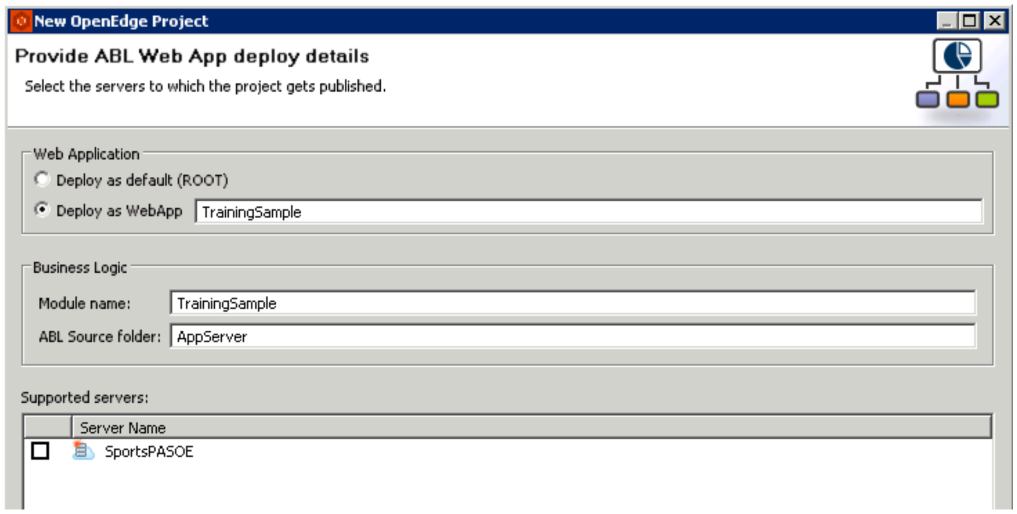
Select a transport: WEB

☐ Create a Data Object Service (Generate a JSDO catalog)

Description:

Select to create an OpenEdge project with a WebSpeed (WebHandler) service that can be deployed to PAS for OpenEdge.

8. Click **Next**, and note that we can deploy to either the default project (ROOT) or to a specific WebApp. **It is highly encouraged that unless you fully intend to host only one WebApp within your PAS instance that you should deploy as a named WebApp**.
   a. Since we will host two webapps within the same PAS instance, click on the radio button to "*Deploy as WebApp*" and accept the given project name.
   b. We need to **uncheck** the option for the *SportsPASOE* server, as we will prepare a new PAS instance in just a moment.



New OpenEdge Project

Provide ABL Web App deploy details

Select the servers to which the project gets published.

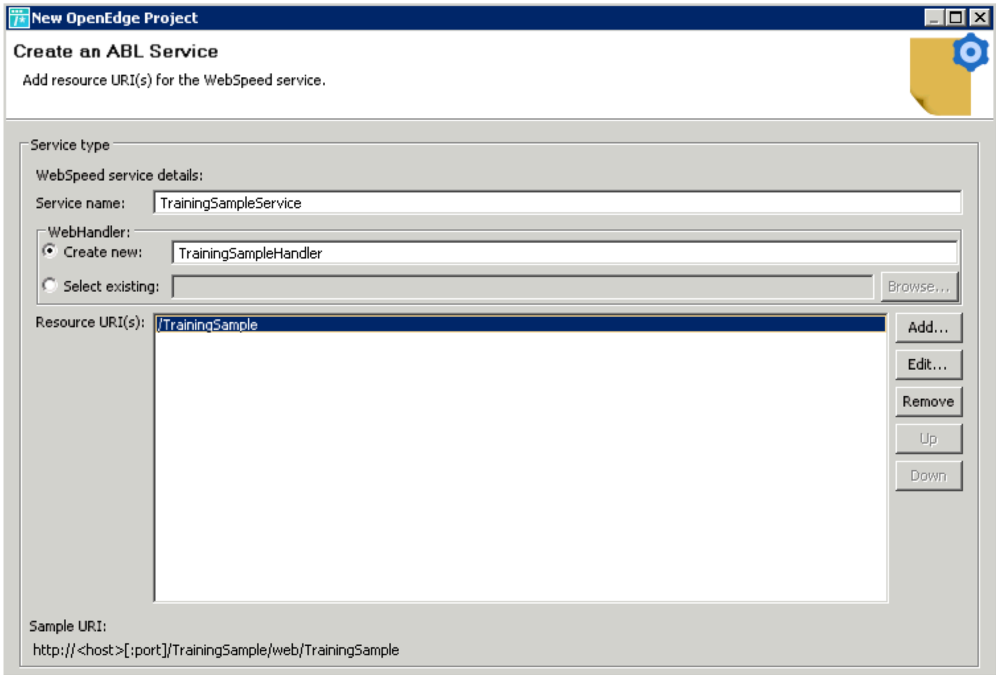
Web Application
- Deploy as default (ROOT)
- [✓ Deploy as WebApp] TrainingSample

Business Logic
- Module name: TrainingSample
- ABL Source folder: AppServer

Supported servers:

	Server Name
☐	SportsPASOE

9. Click **Next** to proceed to the ABL Services dialog, which will already be prepared to utilize a WebHandler for our exposed service.
   a. Just leave all defaults for the "**TrainingSampleService**" as provided and click **Next**.



New OpenEdge Project

Create an ABL Service

Add resource URI(s) for the WebSpeed service.

Service type

WebSpeed service details:

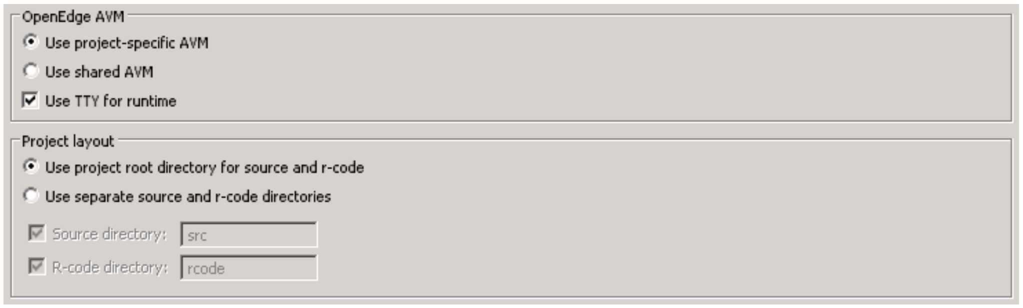
Service name: TrainingSampleService

WebHandler:
- [✓ Create new:] TrainingSampleHandler
- Select existing:

Resource URI(s): /TrainingSample

Sample URI:
http://<host>[:port]/TrainingSample/web/TrainingSample

10. Click **Next** and accept the defaults for the AVM, and accept defaults for source and r-code.

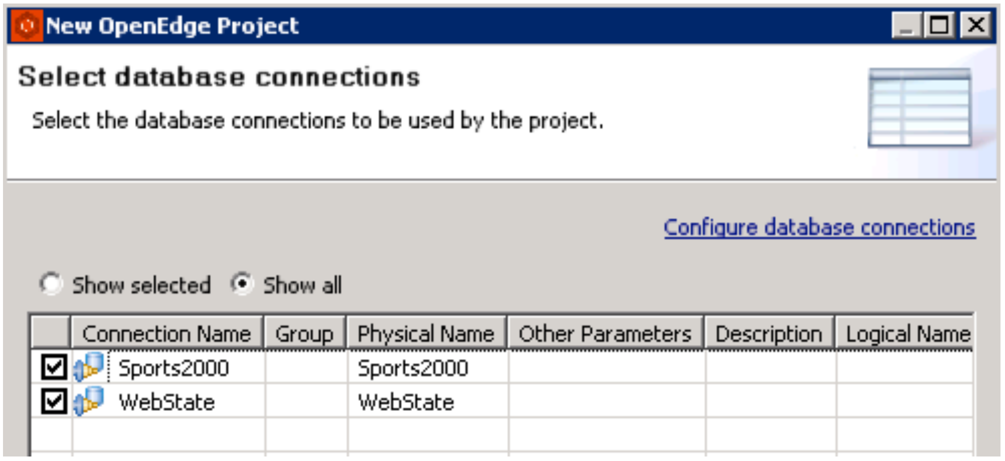


OpenEdge AVM
- [✓ Use project-specific AVM]
- Use shared AVM
- ☑ Use TTY for runtime

Project layout
- [✓ Use project root directory for source and r-code]
- Use separate source and r-code directories
- ☑ Source directory: src
- ☑ R-code directory: rcode

11. Click **Next**, and accept the defaults for the PROPATH by clicking **Next** again.

12. Select both the **Sports2000** and **WebState** database connections, then click **Finish**.
    a. If prompted to open any additional perspectives, just check the box to remember this decision and accept the notification. The project should now be created for you.


New OpenEdge Project

Select database connections

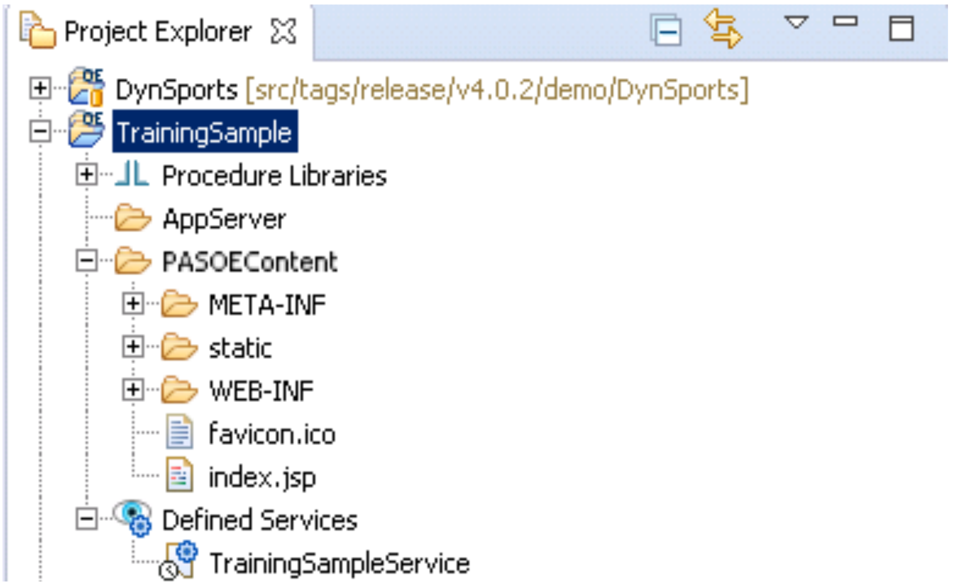
Select the database connections to be used by the project.

Configure database connections

Show selected [✓ Show all]

	Connection Name	Group	Physical Name	Other Parameters	Description	Logical Name
✓	Sports2000		Sports2000			
✓	WebState		WebState			

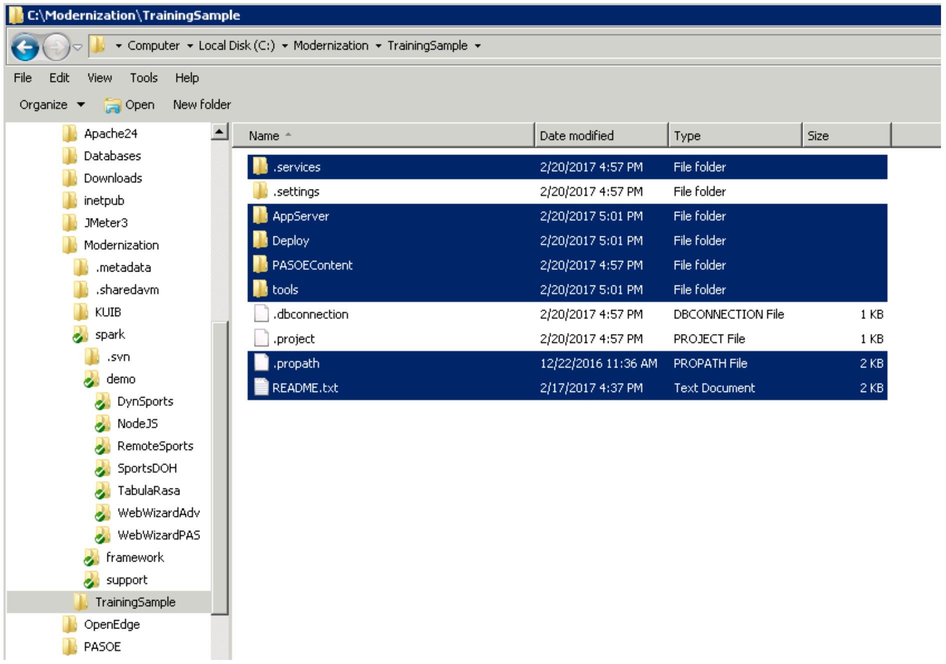
13. If created successfully, you should now see your project in the explorer view. The default folders created include **AppServer** for your common ABL code, **PASOEContent** for the Tomcat WebApp itself, a **TrainingSampleService** under *Defined Services*, and your new WebHandler class opened for editing.
    a. The "*TrainingSampleHandler.cls*" class file will not be needed, so you may close the editor. We just needed to specify a default service for purposes of completing the wizard. We will overlay a fully-customized WebHandler-based service from PMFO as our next steps.


```
Project Explorer
+ DynSports [src/tags/release/v4.0.2/demo/DynSports]
- TrainingSample
    + Procedure Libraries
      AppServer
    - PASOEContent
        + META-INF
        + static
        + WEB-INF
          favicon.ico
          index.jsp
    - Defined Services
          TrainingSampleService
```

Project Tailoring

With the project now created, we can make some additional changes to tailor it to our needs.

1. First, **close the PDSOE application**, as we will overlay some special project options that can only be incorporated correctly upon restart of the Developer Studio.
2. In the Windows Explorer, navigate to the **C:/Modernization/Spark-Demo/oe117** folder.
3. From within the folder called **TabulaRasa** (Latin for "Blank Slate") select all contents (Ctrl+A) in this folder, and **Copy** everything (Ctrl+C).
4. Navigate to the new **C:/Modernization/TrainingSample** folder and **Paste** the contents (Ctrl+V) within this folder. When prompted to overwrite any files/folders, answer **Yes** to all.
    a. When complete, the following files and folders should be highlighted.

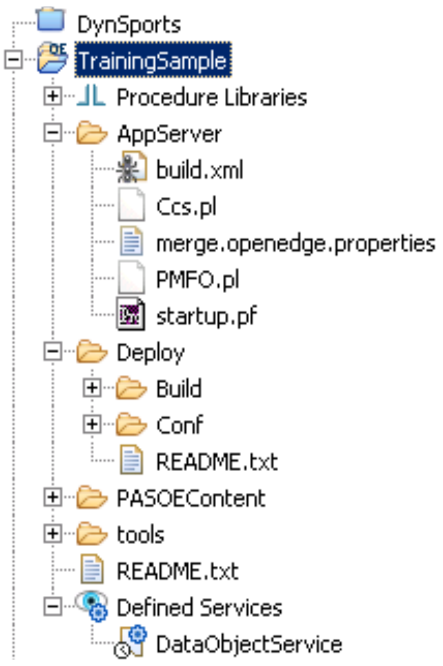


Name	Date modified	Type	Size
.services	2/20/2017 4:57 PM	File folder	
.settings	2/20/2017 4:57 PM	File folder	
AppServer	2/20/2017 5:01 PM	File folder	
Deploy	2/20/2017 5:01 PM	File folder	
PASOEContent	2/20/2017 4:57 PM	File folder	
tools	2/20/2017 5:01 PM	File folder	
.dbconnection	2/20/2017 4:57 PM	DBCONNECTION File	1 KB
.project	2/20/2017 4:57 PM	PROJECT File	1 KB
.propath	12/22/2016 11:36 AM	PROPATH File	2 KB
README.txt	2/17/2017 4:37 PM	Text Document	2 KB

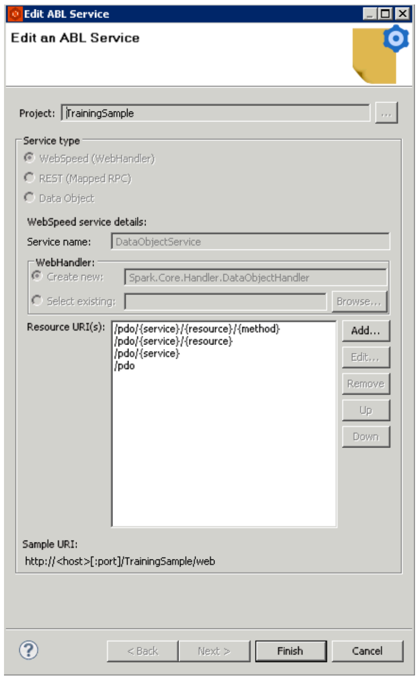
5. **Re-launch** the Progress Developer Studio (PDSOE), so that we can complete verification of our recent changes. Notably, we should see the following items to indicate success:
    a. New root folders present, such as "**Deploy**" and "**tools**".
    b. New items such as **Ccs.pl** and **PMFO.pl** files under the *AppServer* folder.
    c. The **PROPATH** should now reflect the addition of these 2 procedure libraries.
    d. Double-clicking on the defined service "**DataObjectService**" should show several *Resource URI(s)* for a *Spark.Core.Handler.DataObjectHandler* WebHandler.
    e. Presence of the file **PASOEContent/WEB-INF/openedge/Business/UserData.cls**
    f. There should be a new set of directories and files under the **PASOEContent/static** folder as well, which will fully prepare us for the client-side piece of the application.
6. We can now delete the defined service "**TrainingSampleService**" initially created. Also **PASOEContent/WEB-INF/openedge/TrainingSampleHandler.cls** may safely be deleted.
    a. Under "*Defined Services*" right-click on "*TrainingSampleService*" and select **Delete**.
    b. Under the **PASOEContent/WEB-INF/openedge/** folder, delete the file named "*TrainingSampleHandler.cls*"

7. Open the file **AppServer/startup.pf** via *Text Editor* add the following database connections:

```
-db webstate -H localhost -S 8500
-db sports2000 -H localhost -S 8600
```



```
DynSports
TrainingSample
  Procedure Libraries
  AppServer
    build.xml
    Ccs.pl
    merge.openedge.properties
    PMFO.pl
    startup.pf
  Deploy
    Build
    Conf
    README.txt
  PASOEContent
  tools
  README.txt
  Defined Services
    DataObjectService
```



Edit ABL Service

Edit an ABL Service

Project: TrainingSample

Service type
- [✓ WebSpeed (WebHandler)]
- REST (Mapped RPC)
- Data Object

WebSpeed service details:

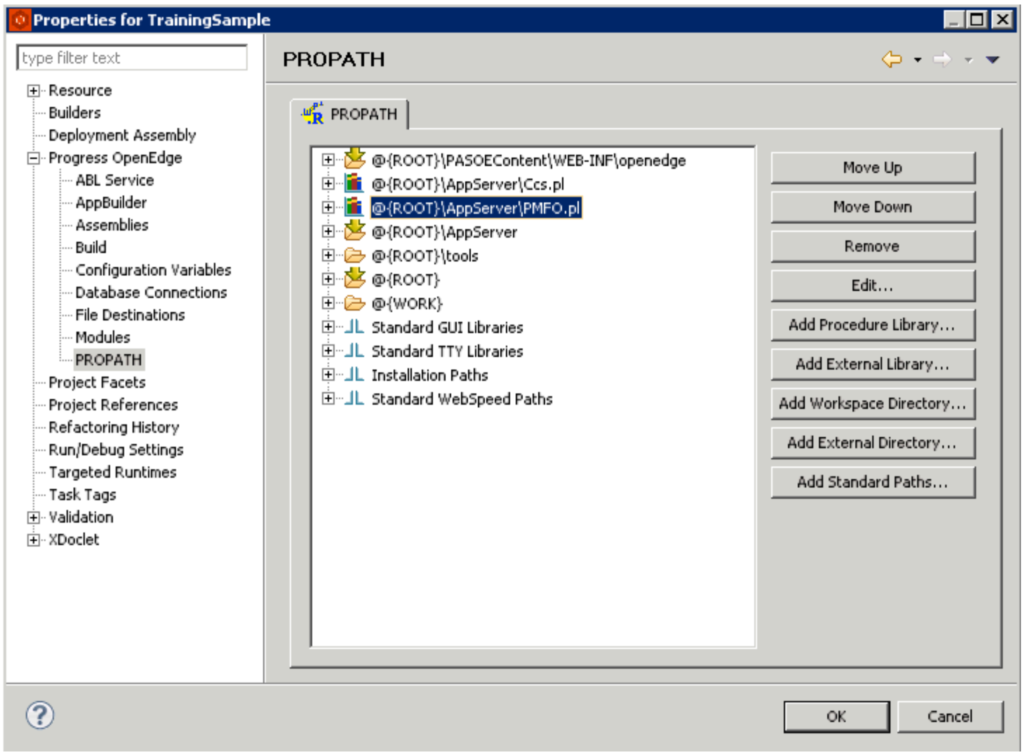
Service name: DataObjectService

WebHandler:
- [✓ Create new:] Spark.Core.Handler.DataObjectHandler
- Select existing:

Resource URI(s):
```
/pdo/{service}/{resource}/{method}
/pdo/{service}/{resource}
/pdo/{service}
/pdo
```

Sample URI:
http://<host>[:port]/TrainingSample/web

Above, right, and below: Screenshots of several
visible differences that resulted from our Copy/Paste operation from TabulaRasa.



Properties for TrainingSample

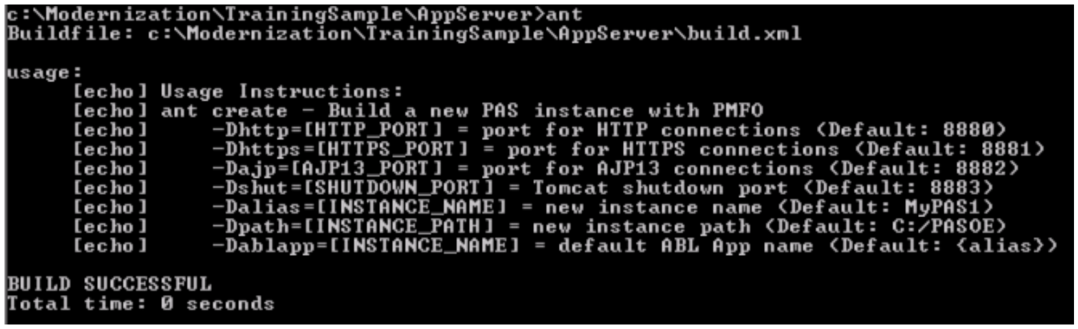
PROPATH

```
@{ROOT}\PASOEContent\WEB-INF\openedge
@{ROOT}\AppServer\Ccs.pl
@{ROOT}\AppServer\PMFO.pl
@{ROOT}\AppServer
@{ROOT}\tools
@{ROOT}
@{WORK}
Standard GUI Libraries
Standard TTY Libraries
Installation Paths
Standard WebSpeed Paths
```

Creating a PASOE Instance

At this point we have a project but nothing to publish it to. We need to create a new PASOE instance and update our project's configuration. Before we begin we need to choose a port where this server will reside. OpenEdge ships with a default "*oepas1*" instance which lives on port 8810, and your virtual machine should already have a "*SportsPASOE*" that was created on 8820. So, for our purposes we'll start with both a different name and alternate ports, starting with 8830 and adjusting as necessary for the remaining ports.

1. To standardize the process of creating the necessary PAS instance, this has been scripted using ANT. To run the script, first open a Windows Command Prompt as Administrator:
    a. Click the Command Prompt icon in the taskbar (shown at right).
    b. Use the admin password **Pr0gress2016** when prompted.
    c. Navigate to **C:/Modernization/TrainingSample/AppServer**
2. From the directory noted above, run "ant" to see usage instructions as follows:



```
c:\Modernization\TrainingSample\AppServer>ant
Buildfile: c:\Modernization\TrainingSample\AppServer\build.xml

usage:
     [echo] Usage Instructions:
     [echo] ant create - Build a new PAS instance with PMFO
     [echo]      -Dhttp=[HTTP_PORT] = port for HTTP connections (Default: 8880)
     [echo]      -Dhttps=[HTTPS_PORT] = port for HTTPS connections (Default: 8881)
     [echo]      -Dajp=[AJP13_PORT] = port for AJP13 connections (Default: 8882)
     [echo]      -Dshut=[SHUTDOWN_PORT] = Tomcat shutdown port (Default: 8883)
     [echo]      -Dalias=[INSTANCE_NAME] = new instance name (Default: MyPAS1)
     [echo]      -Dpath=[INSTANCE_PATH] = new instance path (Default: C:/PASOE)
     [echo]      -Dablapp=[INSTANCE_NAME] = default ABL App name (Default: <alias>)

BUILD SUCCESSFUL
Total time: 0 seconds
```

3. Executing the "**ant create**" command will create a PAS instance called "MyPAS1" at the noted ports and residing at the noted path. However, **we MUST change these default options** to agree with our forthcoming examples and expected ports, using the command line below:

    ant create -Dhttp=8830 -Dhttps=8831 -Dajp=8832 -Dshut=8833 -Dalias=TrainPAS

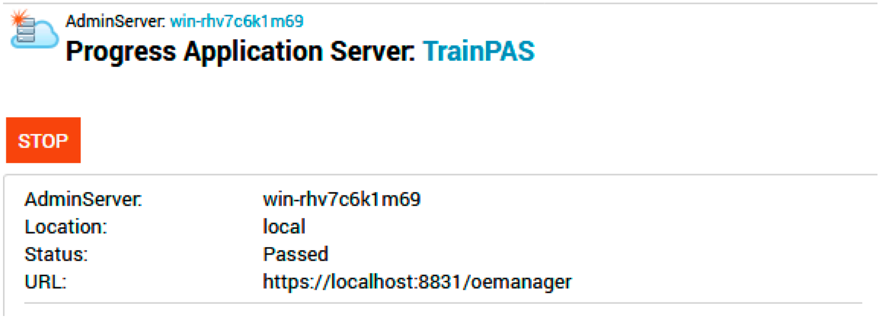
4. Running the above command should perform the creation of the new PAS instance via the "**pasman**" binary, while dumping output in a TXT file in the same directory. In addition:
    a. The new instance will be configured using the ports and name specified above.
    b. All PMFO configuration files will be copied to their expected location under CATALINA_BASE/conf, and PL/PF files copied to CATALINA_BASE/openedge
    c. The script will run "**oeprop**" to merge the file "*merge.openedge.properties*". This will configure the ABL portion of the application with the necessary relative PROPATH entries, and Session Startup/Shutdown procedures.
5. Return to the command prompt window, and start the instance with the command below:

```
pasman start –I TrainPAS
```

This should open a new (Java) command window, which will show the process of starting the Tomcat server with accompanying AppServer. When you see a line "**Server startup in NNNNN ms**" that means the server is now fully operational. We can confirm this via several means:

1. From your original command prompt, run "*pasman plist –I TrainPAS*" to see the PID's in use. One is the **java.exe** process running Tomcat, and the other is the **_mproapsv.exe** which is your MSAS (Multi-Session AppServer) Agent.

2. From within the OpenEdge Management Console (aka. OpenEdge Explorer) you can view the status of your server by looking under the “Progress Application Server” section.



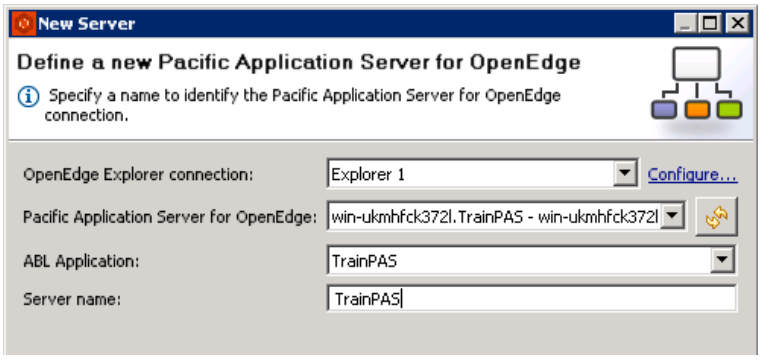
AdminServer: win-rhv7c6k1m69
Progress Application Server: TrainPAS

AdminServer:	win-rhv7c6k1m69
Location:	local
Status:	Passed
URL:	https://localhost:8831/oemanager

3. Within your PDSOE project, when the server is added to the **Servers** view. That has not yet been done, so we will configure that next.

Return to your PDSOE workspace, and perform the following steps:

1. Select **Window** → **Show View** → **Servers**
2. **Right-click** in the Servers view and select **New** → **Server**
3. Select the type “**Progress Application Server for OpenEdge**” and click **Next**.
4. Select your recently-created “**TrainPAS**” server from the 2nd dropdown, and shorten the server name to just “*TrainPAS*”. Click the **Next** button to continue.



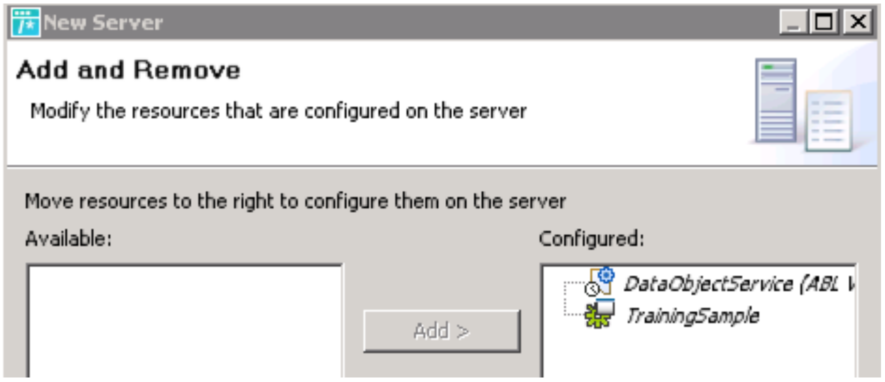
Define a new Pacific Application Server for OpenEdge

Specify a name to identify the Pacific Application Server for OpenEdge connection.

OpenEdge Explorer connection:	Explorer 1
Pacific Application Server for OpenEdge:	win-ukmhfck372l.TrainPAS - win-ukmhfck372l
ABL Application:	TrainPAS
Server name:	TrainPAS

5. From the options on the left, we’ll select our newly created project’s services. These should be the *WebApp* simply called “**TrainingSample**” and the *Service* fully qualified as “**DataObjectService (ABL WebSpeed Service) → TrainingSample**”.

   a. When these are added to your server, click **Finish**.



Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

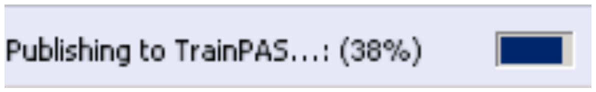
Available:

Configured:

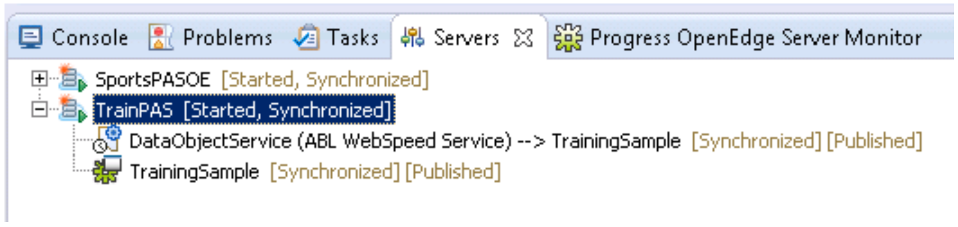
DataObjectService (ABL V

TrainingSample

6. Watch the bottom-right section of your PDSOE application to know if the system is working on publishing your code to the server instance.


Publishing to TrainPAS...: (38%)

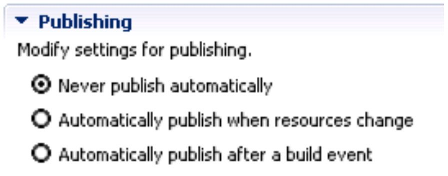
7. Upon completion, your project's WebApp and the Defined Service should be published to the PAS instance. Please be patient as this step may take a few minutes. You can monitor using the progress bar at the bottom right of the Eclipse workspace.
   a. If all goes well, you should see the status as "*Started*" and "*Synchronized*".
   b. If the servers appears as "*Stopped*", right-click and select "*Start*".
   c. If the server refuses to start, remove all files from the **C:/PASOE/TrainPAS/logs** folder and try to start again. **Reason:** If a stale **catalina-<instance_name>.pid** file exists, it may cause the server to think another instance is still running. If another instance is truly running, you would receive errors about the files being unable to be deleted.



SportsPASOE [Started, Synchronized]
TrainPAS [Started, Synchronized]
DataObjectService (ABL WebSpeed Service) --> TrainingSample [Synchronized] [Published]
TrainingSample [Synchronized] [Published]

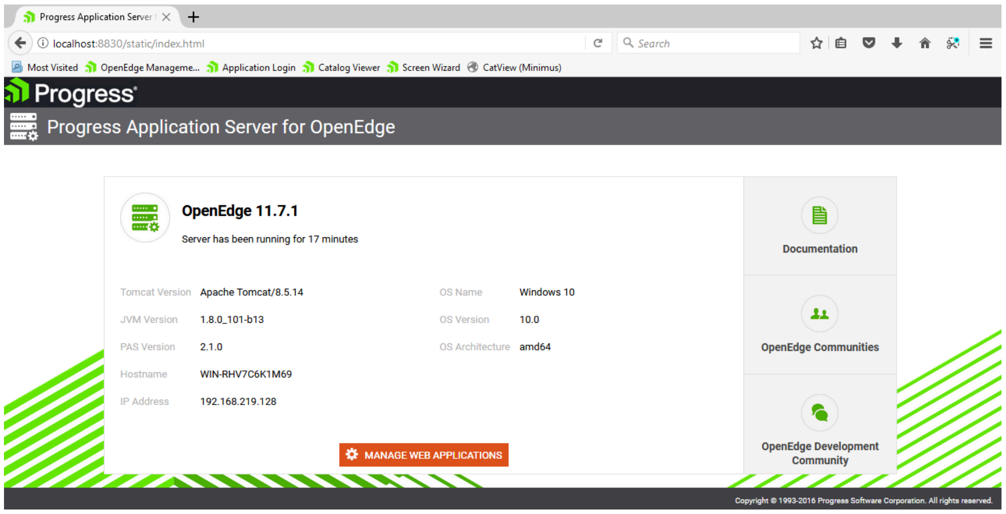
8. As part of the ANT script process, our instance has been configured already with the "*startup.pf*" file we modified for databases, and PROPATH entries for including the CCS/PMFO libraries, so we don't need to specify any further settings at this point.

   However, we should disable one option in PDSOE as follows:
   a. **Double-click** on the "*TrainPAS*" instance to open the property sheet.
   b. At the top-right, expand "*Publishing*" and select "*Never publish automatically*". Press **Ctrl+S** to save your changes (or use File → Save).


Publishing
Modify settings for publishing.
[✓ Never publish automatically]
Automatically publish when resources change
Automatically publish after a build event

   c. You are now in control of when to publish files to the PAS instance.

9. We should be able to now test that our server is running property by visiting at the URL http://localhost:8830 to view the default landing page.
   a. Just to note, this page is served by the ROOT WebApp.
   b. Clicking on the "*Manage Web Applications"* will take us to the Tomcat manager.
   c. This should ONLY be made available within a Development environment!
   d. The default username/password for this feature is "tomcat/tomcat".


Progress Application Server
localhost:8830/static/index.html
Progress
Progress Application Server for OpenEdge
OpenEdge 11.7.1
Server has been running for 17 minutes
Tomcat Version Apache Tomcat/8.5.14
JVM Version 1.8.0_101-b13
PAS Version 2.1.0
Hostname WIN-RHV7C6K1M69
IP Address 192.168.219.128
OS Name Windows 10
OS Version 10.0
OS Architecture amd64
MANAGE WEB APPLICATIONS
Documentation
OpenEdge Communities
OpenEdge Development Community
Copyright © 1993-2016 Progress Software Corporation. All rights reserved.

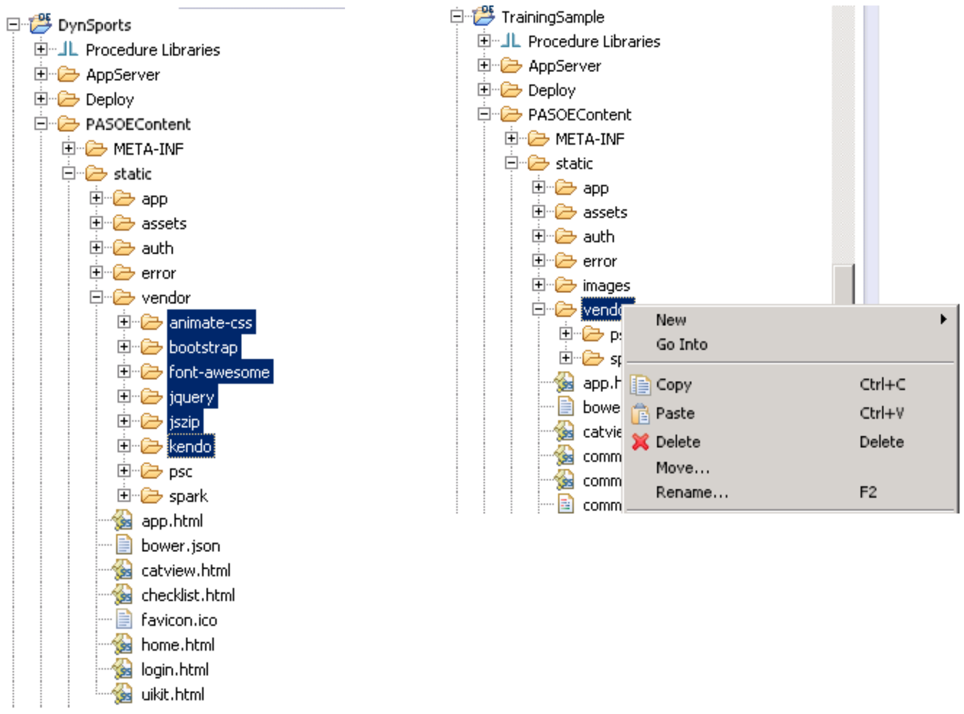
10. Now we need some additional 3rd-pary, client-side libraries which we will simply copy from an existing project and publish to our PAS instance. For this step, navigate to the **DynSports/PASOEContent/static/vendor** directory and do the following:

    a. Select: **animate-css, bootstrap, font-awesome, jquery, jzip, kendo**

    b. Right-click and **Copy** these selected folders.

    c. Navigate to **TrainingSample/PASOEContent/static/vendor**

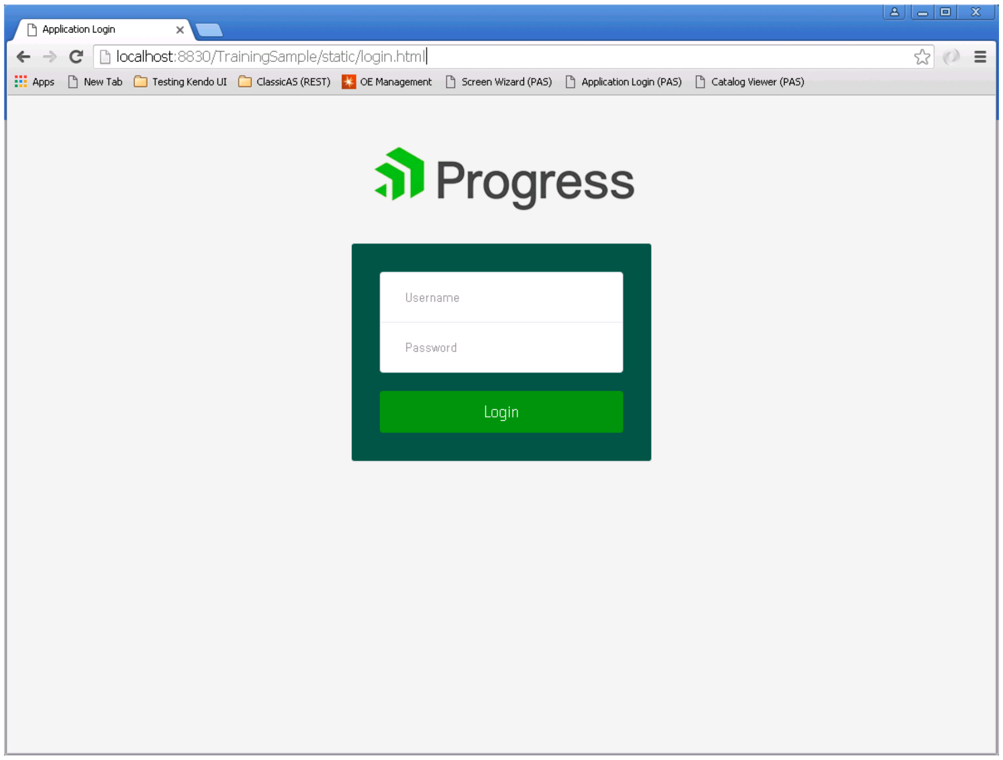
    d. Right-click and select **Paste** these selected items.



11. With the recent changes to our new project, we will need to publish these to our PAS instance.

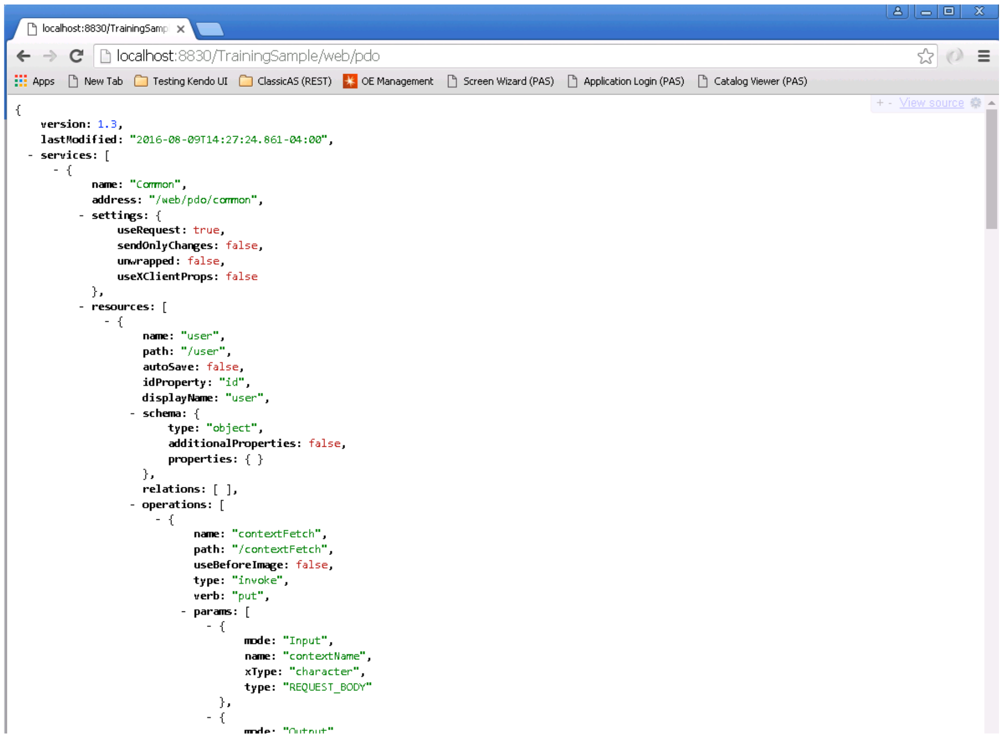
    a. In the *Servers* view, **right-click** on the *TrainPAS* instance and select "**Publish**".

12. Once published and synchronized, we should be able to test our "**TrainingSample**" WebApp by visiting **http://localhost:8830/TrainingSample/static/login.html** to view the login page.



13. As a final test, we should attempt to reach our WebHandler which should serve up a basic catalog of services that have been exposed: **http://localhost:8830/TrainingSample/web/pdo**

    You should see a JSON payload delivered to your browser. If not, then you will need to retrace your steps to make sure all files were copied to your new project as expected.



```
{
    version: 1.3,
    lastModified: "2016-08-09T14:27:24.861-04:00",
  - services: [
      - {
            name: "Common",
            address: "/web/pdo/common",
          - settings: {
                useRequest: true,
                sendOnlyChanges: false,
                unwrapped: false,
                useXClientProps: false
            },
          - resources: [
              - {
                    name: "user",
                    path: "/user",
                    autoSave: false,
                    idProperty: "id",
                    displayName: "user",
                  - schema: {
                        type: "object",
                        additionalProperties: false,
                        properties: { }
                    },
                    relations: [ ],
                  - operations: [
                      - {
                            name: "contextFetch",
                            path: "/contextFetch",
                            useBeforeImage: false,
                            type: "invoke",
                            verb: "put",
                          - params: [
                              - {
                                    mode: "Input",
                                    name: "contextName",
                                    xType: "character",
                                    type: "REQUEST_BODY"
                                },
                              - {
                                    mode: "Output"
```

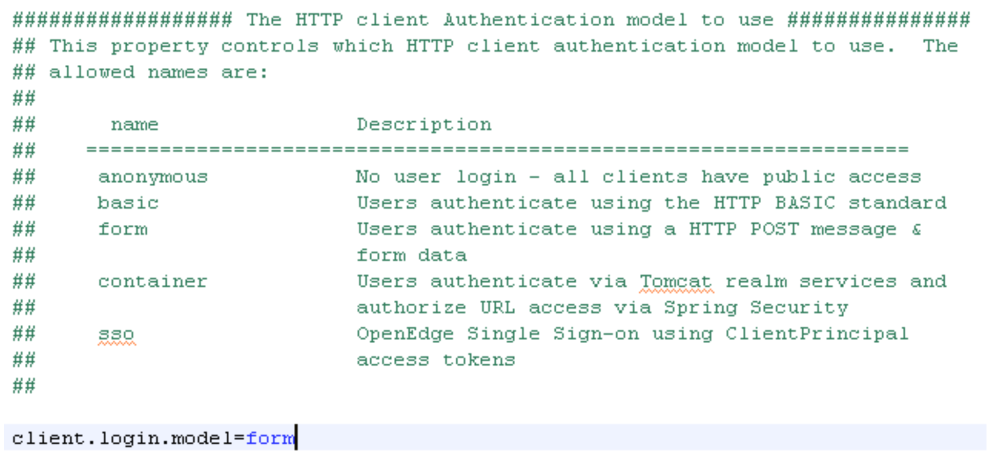
NOTE

By default, all new PAS development instances and PDSOE projects run in an unsecured mode, with a project's default Security Model set as "**Anonymous**". This is great for initial debugging purposes, but in the real world we will mostly be dealing with secured application.

Security Configuration

Once the service has been confirmed as working by accessing the catalog, it's time to look at our security configuration and upgrade to at least something that requires a username and password: **Form-Local** security. The reason for this is simple: you can always make a secured application more permissive, but you cannot easily make a public application more secure. And thanks to the items from the sample project we overlaid on our real project, we can make this change very quickly. Along the way, a few items will be highlighted to explain what was done, in case you wish to repeat these actions again on another project of your own.

1. In PDSOE, double-click on the file **PASOEContent/WEB-INF/oeablSecurity.properties**
2. Look for the name "**client.login.model**" which specifies the security model to use. Currently this is set to "anonymous". Change the value to "form" to allow credentials to be posted.



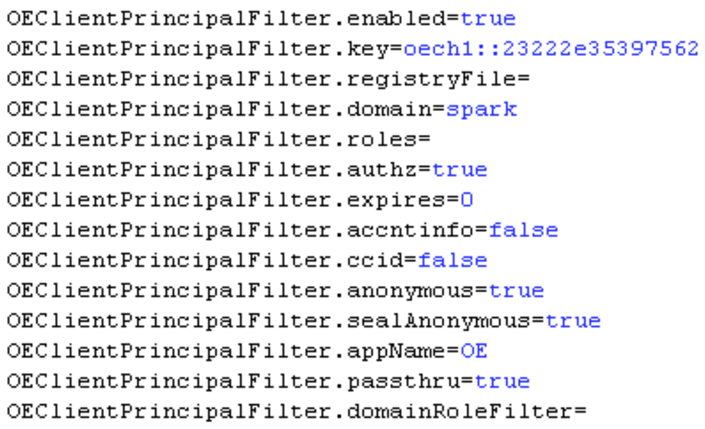
```
################## The HTTP client Authentication model to use ###############
## This property controls which HTTP client authentication model to use.  The
## allowed names are:
##
##      name                Description
##    ==================================================================
##     anonymous            No user login - all clients have public access
##     basic                Users authenticate using the HTTP BASIC standard
##     form                 Users authenticate using a HTTP POST message &
##                          form data
##     container            Users authenticate via Tomcat realm services and
##                          authorize URL access via Spring Security
##     sso                  OpenEdge Single Sign-on using ClientPrincipal
##                          access tokens
##

client.login.model=form
```

   a. Note that the property "**http.all.authmanager**" remained set to "*local*" which indicates user accounts will be defined in a flat file. The default file is "**users.properties**".

   b. The difference between the "**form**" and "**basic**" authentication models is that "form" requires credentials to be POSTed to a specific URL once for authentication, and "basic" requires a custom header to be sent with a hash of the credentials on each request.

   c. For our purposes we expect the user to login once on a specific page, and have that authenticated session used for the duration of all subsequent requests (without passing credentials each time).

3. The next crucial section in this file are the "**OEClientPrincipalFilter**" options which determine how to create and seal the CP token, which will be passed to our ABL code when "**enabled**" is "true".



```
OEClientPrincipalFilter.enabled=true
OEClientPrincipalFilter.key=oech1::23222e35397562
OEClientPrincipalFilter.registryFile=
OEClientPrincipalFilter.domain=spark
OEClientPrincipalFilter.roles=
OEClientPrincipalFilter.authz=true
OEClientPrincipalFilter.expires=0
OEClientPrincipalFilter.accntinfo=false
OEClientPrincipalFilter.ccid=false
OEClientPrincipalFilter.anonymous=true
OEClientPrincipalFilter.sealAnonymous=true
OEClientPrincipalFilter.appName=OE
OEClientPrincipalFilter.passthru=true
OEClientPrincipalFilter.domainRoleFilter=
```

a. This asserts our identity against any connected databases, which are configured to use a "**domain**" called "*spark*" for Single Sign-On.

b. For each domain we need a passphrase, which is set here by the "**key**" property. This is an encrypted key which allows us to place it in the configuration file without risk.

c. Note: If our application worked with a multi-tenant database, we would instead use a "**registryFile**" to define all domains, leaving "*domain*" and "*key*" blank.

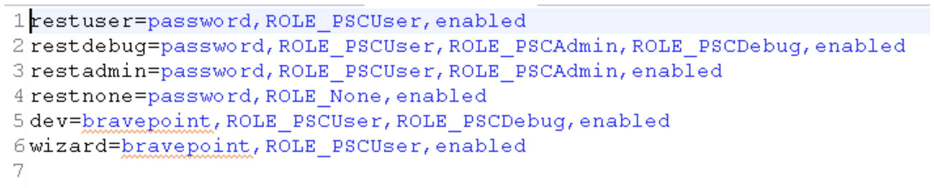
d. To allow us to pass an anonymous token we have both the "**anonymous**" and "**sealAnonymous**" properties set to true. And the property "**passthru**" indicates that the CP token should always be sent, even in the event of an anonymous user.

e. The option "**authz**" indicates that we should load any associated roles for the user account as "*authorities*" in the sealed CP token.

4. Let's confirm the setup of the "**users.properties**" file by opening that via double-click.

    a. This is where we have our default users specified for our project. We can see that there are some default "rest" accounts, as well as our application's two crucial "**dev**" and "**wizard**" accounts.

    b. Note that the user account "**dev**" has a password "**bravepoint**" in clear text. While it is possible to use hashed values here, it is both beyond the scope of this material and more effective to setup **Form-OERealm** security to authenticate against a database.

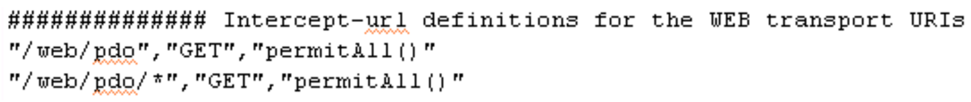


```
restuser=password,ROLE_PSCUser,enabled
restdebug=password,ROLE_PSCUser,ROLE_PSCAdmin,ROLE_PSCDebug,enabled
restadmin=password,ROLE_PSCUser,ROLE_PSCAdmin,enabled
restnone=password,ROLE_None,enabled
dev=bravepoint,ROLE_PSCUser,ROLE_PSCDebug,enabled
wizard=bravepoint,ROLE_PSCUser,enabled
```

5. Finally, open **PASOEContent/WEB-INF/oeablSecurity.csv** via double-click on the file.

    a. This file contains a special pattern that defines the intercept rules for URI's that may be accessed within our application. These are applied in order of appearance, determining any public access or special security that must be applied.

    b. The following lines allow any user, authenticated or anonymous, to access the URL patterns /pdo or /pdo/{service} via only the GET method.



```
############## Intercept-url definitions for the WEB transport URIs
"/web/pdo","GET","permitAll()"
"/web/pdo/*","GET","permitAll()"
```

c. The last intercept rule should always place a default behavior for a transport which requires any requesting user to have a minimum role to access (here, "*PSCUser*").

```
"/web/**","*","hasAnyRole('ROLE_PSCUser')"
```

d. Note that there is a special role called "**ROLE_ANONYMOUS**" which can be used to grant access to users who have not yet authenticated. These are referred to as "anonymous" users.

If you make changes to any of these files, upon closing them you would notice the PAS server status change to **[Started, Republish]** indicating that you must publish the changes to the server. **Additionally, you would need to restart the Tomcat server to pick up the changes in security model!** After restarting the PAS instance, you should now be able to log in to the application using the "**dev/bravepoint**" account as found above in the "*users.properties*" file.

1. From the "**Servers**" view, **Right-click** on *TrainPAS*, and select **Restart**.
2. Wait for the server to re-start and attempt a login to the application.

NOTE

If there are still any issues with running the application at this point, don't forget to consult the "**Troubleshooting Checklist**" in **Appendix A**. This must absolutely be in working order before we continue with the remainder of the material. **If you performed a copy/paste of any quoted strings above, then you may need to replace the typographer's quotes (as found in this document) with standard double-quotes.**

NOTE

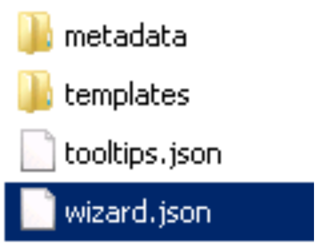
It is not necessary that every connected database contain the same or full set of domains in order to operate, though that is the pattern used for our purposes here. At the very least there must be one database which contains all domains you wish to utilize for your application. This will automatically become the default, or authoritative, database for loading domains. All other connected databases must utilize the Database Option "Use Application Domain Registry" to allow that connected database to trust the domains loaded by the application.

Incorporating the Wizard

At this point we should have a usable application, but we need to populate it with our own back-end logic and UI screens. To that end, we have a "Screen Wizard" that we can incorporate into our new PAS instance! We'll deploy our generic "**Web Wizard**" to the server, and then tailor it for our needs.

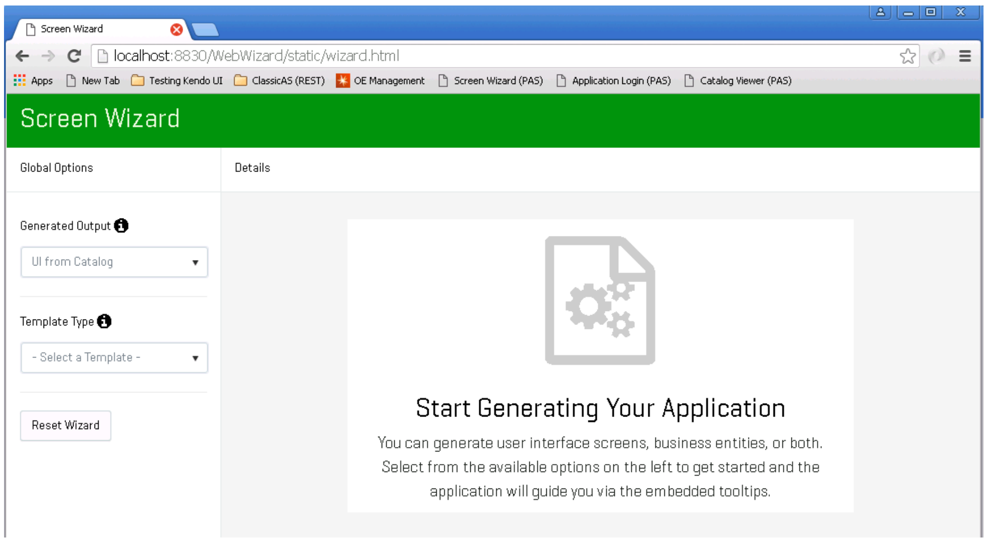
1. From a command window (used previously for creating the PAS instance, change the directory (via "**cd**" command) to **C:/PASOE/TrainPAS/bin**
2. Run the command "`tcman stop`" to ensure the PAS instance is not currently running.
3. Run the following command to deploy the WAR file to this PAS instance:

   ```
   tcman deploy C:/Modernization/Spark-Demo/oe117/WebWizardPAS/WebWizard.war
   ```
4. Before we start our PAS instance, we should update the config files for our new WebApp.
   a. Navigate to the **C:/PASOE/TrainPAS/webapps/WebWizard/config** folder.

   b. Open the **wizard.json** file with Notepad++ and change the following:
      i. "DefaultCatalog": "http://localhost:8830/TrainingSample/web/pdo"
      ii. "OutputBase": "C:/Modernization/TrainingSample"
      iii. "OutputConf": "C:/PASOE/TrainPAS/conf/spark"

   c. Save the file and return to PDSOE.
5. With the WAR file successfully deployed and configs tailored to this instance, start the PAS instance via the "**Servers**" view in PDSOE (**Right-click** on *TrainPAS*, select **Start**).
6. When the server is finally "**Started, Synchronized**" we can test our new WebApp by navigating in Chrome to **http://localhost:8830/WebWizard/** to confirm that you see the following page. If you see a loading bar at the top of the screen at 0% don't panic, just press refresh to make sure the server API's are responding after the server restart.



Screen Wizard

Generated Output: UI from Catalog

Start Generating Your Application

You can generate user interface screens, business entities, or both. Select from the available options on the left to get started and the application will guide you via the embedded tooltips.

So now we can discuss what the wizard does! In simplest terms, it allows you to select a piece of pre-built code which contains special tokens, tailor various options for your needs, and perform a replacement of those tokens within the templates to produce runnable code. Together these templates can create either stand-alone screens a complete single-page application (SPA), complete with back-end logic for the REST adapter.

The wizard interface allows for the initial choice of only 2 options: generate a "*BE Class File*" or "*UI from Catalog*". The ideal order of operations is to create your ABL Business Entity (BE) for the server, expose it by publishing to the PAS instance, then produce a UI screen based on data from the newly generated catalog data. This allows you to modify your BE's schema (if needed) and guarantees that the parameters and schema reported by the catalog are exactly what you can work with from a UI perspective.

NOTE

As of the 3.0.0 release of PMFO, selection of an existing dataset/temp-table definition file (residing in the Common directory) and the generation of a BE against that schema is possible (with upload ability still TBD). Of course, there is no reason why you cannot create your own custom BE using your own schema, by hand. As long as you adhere to use of either the **DynamicResource** or **DynamicEntity** classes as the inherited class, the PMFO will be able to utilize your custom logic.

Template Availability

There are several pre-built templates within the wizard that should deliver common and consistent views for a typical real-world application. These are by no means comprehensive of all features of KendoUI or of what one may require for a project, but they illustrate many common concepts.

- **Login/Index (SPA)** – Creates a standard login page for a Single-Page Application (SPA).
- **Landing/Dashboard (SPA)** – Creates a standard landing page for an SPA containing a splash page (or dashboard) and simple site navigation for all other SPA templates. Note: To assist with the generation process, entries for navigation will be created automatically when needed as new SPA screens are generated.
- **Read-Only Grid (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Updateable Grid (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Updateable Form (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Grid & Form (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Grid Inside Grid (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Grid Over Grid (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **Blank Screen (SPA)** – Identical to the original template above, but meant for use in an SPA.
- **From Metadata** – Allows quick re-generation of a past template with all selected options via stored metadata files. Note: The currently stored metadata is based on what the existing wizard requires, and may be subject to change in future versions (though efforts to support backwards compatibility will always be attempted).

The following are functionally similar to the SPA screens above, but generate a standalone HTML page. For our tasks, we will not be making use of these, but remain for demo purposes only.

- **Read-Only Grid** – A screen that allows searching and showing records in a grid format, similar to the Progress GUI BROWSE widget, but with many more configurations and user options.
- **Updateable Grid** – Like the read-only grid, this allows searching and browsing records but it also allows updating, deleting, and adding new records. Again, the Kendo Grid provides many configuration and user options.
- **Updateable Form** – A multi-field screen that allows searching for a single record, then making changes to it, or deleting it. It also allows adding a new record.
- **Grid & Form** – Combines a Kendo Grid to allow search and navigation, then updating or deleting a record via multi-field form. It also allows adding a new record.
- **Grid Inside Grid** – Contains a master grid with parent records that expands to display a detail grid with child records. This layout is also known as Master-Detail.
- **Grid Over Grid** – Similar in function to the Grid Inside Grid template, but displays the master grid above the detail grid together on the same screen.
- **Blank Screen** – A base template that produces a completely blank page, used for custom screens.
- **Run My Progress Code** – Allows refactoring of an existing business logic procedure to be exposed as a business entity. Currently produces a grid interface.

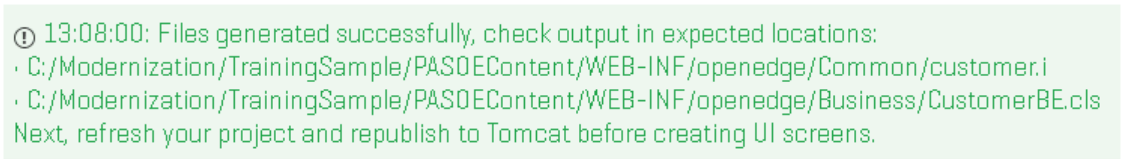
NOTE

As a safeguard against overwriting code that has been customized after generation, so long as the metadata files are kept after a screen is generated the wizard interface should be able to detect and alert you if an overwrite operation will occur. You can then determine if you want to keep the old file or replace it.

Generating Business Entities

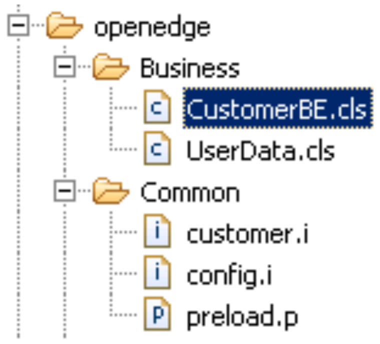
This is the simplest option in the wizard, as it has the fewest options necessary to complete.

1. From the "*Generated Output*" field select "*BE Class File*".
2. On the enabled "*Data Source*" field select "*Database Table*".
3. For the "*Service URI*" you may leave this as the default.
4. Select an option from the "*Database*" field (you should have **Sports2000** available) and note that clicking on the "*Entity Table(s)*" field will display a multi-select for the available tables.
    a. You may choose 1 or more distinct table to generate, or leave the field blank to generate a BE for every table present.
    b. For our purposes start small and select only **Customer**.
5. Click on the **Generate** button to run the actual code generation routine.
    a. Note that the operation should return a status indicating that files were generated, and where on the filesystem they were deposited. Next we can expose this entity.

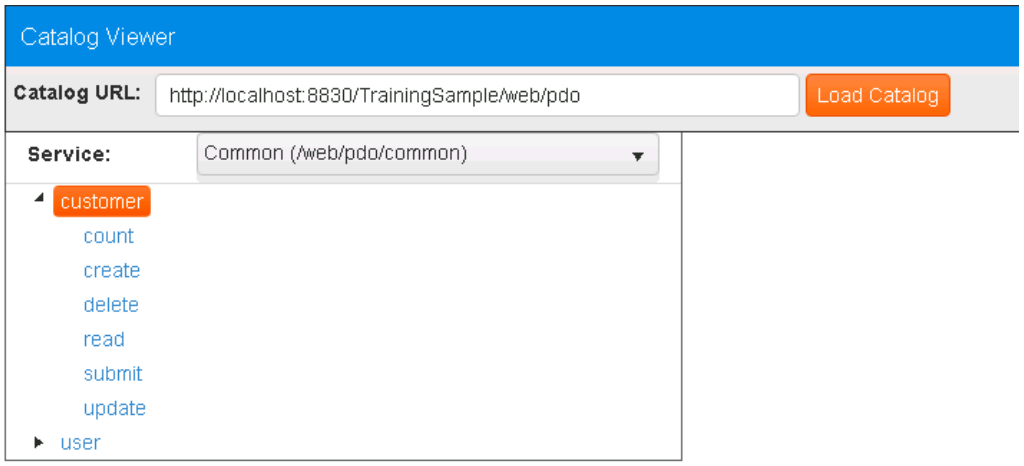

```
13:08:00: Files generated successfully, check output in expected locations:
· C:/Modernization/TrainingSample/PASOEContent/WEB-INF/openedge/Common/customer.i
· C:/Modernization/TrainingSample/PASOEContent/WEB-INF/openedge/Business/CustomerBE.cls
Next, refresh your project and republish to Tomcat before creating UI screens.
```

With the files generated and on disk, we can return to PDSOE to expose them via the PAS instance.

1. Within PDSOE, press F5 to refresh the project. Confirm that "*Customer.cls*" and "*customer.i*" are now present as indicated within the "**/PASOEContent/WEB-INF/openedge/Business**" and "**Common**" directories.
    a. **NOTE**: If you intend to modify the schema of your Business Entity, this is the point at which you should do that. (eg. Adding fields, removing fields, changing labels, etc. to produce what gets read by PDSOE as this will become your schema for the UI portion.)



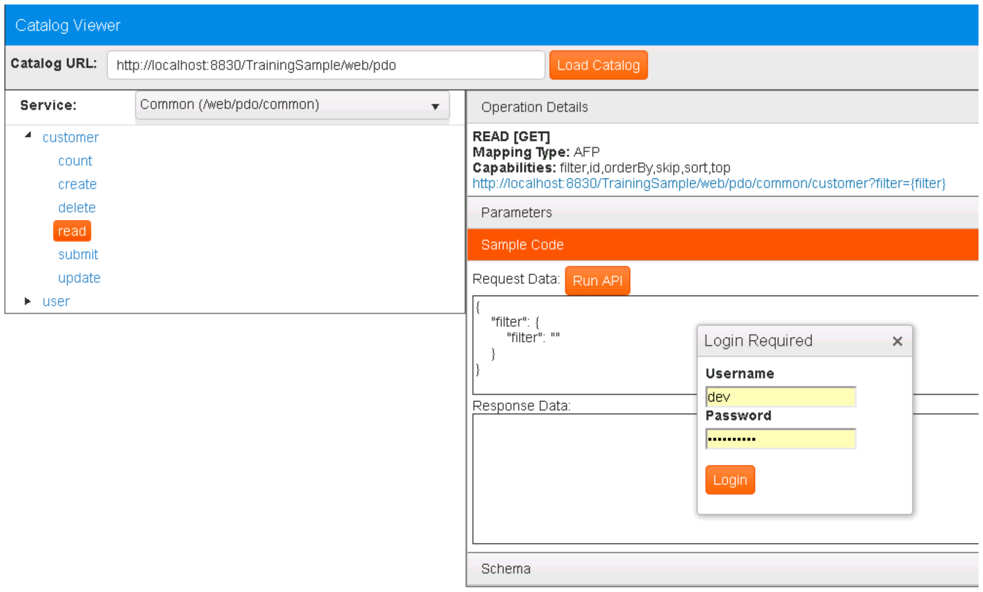
2. Right-click on the **openedge** directory and select **Progress OpenEdge → Compile** to make sure the files can compile without error. We should know if there are any obvious syntax errors before we push and test our code!
3. **Right-click** on the *TrainPAS* server instance in the **Servers** view, and select **Publish**. The new code will be pushed to our server and any current sessions trimmed automatically (the latter is important to us as it removes any singleton/static classes from memory).
4. In Chrome, navigate to **http://localhost:8830/TrainingSample/static/catview.html** to access the Data Object Service Catalog Viewer.
    a. You should now see a **customer** resource on the left. (If you already had a tab open to this page, click on the **Load Catalog** button to refresh from the latest catalog.)



Catalog URL: http://localhost:8830/TrainingSample/web/pdo

Service: Common (/web/pdo/common)

5. We can now test our service by clicking on the "**read**" method available within the **customer** resource (click on the name to expand). You'll be presented with a section stack on the right, click on the "**Sample Code**" option to expand that view. Just click on the "**Run API**" button to execute a call to the resource.
   a. You may be prompted to log in if you have not yet authenticated. If presented with a login, use "**dev**" and "**bravepoint**" as your credentials and press "**Login**".
   b. After logging in, just press the "**Run API**" button again to execute the query.



Catalog URL: http://localhost:8830/TrainingSample/web/pdo

Service: Common (/web/pdo/common)

READ [GET]
Mapping Type: AFP
Capabilities: filter,id,orderBy,skip,sort,top
http://localhost:8830/TrainingSample/web/pdo/common/customer?filter={filter}

Request Data:

```
{
    "filter": {
        "filter": ""
    }
}
```

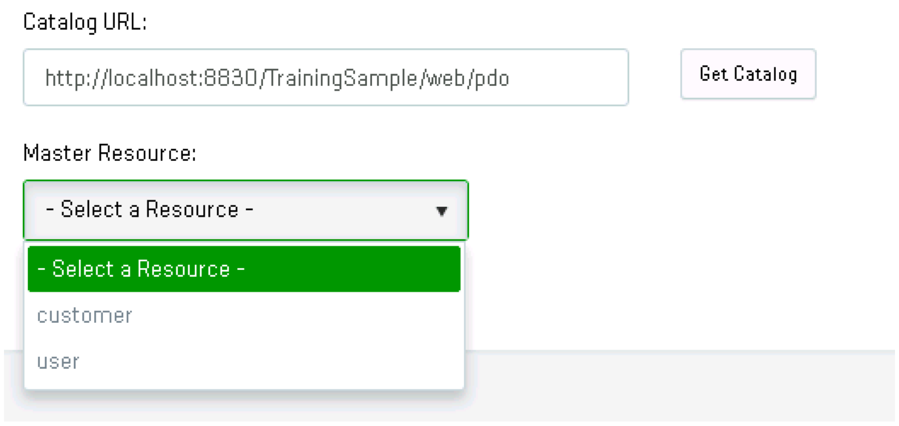
Username: dev
Password: ••••••••••

6. If there were no errors, you should see **Response Data** in JSON format that contains reference to a "**dsCustomer**" object with "**ttCustomer**" array. This means we can now create an SPA screen for our back-end resource.

Generating SPA Screens

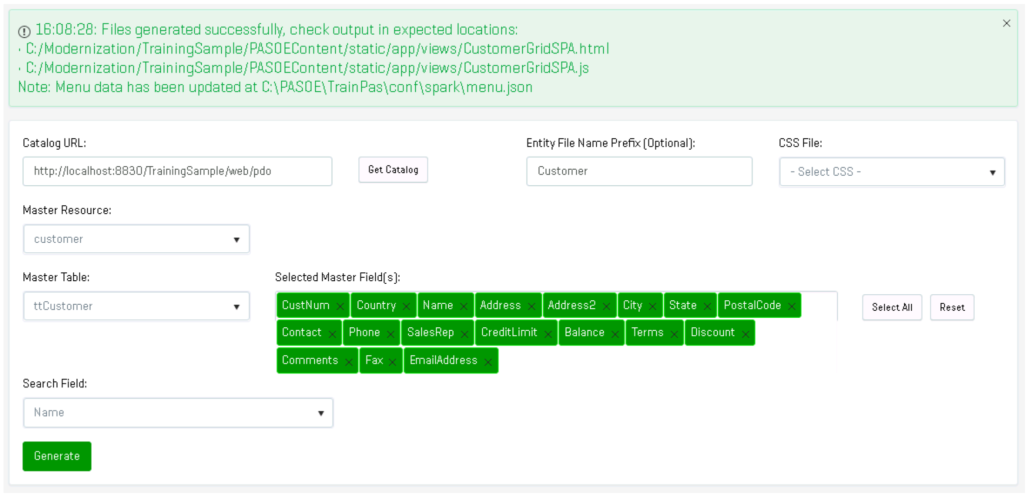
With at least a single Business Entity behind our RESTful service (the *DataObjectHandler* available as WebHandler), we should have something to utilize for building a UI screen. We will do that next via the Screen Wizard tool.

1. From the "*Generated Output*" field select "UI from Catalog".
2. Below this field is the "Template" dropdown, which contains a list of the available templates as noted above. For our coming samples, it is necessary for us to use the SPA pattern.
    a. We already copied a standard (pre-generated) login page, so we should **NOT** need to utilize the "*Login/Index (SPA)*" template, but that is what the option provides.
    b. Likewise, we already have a proper application landing page, so we will **NOT** need to create a "*Landing/Dashboard (SPA)*" screen, but that is what the option provides.
    c. If you need or want to re-generate these, there are no options for these screen types, so simply press the **Generate** button. If you get a message about files already existing, just click **OK** to continue, as you are **knowingly overwriting** the last copy.
    d. Note that the files should be generated and placed directly in the **/PASOEContent/WEB-INF/static/** folder as "**login.html**" and "**app.html**".
3. For our purposes, let's create our first screen by selecting the "*Read-Only Grid (SPA)*" template. Note: This template requires additional properties to be set, so the *Generate* button should be disabled at present.
4. The first visible field is the "*Catalog URL*" entry. This should be pre-set to a URL that should be the location of your catalog. Simply click on the **Get Catalog** button to retrieve it.
    a. If the URL reflects a WebApp name of "**DynSports**" simply change this to our current project's "**TrainingSample**" name. Our change to the wizard.json file should already have adjusted for this.
5. If the catalog was successfully retrieved, the field "*Master Resource*" should become available and populated with a list of available resources. Select the option "*customer*".


Catalog URL: http://localhost:8830/TrainingSample/web/pdo

6. This should now unlock more fields, specifically the "*Master Table*" and "*Selected Master Field(s)*" options. Select the only table option of "*ttCustomer*" (which came from our schema data in the catalog) and you may leave the selected fields blank.
    a. Alternatively, if you wish to select specific fields (as leaving this blank will use all fields) you can click on the "*Selected Master Field(s)*" field to get a dropdown of options. This is a multi-select field, allowing you to click to select the specific fields you wish to use in your UI. Note that the order of selection **does** translate to the order of the fields on the screen after generation.

7. The last enabled option should be the “*Search Field*” which is a special designation for the field used in all queries on this screen. For our table here, we should be able to select the “*Name*” option, which will allow for searches on customer name.
8. At this point the **Generate** button should be enabled, so click the button.
   a. Generation of an SPA screen should produce 2 files: an HTML and a JS. It should have also updated our menu structure, which we will confirm in the next section.



```
16:08:28: Files generated successfully, check output in expected locations:
· C:/Modernization/TrainingSample/PASOEContent/static/app/views/CustomerGridSPA.html
· C:/Modernization/TrainingSample/PASOEContent/static/app/views/CustomerGridSPA.js
Note: Menu data has been updated at C:\PASOE\TrainPas\conf\spark\menu.json
```

Catalog URL: http://localhost:8830/TrainingSample/web/pdo

Entity File Name Prefix (Optional): Customer

CSS File: - Select CSS -

Master Resource: customer

Master Table: ttCustomer

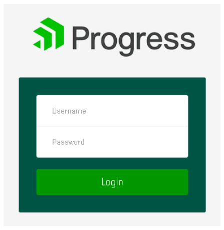
Selected Master Field(s): CustNum, Country, Name, Address, Address2, City, State, PostalCode, Contact, Phone, SalesRep, CreditLimit, Balance, Terms, Discount, Comments, Fax, EmailAddress

Search Field: Name

9. The last thing to be done is to publish the new code to our PAS instance. This is done in the same manner as our business logic that was generated previously. Return to PDSOE and press F5 to refresh any changes on disk. You can then **Right-click** on the **TrainPAS** server and select **Publish** to update the static files.

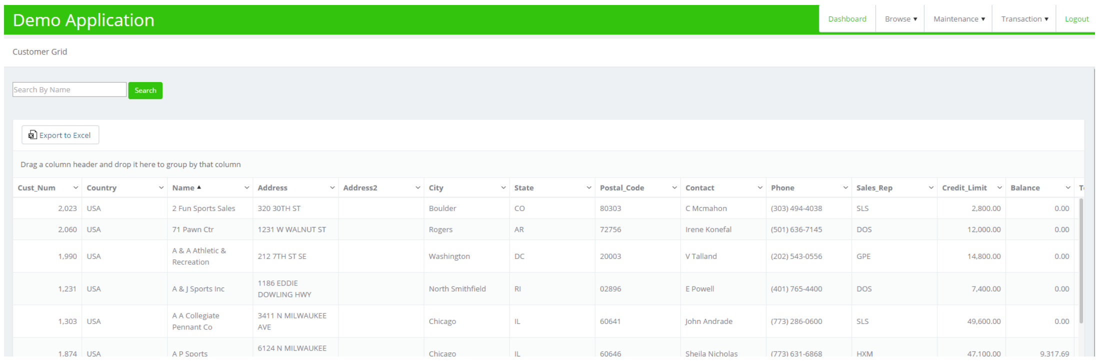
That’s really all there is to it. You can generate more screens by following the same steps, and just selecting the values as needed. Certain templates will require more selections, but those will be revealed on an as-needed basis depending on the type of template and what options are selected on-screen. When the necessary minimal options are selected, the **Generate** button will become enabled for you to create the new screen. But for now this concludes our initial screen generation, and now we can begin to run the output and test our new Single-Page Application!

Running Your Application



From your web browser, open a new tab directly to the SPAlogin page at **http://localhost:8830/TrainingSample/static/login.html**. If you recall, we set up a basic user account within the Tomcat WebApp which should give us access to the application. To access the application provide the username and password of "**dev**" and "**bravepoint**", respectively. If the authentication process works, you should be redirected to the "*app.html*" page after login.

This is the landing page, which should include a menu at the top right. If this is not present, or is empty, then the REST request to get menu data may not have run correctly as this is a data-driven menu. As a result of generating our Customer Grid screen, there should be an item under the "*Browse*" header. Mouse-over that item and it should expand to display the "*Customer Read-Only Grid*" that we created in the previous steps. Clicking that item should then load the screen and fetch all available records by default. Note that there is a "*Search by Name*" placeholder text within a field, and that the grid is ordered by "*Name*" by default—all the result of selecting that field as our "*Search Field*" value. If you enter a value such as "*aaa*" into the field and press **Search**, the screen should react by filtering results (via the server) to a single record. Below is the result of searching "wide open" with no criteria.



Drag a column header and drop it here to group by that column

Cust_Num	Country	Name ▲	Address	Address2	City	State	Postal_Code	Contact	Phone	Sales_Rep	Credit_Limit	Balance
2,023	USA	2 Fun Sports Sales	320 30TH ST		Boulder	CO	80303	C Mcmahon	(303) 494-4038	SLS	2,800.00	0.00
2,060	USA	71 Pawn Ctr	1231 W WALNUT ST		Rogers	AR	72756	Irene Konefal	(501) 636-7145	DOS	12,000.00	0.00
1,990	USA	A & A Athletic & Recreation	212 7TH ST SE		Washington	DC	20003	V Talland	(202) 543-0556	GPE	14,800.00	0.00
1,231	USA	A & J Sports Inc	1186 EDDIE DOWLING HWY		North Smithfield	RI	02896	E Powell	(401) 765-4400	DOS	7,400.00	0.00
1,303	USA	A A Collegiate Pennant Co	3411 N MILWAUKEE AVE		Chicago	IL	60641	John Andrade	(773) 286-0600	SLS	49,600.00	0.00
1,874	USA	A P Sports	6124 N MILWAUKEE		Chicago	IL	60646	Sheila Nicholas	(773) 631-6868	HXM	47,100.00	9,317.69

Troubleshooting

At this point, there should ideally have been no issues encountered. The expected files should have been generated, and the user account should be present as it was one we reviewed earlier. Though there is always room for potential error. If you encounter any issues remember to first consult the "**Troubleshooting Checklist**" as found in **Appendix A**. Beyond that, remember that there are multiple pieces at play, working together to bring you this application. If you encounter an issue, attempt to quantify it as either a content issue or a data issue: content issues are static files that should be present and served up by Tomcat, while data issues are requests for a RESTful endpoint and involve Tomcat AND the AppServer. The primary benefit of using the Screen Wizard up to this point for generating our custom application is to know that everything is configured and working correctly up to that point. Any issues beyond the wizard are likely to be isolated to the generated content and/or the new back-end resources, which should help to narrow down the cause of the problem.

Exercise: Create Additional Entities

Now that we have completed an entire process from back to front, let's prepare for some additional examples later. To that end, create 2 new Business Entities for the **Order** and **OrderLine** tables from the **Sports2000** database. These should then be exposed to our REST service and published to Tomcat. Remember to test your new resources via the **Catalog Viewer**!

NOTES: