



IBM Software Group

**WebSphere Process Server V6.2**  
**WebSphere Enterprise Service Bus V6.2**  
**WebSphere Integration Developer V6.2**  
*Service Component Architecture (SCA) tools  
and examples*



@business on demand.

© 2009 IBM Corporation  
Updated June 26, 2009

This presentation addresses Service Component Architecture, SCA, introducing the tools and providing some examples.



## Agenda

- Tools support
- Example
- Summary



This section provides an overview of the tools support for building SCA applications.

## SCA module assembly

- The assembly editor
  - ▶ Primary tool for composing SCA-based applications
  - ▶ Visual tool for specifying
    - Components (Interfaces, implementation type, and references)
    - Stand-alone references
    - Exports
    - Imports
    - Wires connecting component references with the appropriate target service
  - ▶ Several development approaches are supported
    - Top down
    - Bottom up
    - Meet in the middle



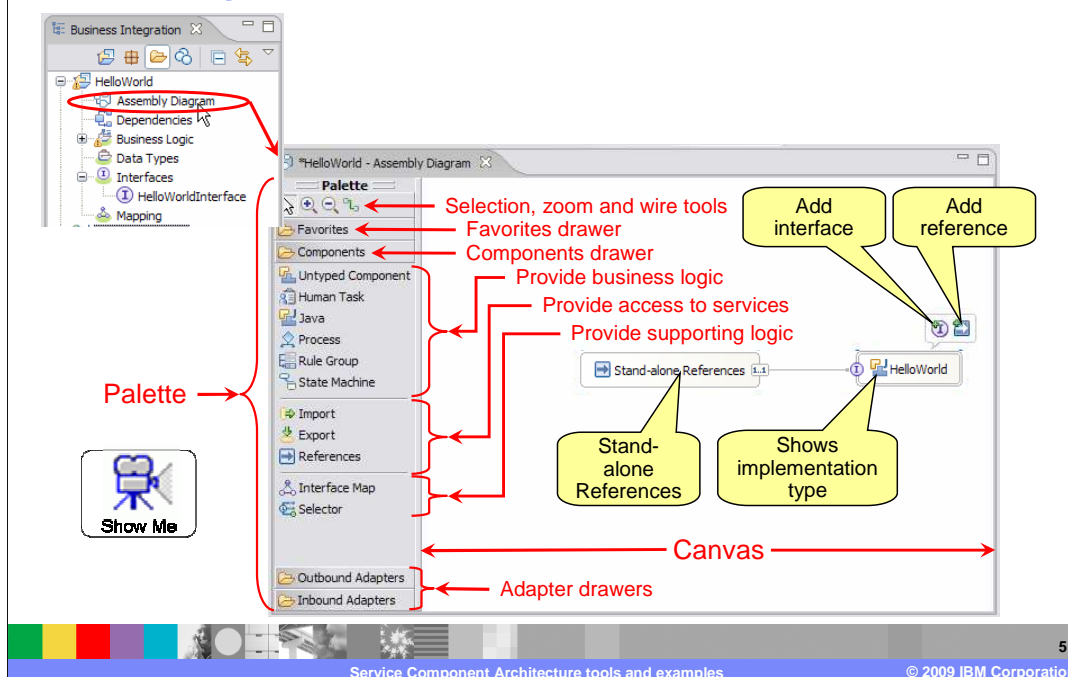
WebSphere Integration Developer provides first class tools support for building SCA based applications targeted for WebSphere Process Server and WebSphere Enterprise Service Bus. The primary tool for defining and assembling SCA artifacts is the assembly editor. This editor allows developers to visually build SCA elements such as components, exports, imports, and stand-alone references and to wire them together to build composite applications. For each element that is visually created using the assembly editor, WebSphere Integration Developer takes care of generating the appropriate Service Component Definition Language, SCDL, artifacts.

When using the assembly editor there are several development approaches. First, you can build your SCA application using a top down development model. In this case you can use the assembly editor to diagram and model your application before you create any backing business logic. Once components are added to the assembly diagram in the editor, you can assign interfaces and even create new interface definitions for each component from within the editor. Once the SCA components are defined with their interfaces and references, you generate skeleton implementations and edit them to add your business logic.

In the bottom up development model, you start by defining your business logic by implementing BPEL processes, business state machines, business rules, and human tasks. Then, you create the SCA components for these implementations by dragging them and dropping them onto the assembly diagram. As part of this process, the appropriate interfaces and references are automatically added to the components. You complete the assembly by wiring the SCA elements together.

Finally, in the meet in the middle development model, you define the elements in the assembly diagram and their implementations in parallel. As with the top down approach, you create the assembly diagram in the assembly editor. But unlike the top down approach, you do not generate implementations from the elements in the diagram. Instead, you select the appropriate implementation that you created in parallel. Be aware that the parallel activities, creating the diagram and creating their implementations, are not done in isolation. You must know the interfaces and references for each element in order to create its implementation, and vice versa. The point is that you do not have to wait for one activity to finish before you begin with the other.

## Assembly editor basics



Each module project in WebSphere Integration Developer has one assembly diagram associated with the SCA project. The assembly diagram for a module is found in the Business Integration view directly under the project folder. To open the assembly diagram in the assembly editor, double click on the assembly diagram icon in the business integration view.

On the left of the assembly editor is a palette that allows you to add various SCA artifacts to the assembly diagram. The palette groups elements into drawers that can then be expanded and collapsed. The primary drawer is labeled Components and is expanded in this screen capture. Within the expanded view there are three groupings. The first grouping is for those SCA components that provide business logic, which are human tasks, Java™, business processes, business rule groups and business state machines. The second group is for those SCA artifacts that provide access to services, specifically imports, exports and stand-alone references. The third grouping is for components that provide supporting logic, which are interface maps and selectors. The two collapsed drawers at the bottom of the palette are for outbound and inbound adapters which provide access to imports and exports associated with specific adapter types. Finally, notice the collapsed drawer at the top of the palette, labeled Favorites. This drawer enables you to create your own grouping of the components and adapters you regularly use, allowing you to work from that drawer without having to open the others. At the top of the palette is a toolbar containing a selection tool, zoom out and zoom in tools and the wire tool.

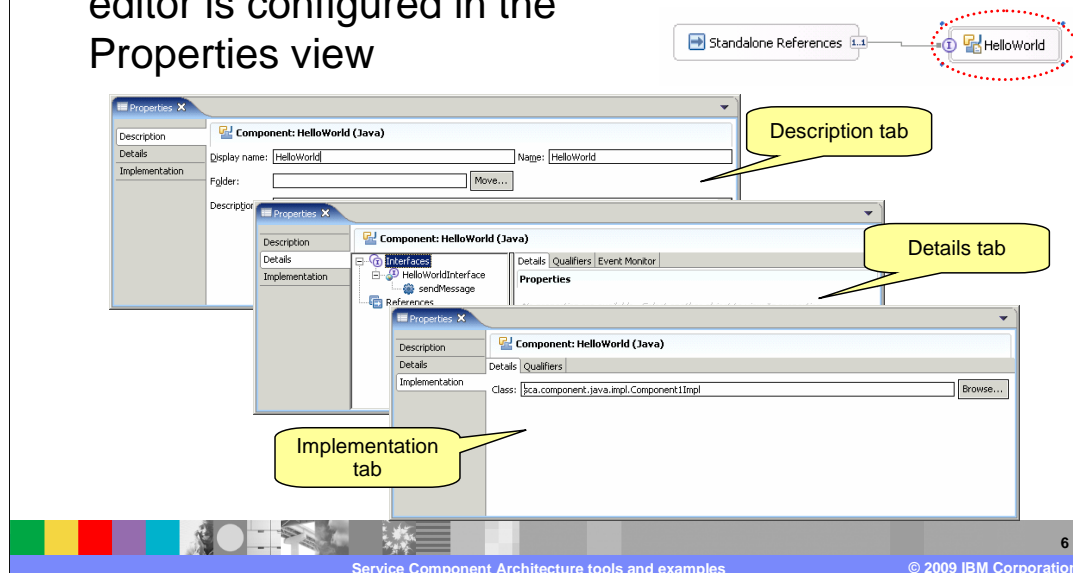
The canvas area of the assembly diagram shows the various components that make up the SCA application in the module project. Much of your assembly work can be done from within the assembly diagram using the palette and pop-up menus. In this screen capture, on the right, is a Java component called HelloWorld. The icons above it pop up to allow you to add an interface or a reference to the component. The icon on the component itself shows the implementation type of the component and these icons match the icons used in the palette. You can also see the stand-alone references, which is wired to the HelloWorld component, making that component available to non-SCA clients.

In addition to the diagram on the canvas, the properties view is an important part of building your assembly diagram. The properties view, which is not shown on this slide, is where specific configuration information is specified for the artifacts on the canvas.

For a more detailed explanation of the assembly editor click the Show Me icon to see a demonstration.

## Properties view

- Selected element in assembly editor is configured in the Properties view



Like most editors, the assembly editor is tightly integrated with the properties view. When a particular component is selected in the assembly editor, the corresponding properties for that component are available for viewing and editing in the Properties view. For SCA components there are three tabs that are displayed in the properties view.

The description tab is used to view and edit general properties about the component.

The details tab is used to view and edit interfaces and references associated with the component. Note that there is also the ability from this tab to set qualifiers for interface and reference scopes. The implementation tab is used to view and edit configuration associated with the particular implementation type for the selected component. Implementation level qualifiers are also set from this tab.

The properties view of imports and exports are conceptually similar to that of components. The description and details tabs are essentially the same but the implementation tab is replaced with a binding tab. Depending upon the type of binding associated with the import or export, there can be additional tabs in the properties view.




## Adding elements to the assembly

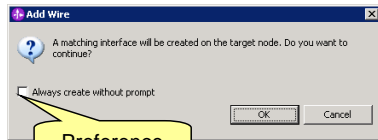
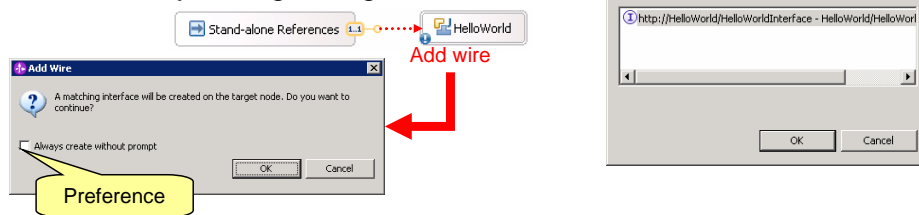
Type	Add Options
Component	<ul style="list-style-type: none"> <li>▪ Add component from palette or pop-up menu of assembly diagram</li> <li>▪ Drop an interface from module or library project onto assembly diagram and select "<b>Component with no implementation type</b>"</li> <li>▪ Drop an implementation onto assembly diagram</li> </ul>
Import	<ul style="list-style-type: none"> <li>▪ Add import from palette or pop-up menu of assembly diagram</li> <li>▪ Drop an export from another module onto assembly diagram</li> <li>▪ Drop an interface from module or library project onto assembly diagram and select "<b>Import with...</b>"</li> </ul>
Export	<ul style="list-style-type: none"> <li>▪ Add export from palette or pop-up menu of assembly diagram</li> <li>▪ Select a component in assembly diagram and choose "Export..." from the pop-up menu</li> <li>▪ Drop an interface from module or library project onto assembly diagram and select "<b>Export with...</b>"</li> </ul>



The table on this slide lists some the various ways you can add components, imports, and exports to the assembly diagram. In all cases, these elements can be added to the assembly diagram using the assembly editor palette or dropping the appropriate interface onto the assembly diagram and selecting the appropriate component type from a dialog box. Typically the method the you will use for adding components to the assembly diagram will depend on whether you are using a top down or bottom up development approach.

## Adding interfaces

- Interfaces can be added
  - ▶ Before wiring an existing component
    - Action bar 
    - Properties view 
    - Context menu 
  - ▶ When a new component is created
    - Dropping an interface on the assembly
    - Dropping an implementation on the assembly
  - ▶ Automatically during wiring




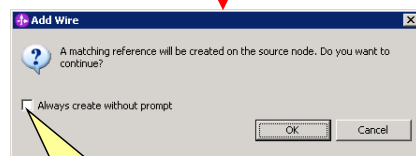
This slide looks at the most common ways that an interface definition is added to a component.

For an existing component that has already been added to the assembly diagram, you might need to add one or more interfaces to the component definition. Interfaces can be added to the component definition explicitly, using the appropriate action bar icon, or from the properties view, or from the pop-up menu. However, interfaces can also be added to a component automatically when you take a particular action. For example, if an interface or implementation is dropped onto the assembly diagram from the business integration view, a new component is created which is already configured with the appropriate interface definition. Likewise, an interface definition can automatically be added to a component when an attempt is made to wire a reference with a particular interface to a component that does not include that interface definition. In this case, WebSphere Integration Developer prompts you to find out if you want to have an interface automatically created on the target component.

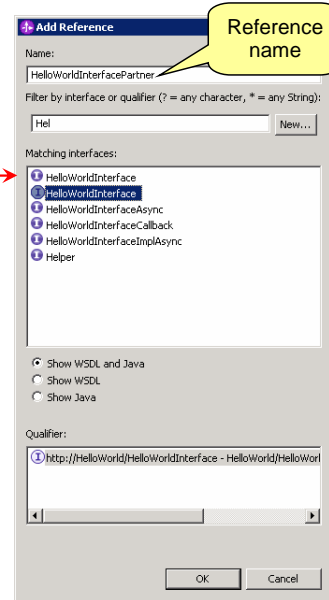


## Adding references

- References can be added
  - ▶ Before wiring to a target component
    - Action bar 
    - Properties view
    - Context menu
  - ▶ Automatically during wiring



Preference



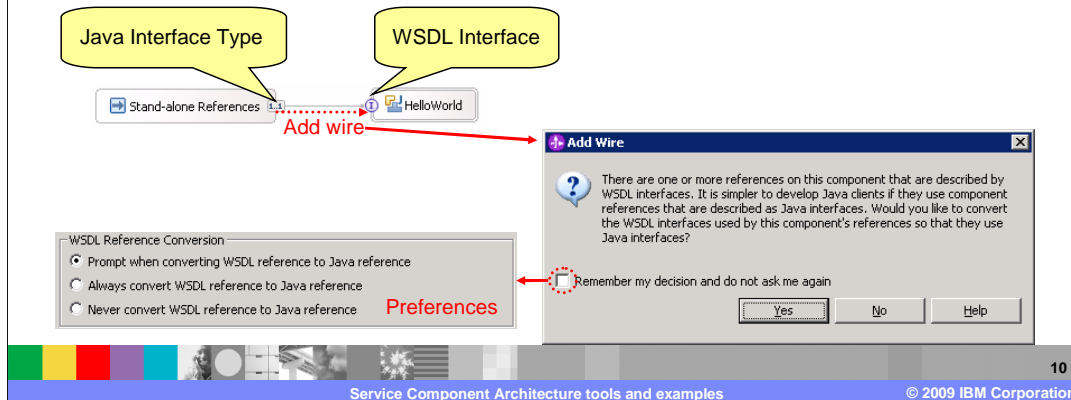
Reference name



Like interfaces associated with a component, references on a component definition can be added in several different ways. References can be added explicitly from the action bar that appears above a component in the assembly diagram. Likewise, you can also use the properties view or pop-up menu for the component to add a reference. Another approach is to allow the assembly editor tools to add the reference automatically when wiring together two components if the caller does not already have a reference to the target component. In this case, the assembly editor will prompt you, indicating that a reference is going to be added to the source node. This prompt can be disabled by selecting the assembly editor preference to always create a reference without prompting.


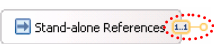
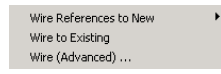
## Java to WSDL references

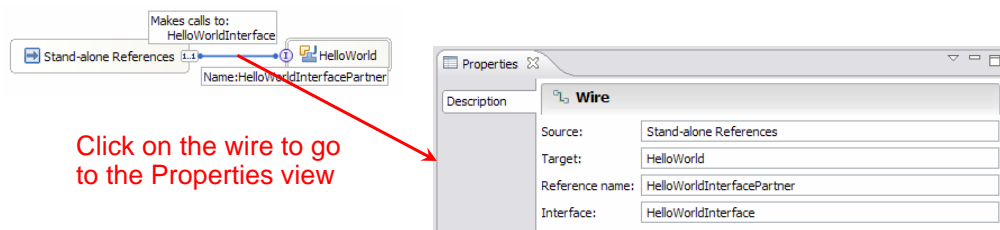
- Java-to-WSDL references
  - ▶ Allows stand-alone references or Java components to access components with WSDL interfaces
    - Client can now use type safe invocation rather than dynamic



The assembly editor also provides a feature that converts WSDL interface references to a Java interface on a reference definition. This allows stand-alone references or Java components to access target components with a WSDL interface using a strongly typed Java interface rather than the dynamic invocation approach. When wiring together a stand-alone reference or a Java component with a interface that has a WSDL port type interface, the tools will prompt you about whether to convert the reference to a Java interface. Selecting 'Yes' causes the tools to generate a Java interface based upon the target WSDL interface. By selecting the 'Remember my decision...' check box, you can automatically set a preference to have the tools always convert the WSDL reference to a Java reference or to never perform this conversion. This preference can also be directly set using the preferences dialog.

## Wires

- Components can be wired together by the
  - ▶ Wiring tool from palette 
  - ▶ Wiring handle  
  - ▶ Context menu option 
- The properties view displays important information about a wire

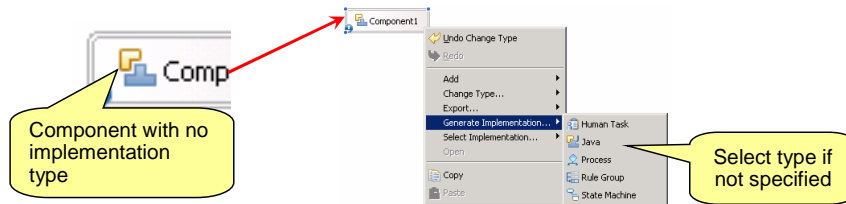


There are several ways components can be wired together. First a wire can be created between two components by using the wiring tool from the palette. There is also a yellow wiring handle that appears when you hover the mouse pointer over side of a component or reference. This wiring handle can be selected and pulled over to the target component to create a wire. Finally, there are several pop-up menu options available for creating wires between components.

By selecting a wire and going to the properties view you can view some important information about the wire definition. It defines the source and target for the wire along with the interface associated with the wire. It provides the reference name which is needed to locate a service using the ServiceManager with the client programming model. You can also determine the reference name by explicitly selecting the reference in the assembly diagram and viewing the properties view or by using the help provided when you hover the mouse pointer over the reference.

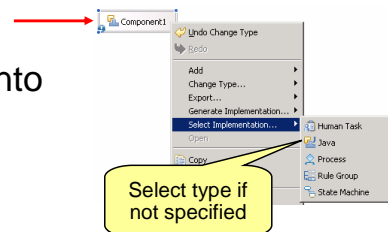
## Specifying the implementation

### ■ Generating a new implementation



### ■ Select an existing implementation

- ▶ Context menu for component
- ▶ Drop existing implementation onto assembly diagram



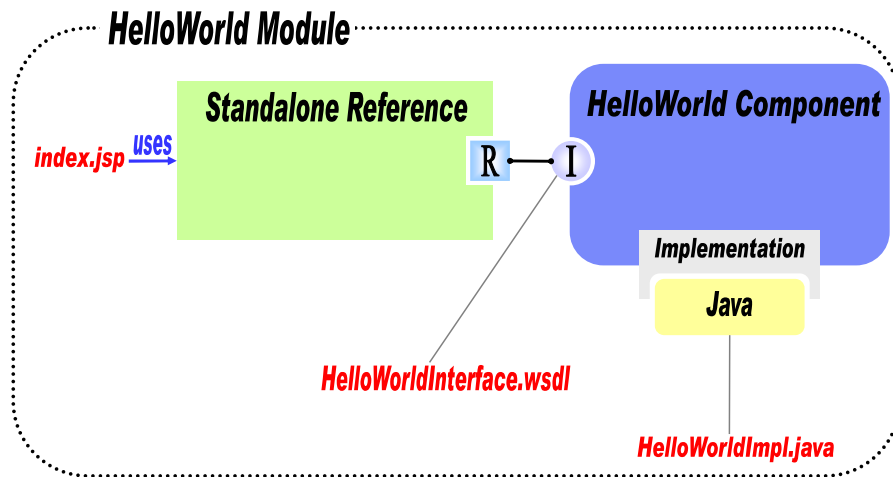
Components in an SCA module will all ultimately have an implementation type associated with them. There are several ways to indicate this implementation type for a component. The first way to do this is when creating the component by selecting a component of a particular type, such as Java or Human Task, from the palette and then dropping it onto the canvas. Then use the pop-up menu for the component and select Generate Implementation. You might also have a component on the assembly diagram that has no implementation type associated with it, in which case the Generate Implementation... choice on the pop-up menu cascades to a list of possible types. The upper portion of the slide shows this particular situation.

The equivalent mechanism works when selecting an existing implementation for an component by using the pop-up menu choice Select Implementation. If the component already has a type associated with it you are given a choice of the existing implementations of that type. However, if you have a generic component on the assembly diagram that has no implementation type, the pop-up menu will cascade allowing you to select the type of implementation you want.

If you do not already have the component on the assembly diagram, you can drop an existing implementation directly on the canvas and a component with that implementation is created.

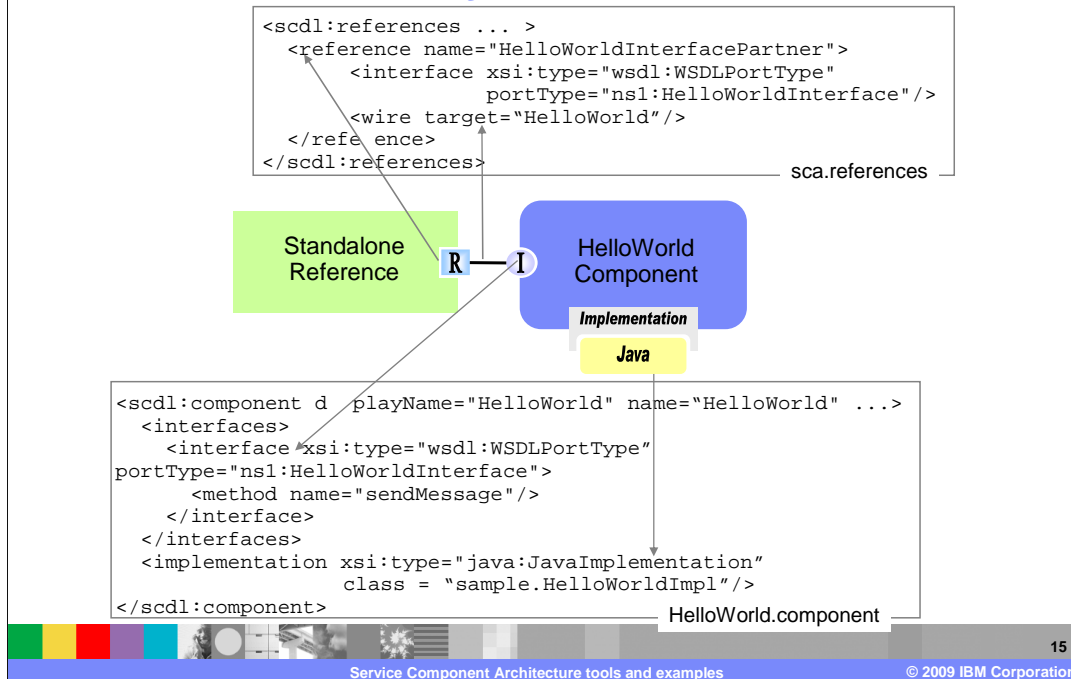


## HelloWorld example overview



The diagram on this slide provides an overview of the example used to highlight the SCA programming model in the next several slides. In this example, there is a HelloWorld module that contains a simple component that has a WSDL port type interface and a Java implementation. This component does not have any references to other components. Also included in the module is a stand-alone reference that is used by a client JSP to invoke the simple HelloWorld component.

## HelloWorld assembly definitions



As a brief introduction to the Service Component Definition Language, SCDL, this slide provides an overview of the HelloWorld component definition, in addition to the stand-alone reference definition. Note that for the HelloWorld component the interface is a WSDL port type, and that the implementation type is Java. Notice the reference name in the stand-alone reference definition. This name is needed later in this example to pass to the ServiceManager to look up the HelloWorld service. Also note in the stand-alone reference that the wire definition has a target that points to the HelloWorld component.

## HelloWorld implementation details

The screenshot shows the 'HelloWorldInterface.wSDL' editor. The 'Define Operation(s)' section is active, displaying a table with the following data:

	Name	Type
sendMessage		
Input(s)	message	string
Output(s)	status	string

A red arrow labeled 'Implementation' points from the 'sendMessage' operation to the following Java code snippet:

```
public class HelloWorldImpl {  
    :  
    // Methods common to all Java SCA component implementations removed  
    :  
    public String sendMessage(String message) {  
        return "The following message was submitted: " + message;  
    }  
}
```

The code is saved in 'HelloWorldImpl.java'.

In the example presented so far, the HelloWorldInterface associated with the HelloWorld component is defined using a WSDL port type interface. The top of the slide shows the definition of the HelloWorldInterface in the interface editor. The bottom of the slide shows portions of the Java class used to implement the component. It shows the sendMessage method that implements the operation defined on the interface.



## HelloWorld client implementation

```
try {
    ServiceManager serviceManager = new ServiceManager();
    Service service = (Service)
        serviceManager.locateService("HelloWorldInterfacePartner");

    String theMessage = request.getParameter("message");
    DataObject resp = (DataObject) service.invoke("sendMessage", theMessage);
    if (resp != null) {
        out.println("<p>" + resp.getString("status") + "</p>");
    }
} catch (Exception e) {
    System.out.println(e);
}
```

Reference  
Name

Get status String from  
returned DataObject and  
add it to print stream

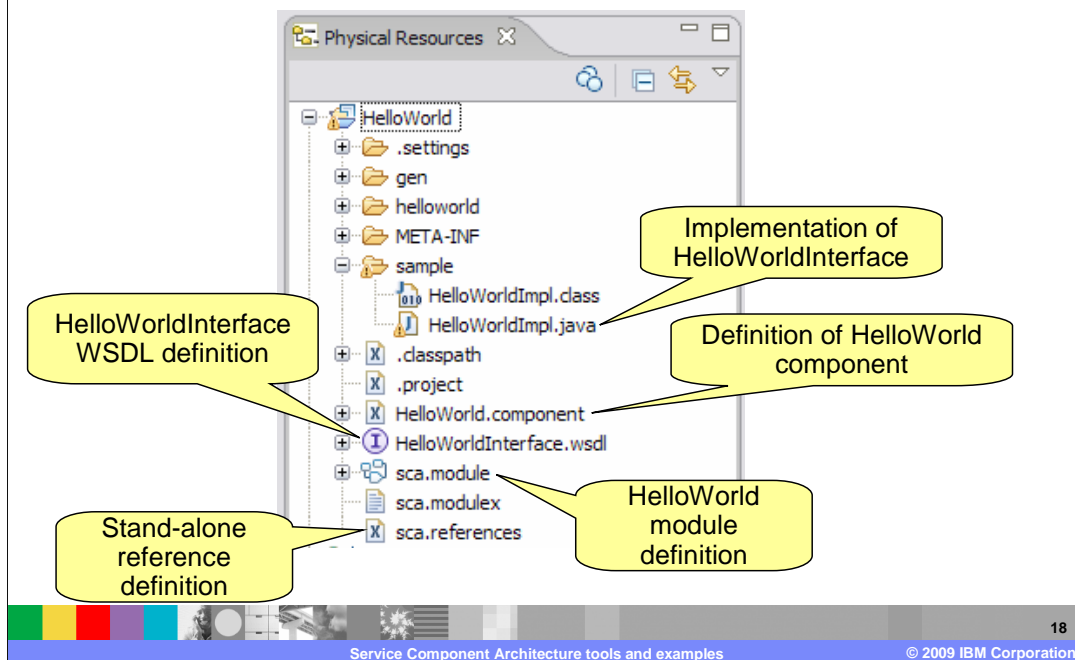
Invoke the  
"sendMessage"  
operation

index.jsp (Client Programming Model)

This slide highlights the client code needed to invoke the HelloWorld component from a client JSP. The first step is to use the ServiceManager to locate the HelloWorld service. This is done with the locateService method, passing in the reference name for the stand-alone reference that is wired to the HelloWorld component.

The next step is to call the invoke method to invoke the sendMessage operation on the HelloWorld interface. Since the interface to the component is a WSDL port type, the dynamic invocation rather than the type safe APIs are used. Finally the response from invoking the service is displayed.

## HelloWorld artifacts



The Physical Resources view in WebSphere Integration Developer is a useful view for looking at all of the resources that make up an SCA module. These resources can be particularly useful when debugging an application or just learning more about the definition language used to define SCA components. Note that there are a set of filters that apply, by default, to the physical resources view and therefore not all of these artifacts will show when you first switch to this view. This screen capture was taken after all of the filtering was turned off.



## Summary

- Introduction to SCA module assembly
  - ▶ Basics of assembly editor
  - ▶ Properties view
  - ▶ Adding components, interfaces and references
  - ▶ Wiring
- Simple SCA example
  - ▶ Single component with stand-alone reference
  - ▶ Example SCDL artifacts
  - ▶ Java component implementation
  - ▶ Client code



You were presented with an introduction to the assembly editor, the primary tool for developing SCA applications in WebSphere Integration Developer. The basics of the assembly editor were introduced along with its relationship to the properties view. Performing simple tasks, such as adding components, defining their interfaces, references, implementations and wiring was also reviewed. Following that an example was provided, showing some of the SCDL artifacts that are generated. The simple Java implementation and client code needed to call it were also shown.

## Feedback

### Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

[mailto:iea@us.ibm.com?subject=Feedback\\_about\\_WBPMv62\\_SCA\\_SCAToolsAndExamples.ppt](mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_SCA_SCAToolsAndExamples.ppt)

This module is also available in PDF format at: [../WBPMv62\\_SCA\\_SCAToolsAndExamples.pdf](http://www.ibm.com/developer/education/assistant/..WBPMv62_SCA_SCAToolsAndExamples.pdf)



You can help improve the quality of IBM Education Assistant content by providing feedback.

## Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both: WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>

Java, JSP, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.