IBM®

developerWorks.

Feeding iPhone applications with WebSphere Process Server V6.2 services

Skill Level: Intermediate

Chris Felix (chrisfel@ca.ibm.com)
Software Engineer
IBM

27 Jan 2010

This tutorial shows you how to extend new and existing services in WebSphere® Integration Developer V6.2 to use the new JSON over HTTP binding export, so that you can expose services to Web 2.0 and mobile clients that consume JSON. You will create a simple Apple® iPhone® application that will consume such a service. Integration developers will learn how to expose services to an iPhone developer, while iPhone developers can learn how to create Objective-C clients to consume those services.

Section 1. Before you start

The application will demonstrate an example of an Apple iPhone to WebSphere Process Server (hereafter called Process Server) interaction using the JSON over HTTP binding export available in Version 6.2. However, you can apply the same principles to connect another client to Process Server. Imagine that company ABC currently provides an internal service that allows applications to look up the contact information of employees. The service (OrganizationInterface) is hosted on WebSphere Process Server V6.2, and currently only has a Web service (XML over SOAP) export. Since many of ABC's employees are mobile, the company has asked its development team to create an iPhone application to provide this service.

Objectives

Learn how to:

© Copyright IBM Corporation 2010. All rights reserved.

- Extend existing WebSphere Integration Developer V6.2 services to include the new JSON over HTTP binding export.
- Create an Objective-C service proxy for the JSON over HTTP binding export.

Prerequisites

The following tutorial targets two types of development resources:

- A WebSphere Process Server integration developer.
- An iPhone application developer.

Integration developer role

- Familiar with using WebSphere Integration Developer V6.2 to create integration modules.
- Imported the following workspace provided in the Incomplete_iPhoneServicesModule.zip file.

iPhone developer role

- Familiar with Objective-C.
- Installed Apple's iPhone SDK 3.0 or higher.
- Familiar with the iPhone application development.
- Imported the following project into XCode provided in the Incomplete_iPhoneApp.zip file.

System requirements

For the integration developer:

- A PC running Windows® XP
- IBM® WebSphere Integration Developer V6.2

For the iPhone developer:

- An Apple Mac® running OS X Version 10.5.8
- Xcode Version 3.1 for iPhone development

© Copyright IBM Corporation 2010. All rights reserved.

Duration

- For the integration developer tasks: 2 hours
- For the iPhone developer tasks: 1 hour

Download files

This tutorial provides the following files that you can download:

- Incomplete_iPhoneServicesModule.zip: This is the incomplete workspace that the integration developer will work with in the Integration developer tasks section.
- Complete_iPhoneServicesModule.zip: This is the complete workspace if you wish to skip the Integration developer tasks section and just run the finished code.
- Incomplete_iPhoneApp.zip: This contains the incomplete XCode iPhone applicaton that the iPhone developer will work with in the iPhone developer tasks section.
- Complete_iPhoneApp.zip: This contains the completed iPhone application.

Section 2. Integration developer tasks

Since iPhone applications normally transfer data over cellular networks, message payloads need to be as small as possible. For that reason, XML messages are discouraged and JSON over HTTP is encouraged. Using the new JSON over HTTP binding in WebSphere Integration Developer, exposing such a service is straightforward. In this section, you will modify the provided iPhoneServicesModule to provide a JSON over HTTP binding to the OrganizationInterface.

Create the JSON over HTTP binding

1. Create a new workspace in WebSphere Integration Developer. Import the iPhoneServicesModule project interchange file in

© Copyright IBM Corporation 2010. All rights reserved.

iPhoneServicesModule.zip that you downloaded in the Integration developer prerequisites section. Also, when you import in WebSphere Integration Developer, select **Other -> Project Interchange**. See Figure 1.

2. Once imported, open the assembly diagram. The assembly diagram contains the OrganizationInterfaceImpl Java™ component that implements the OrganizationInterface (Figure 2). The Organization Interface contains one method, *employeeSearch*, which takes in a SearchCriteriaBO and outputs a PageOfEmployeeBO containing all employees that match the search criteria. The actual employee lookup is trivial in this tutorial. The code simply outputs two records with the employee name set to the fullName attribute in SearchCriteriaBO. Take some time to familiarize yourself with the module.

Figure 1. Incomplete iPhoneServicesModule

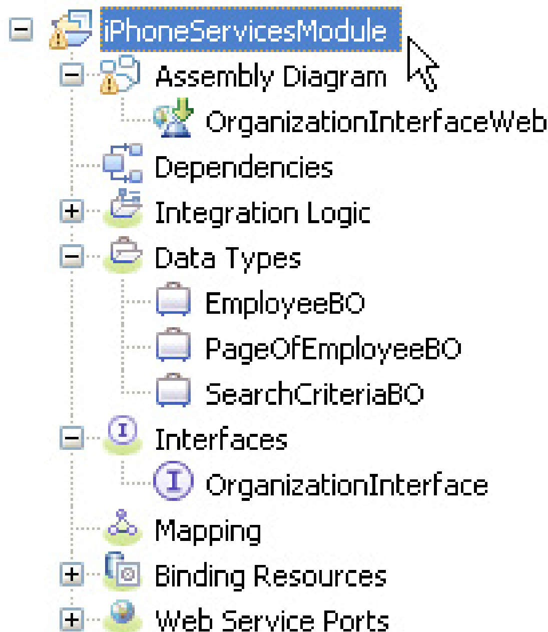
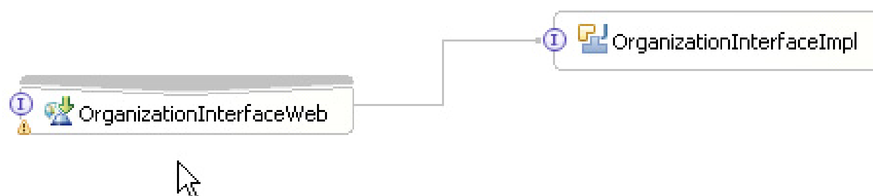


Figure 2. Incomplete assembly diagram



3. Open the assembly diagram.

© Copyright IBM Corporation 2010. All rights reserved.

4. Right-click the **OrganizationInterfaceImpl** Java component and select **Generate Export -> HTTP Binding**.

5. In the dialog that pops up:

   a. Change the Context path to `/OrganizationInterfaceImplMobile`.

   b. Press the **Select** button next to the Default data format.

   c. In the “Data Transformation Configuration” dialog, select the **UTF8JsonDataHandler** inside the JSON category and click **Finish**.

   d. Press the **Select** button next to the Function selector.

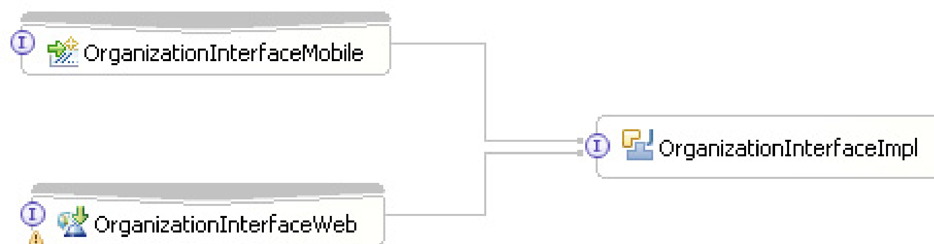
   e. In the “Function Selector Configuration” dialog, select the **HTTP function selector based on URL and HTTP method** and click **Finish**.

   f. Click **OK** in the “Configure HTTP Export Service” dialog.

6. Click the newly generated export and change the name to `OrganizationInterfaceMobile`.

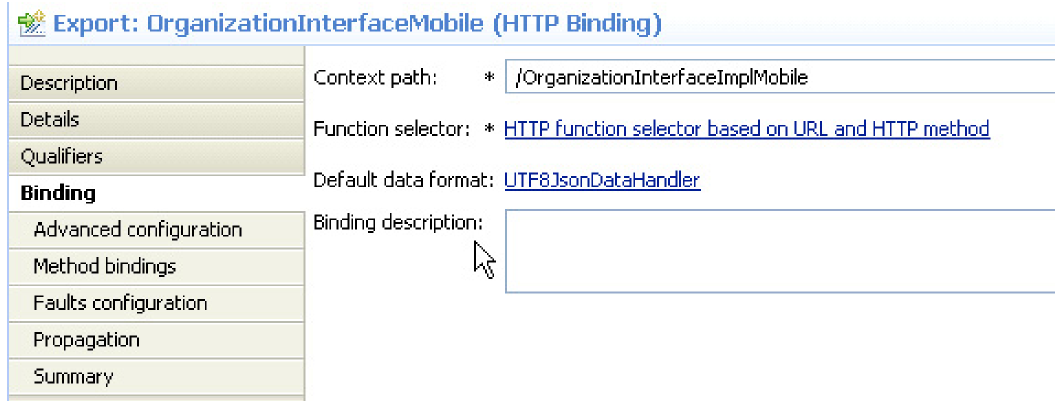
7. Save your changes to the assembly diagram (Figure 3).

8. You can learn more about the various function selectors in Prepackaged HTTP function selectors.
Figure 3. Complete Assembly diagram



9. Click the **OrganizationInterfaceMobile** export and take a moment to look over its properties in the Properties view (Figure 4).
Figure 4. Properties view

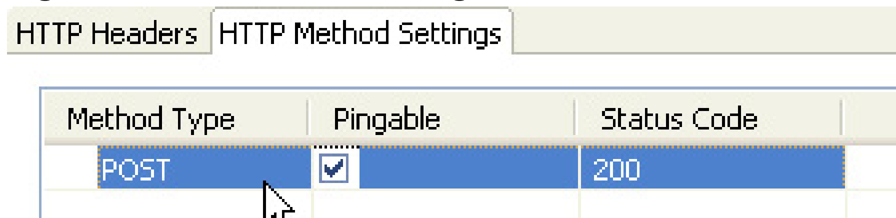
© Copyright IBM Corporation 2010. All rights reserved.



Map service input and output to JSON

In this section, you will make sure that all methods on the service interface transport input and output business objects (BOs) as JSON objects.

1. In the Properties view of OrganizationInterfaceMobile, click **Advanced configuration**.

2. You see a method type of GET already defined in the HTTP Method Settings tab. Remove the GET, and add a POST method instead. Full JSON objects will be sent as input in the body of the POST, instead of sending request parameters in a GET method (Figure 5).
Figure 5. HTTP Method settings


3. Go to the “Method Bindings” section and click the only method on the interface – **employeeSearch**.

4. Notice that the Native method is set to `/OrganizationInterfaceImplMobile/employeeSearch@GET`. Remove that entry.

5. Add the entry `/OrganizationInterfaceImplMobile/employeeSearch@POST`.

6. Click the **Data Serialization** tab and change the Input data format and Output data format to **UTF8JsonDataHandler** by choosing

© Copyright IBM Corporation 2010. All rights reserved.

UTF8JsonHandler inside the JSON category of the “Data Transformation Configuration” dialog. This defines the input and output BO format as JSON. This step is optional since the default data format on the binding is set to UTF8JsonDataHandler.

7. Click the **HTTP Method Settings** tab and remove the GET method type.

8. Save your changes to the assembly diagram.

Test JSON over HTTP Export

In this section, you will test the OrganizationInterfaceMobile export you created.

1. Right-click **OrganizationInterfaceMobile** in the assembly diagram and select **Test component** from the menu.

2. The employeeSearch operation is selected, and you see the searchCriteriaBO input BO in the “Initial request parameters” section.

3. Enter the following values in the searchCriteriaBO, as shown in Figure 6. The values state that you want to find all employees named “John Doe”, and that you want the first page of results (for example, zero indexed). You also do not want more than 10 rows per page, and the results are sorted in ascending order by division (for example, division is the only sortable column in this example):

   a. order = ASC

   b. rowsPerPage = 10

   c. pageIndex = 0

   d. fullName = John Doe

Figure 6. Test input BO

Initial request parameters

Name	Type	Value
searchCriteriaBO	SearchCriteriaBO	✓
division	string	✓
order	string	✓ ASC
rowsPerPage	int	✓ 10
pageIndex	int	✓ 0
fullName	string	✓ John Doe

© Copyright IBM Corporation 2010. All rights reserved.

4. Click **Invoke** in the Events section and click the **Continue** button - the green circle with the white triangle. This will start up WebSphere Process Server V6.2 at the local host test environment server and execute the employeeSearch service method. Please be patient as this may take some time.

5. Notice that the return is a PageOfEmployeeBO containing an array of two EmployeeBOs (Figure 7).
Figure 7. Test output BO

Return parameters:

Name	Type	Value
pageOfEmployeeBO	PageOfEmployeeBO	✓
employees	EmployeeBO[]	
employees[0]	EmployeeBO	✓
firstName	string	✓ John
lastName	string	✓ Doe
division	string	✓ Division ABC
email	string	✓ john doe1@yourcompa

Information to provide to iPhone developer

This section describes the information that an iPhone developer will require to create an Objective-C proxy client to call the employeeSearch service.

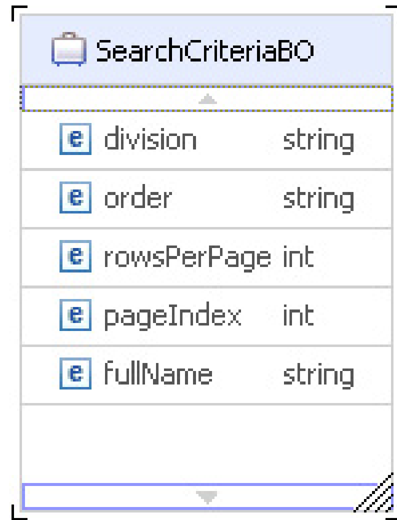
1. Provide the OrganizationInterface.wsdl file. This file will be used to create matching operations, as well as input and output BOs for the client. In this case, a simple image of the interface is enough (Figure 8).
Figure 8. OrganizationInterface

employeeSearch		
Input(s)	searchCriteriaBO	SearchCriteriaBO
Output(s)	pageOfEmployeeBO	PageOfEmployeeBO

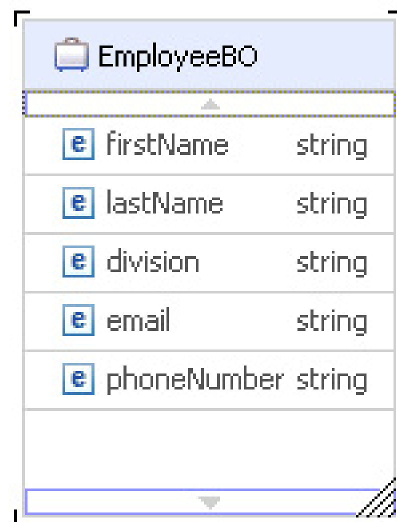
2. Provide details of each BO. In our case, we have three BOs, and images of each will suffice.

    a. SearchCriteriaBO (Figure 9).
    Figure 9. SearchCriteriaBO

© Copyright IBM Corporation 2010. All rights reserved.

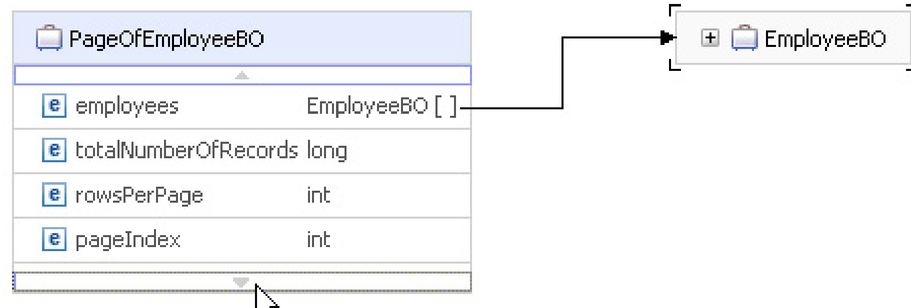


b. EmployeeBO (Figure 10).
Figure 10. EmployeeBO



c. PageOfEmployeeBO (Figure 11).
Figure 11. PageOfEmployeeBO

© Copyright IBM Corporation 2010. All rights reserved.



3. Click the **OrganizationInterfaceMobile** export in the assembly diagram and go to the Properties view.

4. In the Properties view, click the **Summary** section.

5. Provide the table that maps the Method name to the Endpoint (Figure 12).
Figure 12. Service Endpoint URLs

Method Name	Endpoint URL
employeeSearch	iPhoneServicesModuleWeb/OrganizationInterfaceImplMobile/employeeSearch

6. Provide the WebSphere Process Server server address that the iPhone developer can use to call the service. For instance, the developer ends up calling:

```
https://localhost:9445/iPhoneServicesModuleWeb/OrganizationInterfaceImplMobile/
 employeeSearch
```

Your URL may be different if you configured WebSphere Process Server to use a different port.

Section 3. iPhone developer tasks

In this section, you will use the service definition information provided by the integration developer to create an Objective-C service proxy.

Understanding the iPhoneApp

The iPhoneApp XCode project that you downloaded in the Prerequisites section is a simple iPhone application that makes use of a UISearchDisplayController to provide the core user interface. The delegate for UISearchDisplayController is

© Copyright IBM Corporation 2010. All rights reserved.

iPhoneAppViewController.

1. Open the incomplete iPhoneApp XCode project that you downloaded in the Prerequisites section. You will notice that an open source implementation of a JSON parser has been included already. You can download it yourself at http://code.google.com/p/json-framework/ for other projects if you like. Note that the project will not compile at this point.
2. Open **iPhoneAppViewController.m**.
3. Look at the table delegate methods at the start of **iPhoneAppViewController.m**. The UISearchDisplayController contains a table to populate with a list of search results, and iPhoneAppViewController.m contains the delegate methods that populate the table with employee records. The records come from an NSMutableArray called *employees* that contains EmployeeBOs. A service proxy provides the list of employees.
4. After the delegate table methods, you will see another delegate method **searchBarSearchButtonClicked:searchBar**. This is the method that is called when a user submits a search. It uses the service proxy OrganizationInterface to call the employeeSearch service. The service is called asynchronously.
5. Notice below searchBarSearchButtonClicked:searchBar that there are two callback methods. One handles a successful invocation of employeeSearch and the other an unsuccessful invocation. Upon success, the employees array is updated to point to the list of employees returned from employeeSearch. Upon failure, a **UIAlertView** is displayed stating that the search failed.

Add base service

In this section, you will add a base interface and implementation to handle the low level details of sending an asynchronous HTTP request to a service endpoint.

1. Create a new group under the Classes group called **Services**.
2. Right-click the **Services** group and select **Add -> Existing Files**.
3. Navigate to the Classes directory and add **BaseService.h** and **BaseService.m**.
4. Look at BaseService.m, which implements a method called

© Copyright IBM Corporation 2010. All rights reserved.

sendAsyncRequest:

a. Input parameters:

   i. **serviceEndpoint**: A string representing the endpoint URL for the service interface. For example:

```
https://your_local_test_server_address:port/iPhoneServicesModuleWeb/
 OrganizationInterfaceImplMobile/
```

   ii. **toMethod**: A string representing the operation or method to call on the service endpoint (for example, employeeSearch).

   iii. **businessObject**: The business object that the method accepts (for example, a SearchCriteriaBO).

   iv. **delegate**: A reference to a class that implements success:ofMethod and failure:ofMethod methods.

b. The method's purpose is to serialize the businessObject to JSON and POST it to the toMethod on the serviceEndpoint. The included JSON library and the platform's NSMutableURLRequest will handle the mechanics of serializing the BO. If the method call is successful (for example, no communication error, and the return BO does not contain an error code), then the success:ofMethod method is called on the delegate. Otherwise, the failure:ofMethod method is called.

5. The remaining methods in BaseService.m handle the low level HTTP response handling details. In particular:

   a. The method connection:didFailWithError calls failure:ofMethod on the delegate.

   b. The connectionDidFinishLoading:connection de-serializes the response JSON into a dictionary object using the JSON library, and calls success:ofMethod (unless the response JSON contains a service fault, then failure:ofMethod is called).

Add base business object

In this section, you will add a base business object that all business objects will extend.

1. Create a group under Classes called **BusinessObjects**.

© Copyright IBM Corporation 2010. All rights reserved.

2. Right-click **BusinessObjects** and select **Add -> Existing Files**.
3. Add **BaseBO.h** and **BaseBO.m**.
4. Look at BaseBO.m and see that it contains the following:
    a. **Attributes**: A map that will store a JSON object.
    b. **Constructors/initializers**
        i. Default constructor to initialize BaseBO with an empty attributes map, such as init.
        ii. A constructor that accepts a map, and sets attributes to point to that map, such as initWithDictionary.

Create method signature BOs

This section describes the general algorithm for creating Objective-C business objects, given the service definition information provided by the integration developer and discussed earlier.

1. Given the service interface (for example, OrganizationInterface), identify the methods that are bound to HTTP over JSON that you wish to include in a proxy client (for example, employeeSearch).
2. For each method in Step 1, identify the input BO type and the output BO type (for example, SearchCriteriaBO and PageOfEmployeeBO, respectively).
3. For each BO identified in Step 2:
    a. Create an interface of the same name that extends BaseBO (for example, if the BO in the service interface is called EmployeeBO, then you create an interface called `EmployeeBO`).
    b. In the interface file, define getters and setters for each BO attribute.
    c. In the implementation file, define the getters as follows:
        i. For attributes that are simple types in Objective-C, such as NSString, return the value itself. See Listing 1 for examples.
        Listing 1. Getter for simple type

© Copyright IBM Corporation 2010. All rights reserved.

```
- (NSString*) getFirstName{
        return [attributes valueForKey:@"firstName"];
  }
```

ii. For complex types, return a new BO of that type initialized with the map for the given key (Listing 2).

Listing 2. Getter for complex type

```
- (EmployeeBO*) getEmployeeBO{
        return [[EmployeeBO alloc] initWithDictionary:
     [attributes objectForKey:@"employeeBO"]];
 }
```

iii. For an array of simple types, return the array.

iv. For an array of complex types, loop over each object and create a BO of the specified type. Add each BO to an array and return that array.

d. In the implementation file, define the setters as follows:

i. For simple types, you need to translate the value to a String and set the value in the attributes dictionary (Listing 3).

Listing 3. Setter for simple type

```
- (void) setFirstName: (NSString*)firstName{
        [attributes setValue:firstName forKey:@"firstName"];
}
```

ii. For complex types, set the value of the key to the attributes of the input BO (Listing 4).

Listing 4. Setter for complex type

```
- (void) setEmployeeBO: (EmployeeBO*) employeeBO{
  [attributes setValue: [employeeBO attributes]
forKey:@"employeeBO"];
}
```

iii. For an array of complex types, loop over each object and add the attributes of each into a new array. Once done, set the value of the key to the built up array.

Example

The OrganizationInterface contains the method employeeSearch that takes a SearchCriteriaBO as input and returns a PageOfEmployeeBO as output. Looking at

© Copyright IBM Corporation 2010. All rights reserved.

PageOfEmployeeBO, you will see that it contains another complex type called EmployeeBO. Thus, we have three BOs to create. To speed things up, these BOs have already been created. You will just add them to the existing project.

1. Right-click the **BusinessObjects** group in iPhoneApp and select **Add -> Existing Files**.
2. Add **EmployeeBO.h, EmployeeBO.m, PageOfEmployeeBO.h, PageOfEmployeeBO.m, SearchCriteriaBO.h**, and **SearchCriteriaBO.m**.
3. Open any BO and see how the getters and setters are defined as described above. Take some time to look over all of the BOs.

Create a service proxy

The final step in client creation is to create the service proxy. The proxy will essentially be an Objective-C class that implements the service interface. This section describes the general algorithm for creating a proxy.

1. Create an interface of the same name as the service interface file in WebSphere Integration Developer, such as OrganizationInterface. The interface must extend BaseService.h.
2. In the interface, define the method signatures for each method you identified in the Create Method Signature BOs section above (Listing 5).
   Listing 5. EmployeeSearch Method Signature

```
- (void) employeeSearch: (SearchCriteriaBO*)serachCriteriaBO delegate:
 (id)del;
```

   Note that the return type of each method is void. The delegate is a pointer to a class that implements the success and failure callback methods. In our case, it's iPhoneAppViewController (Listing 6).
   Listing 6. EmployeeSearch Callback success

```
- (void) employeeSearch_CallbackSuccess: (PageOfEmployeeBO*)
 pageOfEmployeeBO{
  ...
}
```

   Listing 7. EmployeeSearch Callback failure

```
- (void) employeeSearch_CallbackFailure: (NSError*) error{
  ...
```


© Copyright IBM Corporation 2010. All rights reserved.

```
}
```

3. Add the following method definitions, as shown in Listing 8.
Listing 8. Success and failure of method definitions

```
- (void) success:(NSDictionary*)returnBO ofMethod:(NSString*)mName;
- (void) failure:(NSError*)error ofMethod:(NSString*)mName;
```

4. Create an implementation class for the interface:

    a. Each method simply calls BaseService's sendAsyncRequest with the proper parameters (Listing 9).
    Listing 9. Sample call to sendAsyncRequest

```
- (void) employeeSearch:(SearchCriteriaBO*)searchCriteriaBO delegate:(
    id)del{
        callback = del;
[super sendAsyncRequest:SERVICE_ENDPOINT toMethod:EMPLOYEE_SEARCH
 businessObject:searchCriteriaBO delegate:self];

}
```

a. Define the success:ofMethod method to handle calling the correct CallbackSuccess method on the delegate.

b. Define the failure:ofMethod method to handle calling the correct CallbackFailure method on the delegate.

Example

The OrganizationInterface contains one method employeeSearch that you must provide a proxy for. Such a proxy class has already been created and you will just add it to the iPhoneApp.

1. Right-click the **Services** group and select **Add -> Existing Files**.

2. Select to add **OrganizationInterface.h** and **OrganizationInterface.m**.

3. Open **OrganizationInterface.m** and notice the implementation of employeeSearch. The method makes use of a constant SERVICE_ENDPOINT. You will need to change the value of the constant to match the WebSphere Process Server instance that is running the OrganizationInterface service. For example, in our development environment, we set the SERVICE_ENDPOINT to be

© Copyright IBM Corporation 2010. All rights reserved.

```
https://chrisfel:9445/iPhoneServicesModuleWeb/OrganizationInterfa
```

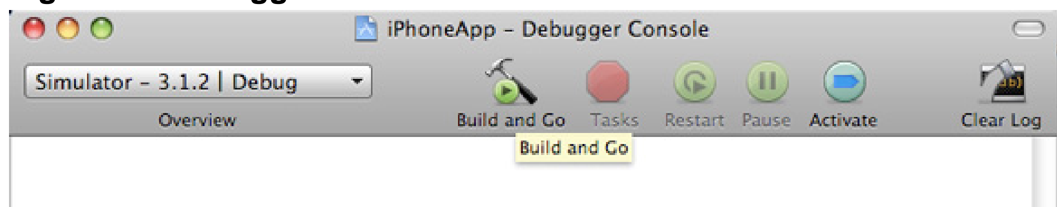
4. Build the iPhoneApp and you should see no errors.

5. Take some time to browse through the OrganizationInterface.m code, especially the success:ofMethod and failure:ofMethod methods, to understand how the **emloyeeSearch_CallbackSuccess** and **employeeSearch_CallbackFailure** methods in iPhoneAppViewController get called.

Testing the iPhoneApp

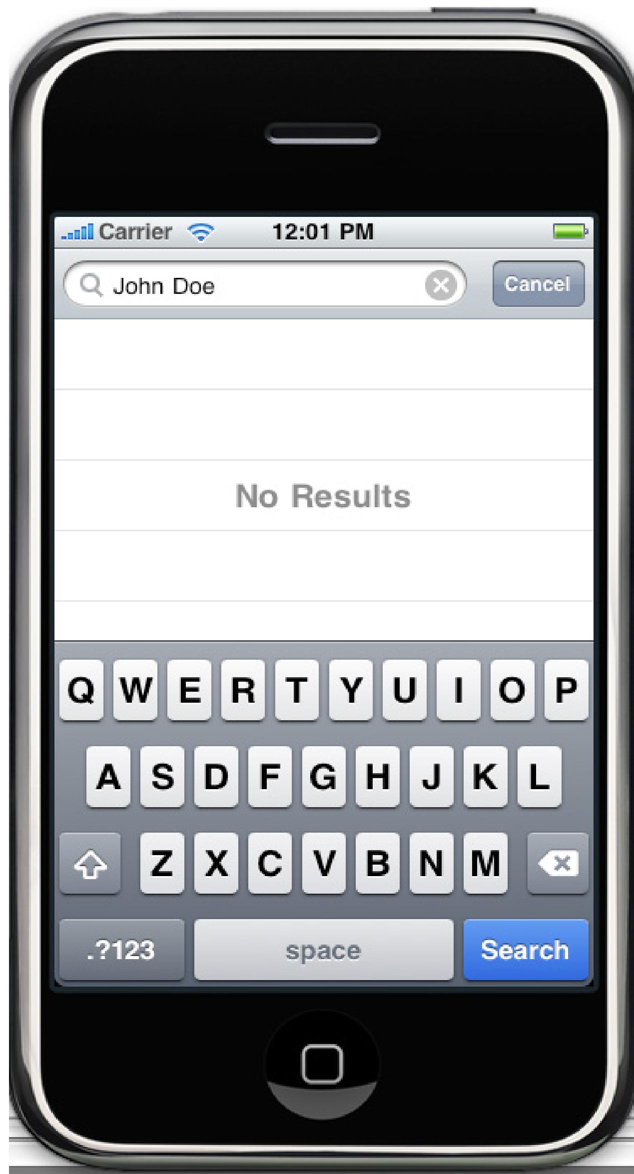
In this section, you will run the iPhoneApp on the iPhone simulator. You will see how searching on a full name, such as `John Doe`, calls the backend service asynchronously and returns results to the iPhoneApp.

1. Open the Debugger Console in XCode and click **Build and Go** (Figure 13).
Figure 13. Debugger Console



2. Type `John Doe` in the search bar (Figure 14).
Figure 14. Search on John Doe

© Copyright IBM Corporation 2010. All rights reserved.



3. Make sure that the WebSphere Process Server local test environment in WebSphere Integration Developer is running and synchronized (Figure 15).

Figure 15. Server ready

Server	State	Status
WebSphere Process Server v6.2	Started	Synchronized

4. Clear the console output in Integration Developer.

5. Go back to the iPhone simulator, and click **Search**.

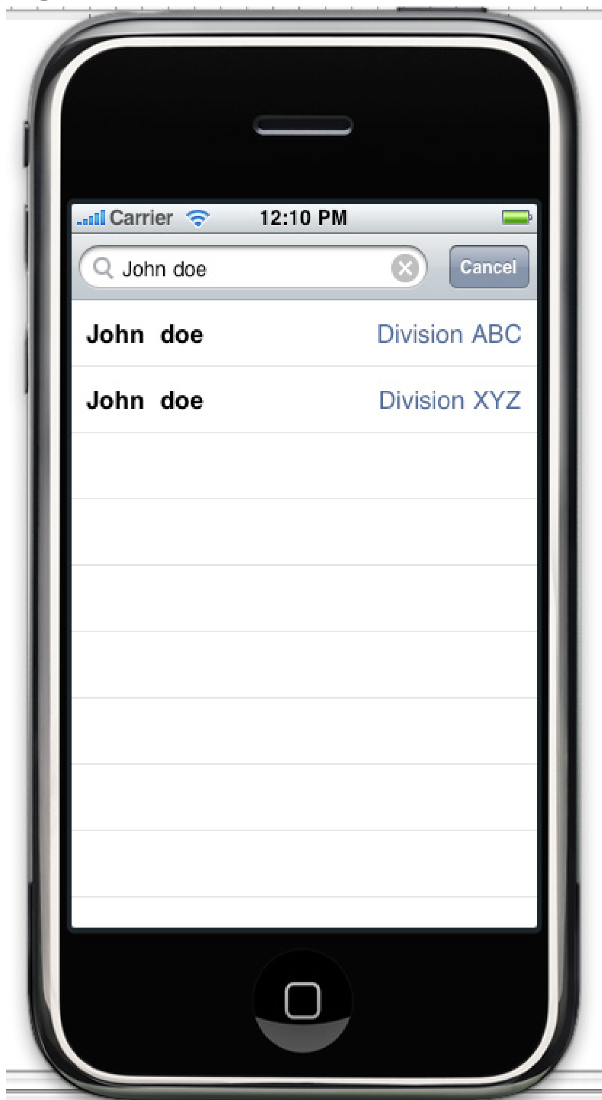
© Copyright IBM Corporation 2010. All rights reserved.

6. Notice the logging activity in the Process Server console (Figure 16).

Figure 16. WebSphere Process Server console output

7. You now see a list of employees matching the full name you searched on in the iPhone simulator (Figure 17).

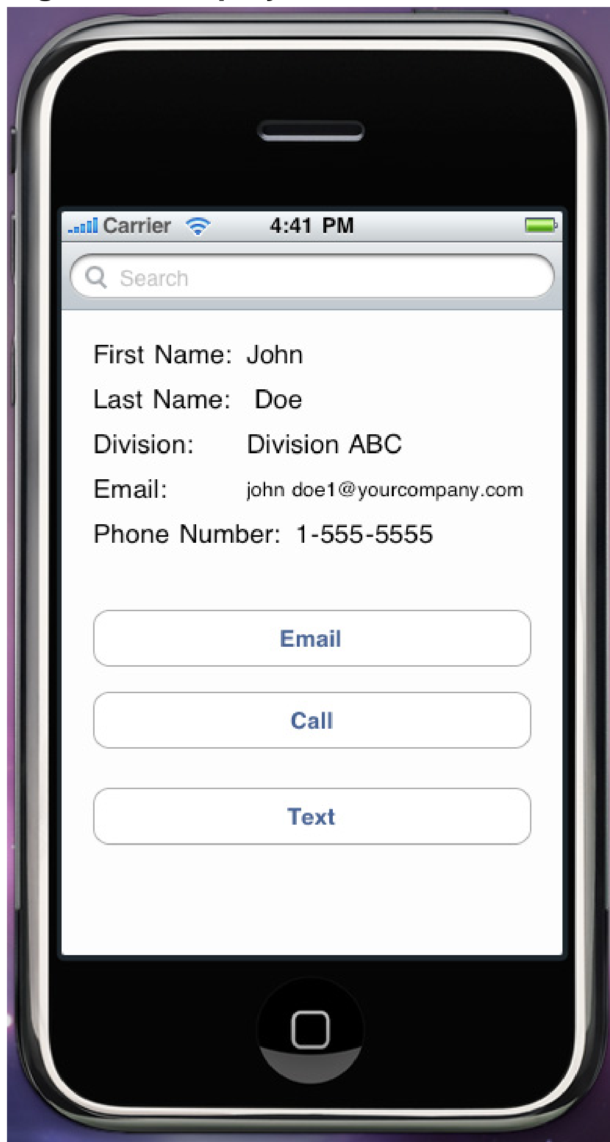
Figure 17. Search results



8. Click one of the names and notice that the EmployeeBO details are populated in the user interface (Figure 18).

© Copyright IBM Corporation 2010. All rights reserved.

Figure 18. Employee record detail



Section 4. Conclusion

This tutorial demonstrated how to extend an existing WebSphere Process Server V6.2 service to include a JSON over HTTP export bnding. The new export made it possible for the service to be consumed by JSON-enabled client proxies. In our case, the proxy was written in Objective-C and targeted for an iPhone application. The tutorial also discussed an algorithm used to take a WebSphere Integration

© Copyright IBM Corporation 2010. All rights reserved.

Developer V6.2 service definition and to create an HTTP over JSON proxy in Objective-C. With the knowledge gained here, you can easily integrate your organization's existing WebSphere Process Server V6.2 services with your organization's mobile applications.

© Copyright IBM Corporation 2010. All rights reserved.

Downloads

Description	Name	Size	Download method
Code sample	Incomplete_iPhoneServicesModule.zip	13KB	HTTP
Code sample	Complete_iPhoneServicesModule.zip	14KB	HTTP
Code sample	Incomplete_iPhoneApp.zip	993KB	HTTP
Code sample	Complete_iPhoneApp.zip	976KB	HTTP

Information about download methods

© Copyright IBM Corporation 2010. All rights reserved.

Resources

Learn

- Cocoa Dev Central: Learn Objective-C
- Json-framework
- Mastering Ajax, Part 10: Using JSON for data transfer
- iPhone Dev Center: Apple Developer Connection
- WebSphere Process Server and WeSphere Integration Developer resource page

Discuss

- WebSphere Integration Developer discussion forum
- WebSphere Process Server discussion forum

About the author

Chris Felix

Chris Felix is a Software Developer and Agile ScrumMaster in the IBM Software Services for WebSphere organization at the IBM Toronto Lab, Canada. He is experienced in developing Web 2.0 applications that leverage Web services provided through WebSphere Process Server. He was recently engaged in a project that required a mobile front end, allowing him to gain experience in iPhone development.

© Copyright IBM Corporation 2010. All rights reserved.