



IBM Software Group

WebSphere® Process Server V6.0.2
WebSphere Enterprise Service Bus V6.0.2
WebSphere Integration Developer V6.0.2
*Service Component Architecture (SCA) tools
and examples*



@business on demand.

© 2007 IBM Corporation
Updated April 4, 2007

This presentation will cover tools and provide examples for Service Component Architecture (SCA).

Goals

- Introduce tools support for assembling SCA modules
- Provide a simple example of an SCA-based application



The goals of this presentation are to introduce tools support for Service Component Architecture provided in WebSphere Integration Developer version 6 and provide a simple example of an SCA-based application.

Agenda

- **Tools support**
- Example
- Summary and references



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

Module assembly in Integration Developer V6

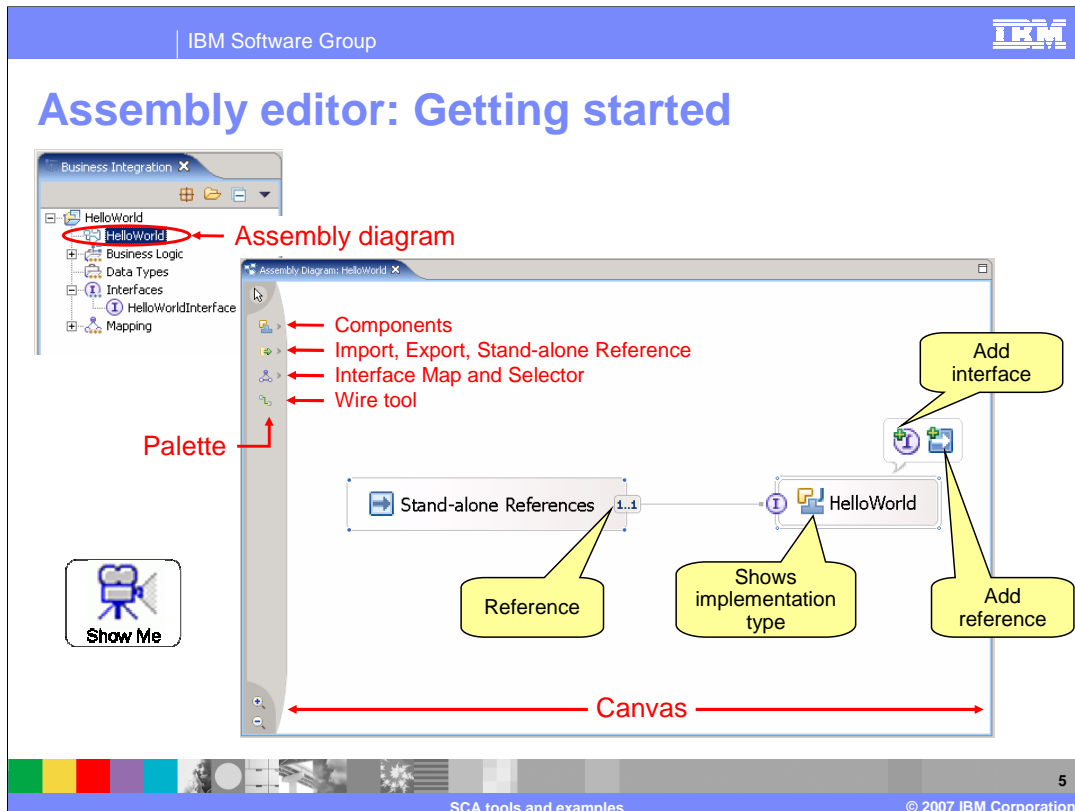


WebSphere Integration Developer provides first class tools support for building SCA based applications targeted for WebSphere Process Server. The primary tool for defining and assembling SCA artifacts is the Assembly Editor. This editor allows developers to visually build SCA elements such as service components, exports, imports, and stand-alone references in addition to wire together components to build a composite application. For each element that is visually created using the assembly editor, Integration Developer tools take care of generating the appropriate Service Component Definition Language (SCDL) for each component behind the scenes.

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 there is any backing business logic created. 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 from them, and edit the implementations 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 artifacts 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 artifacts. 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. Know 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.

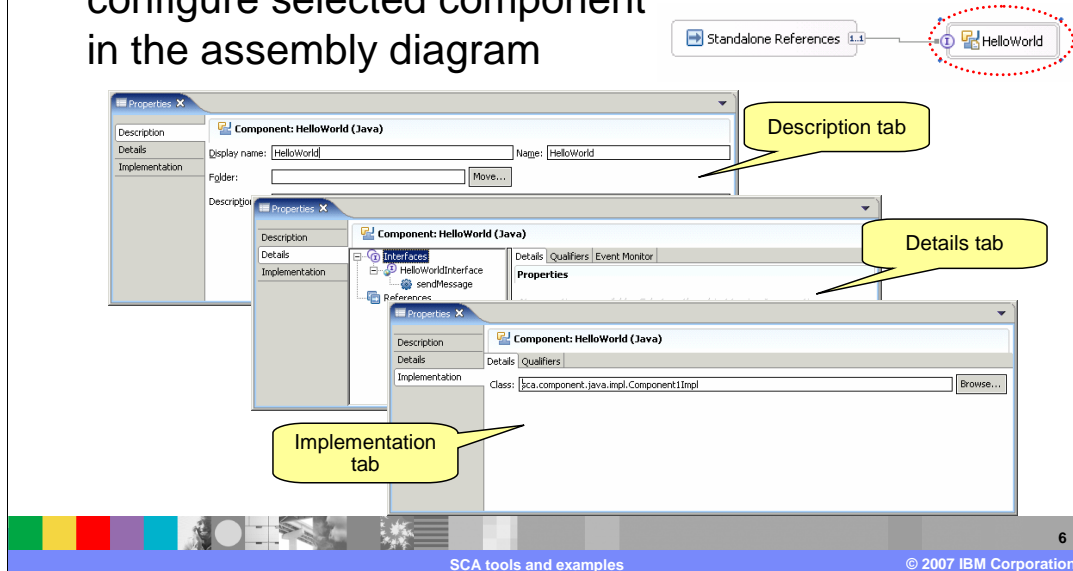


Each module project in WebSphere Integration Developer has one assembly diagram associated with the SCA project. The assembly diagram for a module is found in the Business Integration view directly under the project folder, and is given the same name as the module project. To open the assembly diagram 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 and wire them together to build a composite application. 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 right within the assembly diagram using the palette, action bar, and context menus. In addition to this, the Properties view is also an important part of building your assembly diagram. The Properties view is tightly associated with the assembly diagram, and enables typical and advanced service component configuration activities.

Assembly diagram: Properties

- Use the properties view to configure selected component in the assembly diagram



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 service components there are three tabs that are displayed in the properties view:

Description tab – Is used to view and edit general properties about the service component.

Details tab – Is used to view and edit interfaces and references associated with the service component. Note that there is also the ability from this tab to set qualifiers for interface and reference scopes.

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 import and export components are similar to service components, with the exception of the Implementation tab. For import and export elements you do not have an implementation tab, rather this tab is replaced with a Bindings tab that allows you to configure information that is specific to the type of binding for that element.


Assembly diagram: Adding elements

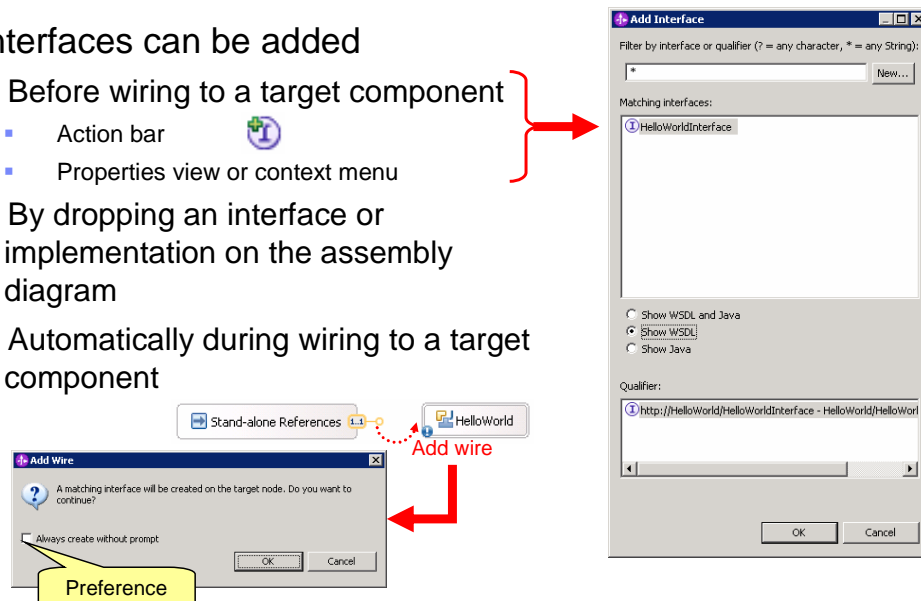
Type	Add Options
Service Component	<ul style="list-style-type: none"> Add component from palette or context menu of assembly diagram Drop an interface from module or library project onto assembly diagram and select "Component with no implementation type" Drop an implementation onto assembly diagram
Import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

7

The table on this slide lists the various ways developers can add components, imports, and exports to the assembly diagram. In all cases, these elements can be added to the assembly diagram using the palette from the assembly diagram editor or by 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 he/she is using a top-down or bottom-up development approach.


Assembly diagram: Interfaces

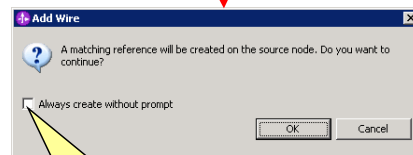
- Interfaces can be added
 - ▶ Before wiring to a target component
 - Action bar 
 - Properties view or context menu
 - ▶ By dropping an interface or implementation on the assembly diagram
 - ▶ Automatically during wiring to a target component



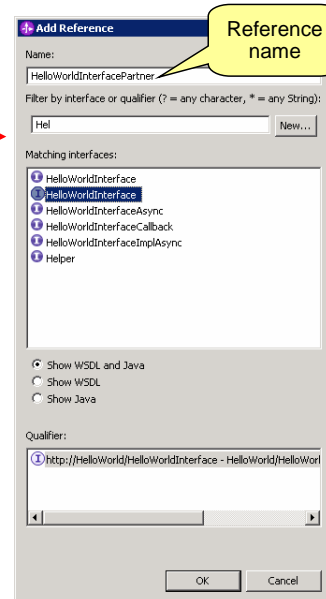
Once a component has been added to the assembly diagram, you may need to add one or more interfaces to the component definition. Interfaces may be added to the component definition explicitly using the appropriate action bar icon, from within the properties view, or from the context menu. However, interfaces can also be added to a service component automatically when you take a particular action. For example, if an interface or implementation is dropped onto the assembly diagram from the physical resources or business integration view, that interface will be added to the service component defined by this action. Likewise, an interface definition can automatically be added to a service component definition 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 they want to have an interface automatically created on the target service component.

Assembly diagram: References

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



Preference

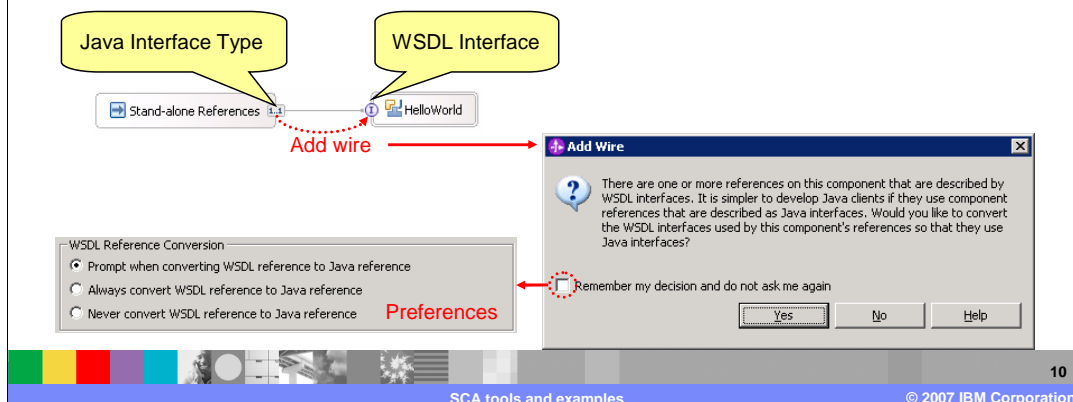


Reference name

Like interfaces associated with a component, references on a service component definition can be added in several different ways. References can be added explicitly from the action bar that appears above a service component in the assembly diagram. Likewise, you can also use the properties view or context menu for a service component to add a reference. Another approach is to allow the assembly editor tools to add the reference automatically when wiring together two components when 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 will be added to the source node. This prompt can be disabled by selecting the assembly editor preference to always create a reference without prompting.



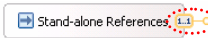
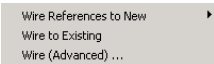
Assembly diagram: References (cont.)

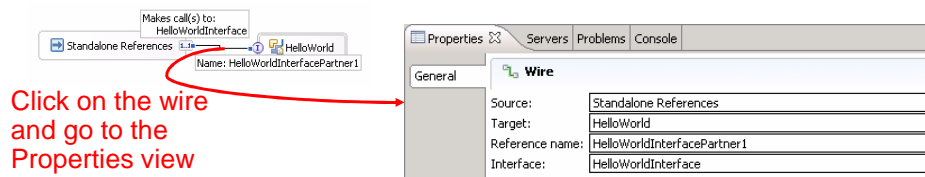
- Java-to-WSDL references
 - ▶ Allows stand-alone references or Java™ service components to access components with WSDL interfaces
 - Client can now use type safe invocation rather than dynamic



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 service components to access target components with a WSDL interfaces using a strongly typed Java interface rather than the dynamic invocation approach. When wiring together a stand-alone reference or a Java service component with a interface that has a WSDL port type interface, the tools will prompt you about whether or not 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 Java reference or to never perform this conversion.

Assembly diagram: Wires

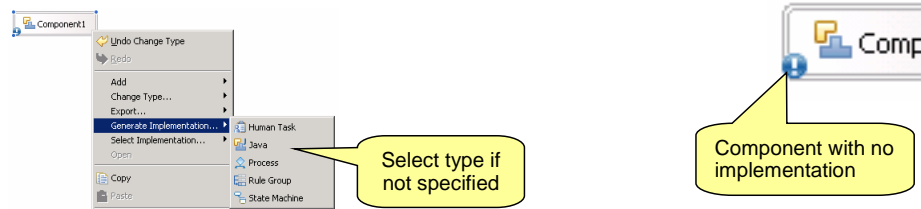
- Components can be wired together by the
 - ▶ Wiring tool from palette 
 - ▶ Wiring handle  
 - ▶ Context menu option 
- The properties view can display important information about a wire



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 an item is selected, on the side of a service component or reference. This wiring handle can be selected and pulled over to the target component to create a wire. Finally, there are several context 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. Specifically, you can determine the reference name which is needed to locate a service from the ServiceManager with the client programming model. You can also view the reference name by explicitly selecting the reference in the assembly diagram and viewing the properties view or the hover help.

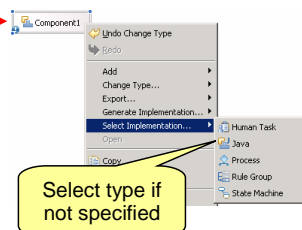
Assembly diagram: Selecting implementation

■ Creating a new implementation



■ Select an existing implementation

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



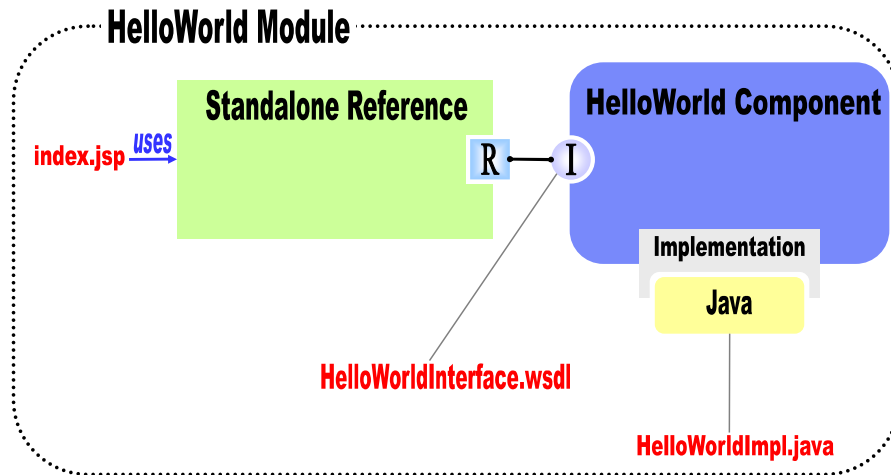
Service components in an SCA module will all ultimately have an implementation type associated with them. There are several ways to indicate this implementation type for a component. The first way to do this is by dropping a component of a particular type, such as Java, Process, or Human Task from the palette and onto the canvas of the assembly editor. However, if you have a generic component with no implementation type on the assembly diagram, you can use the context menu options to either generate a new implementation and select the type, or by selecting an existing implementation. You can select an existing implementation by using the context menu option or by dropping an existing implementation, from the Physical Resources view for example, onto the assembly diagram.

Agenda

- Tools support
- **Example**
- Summary and references

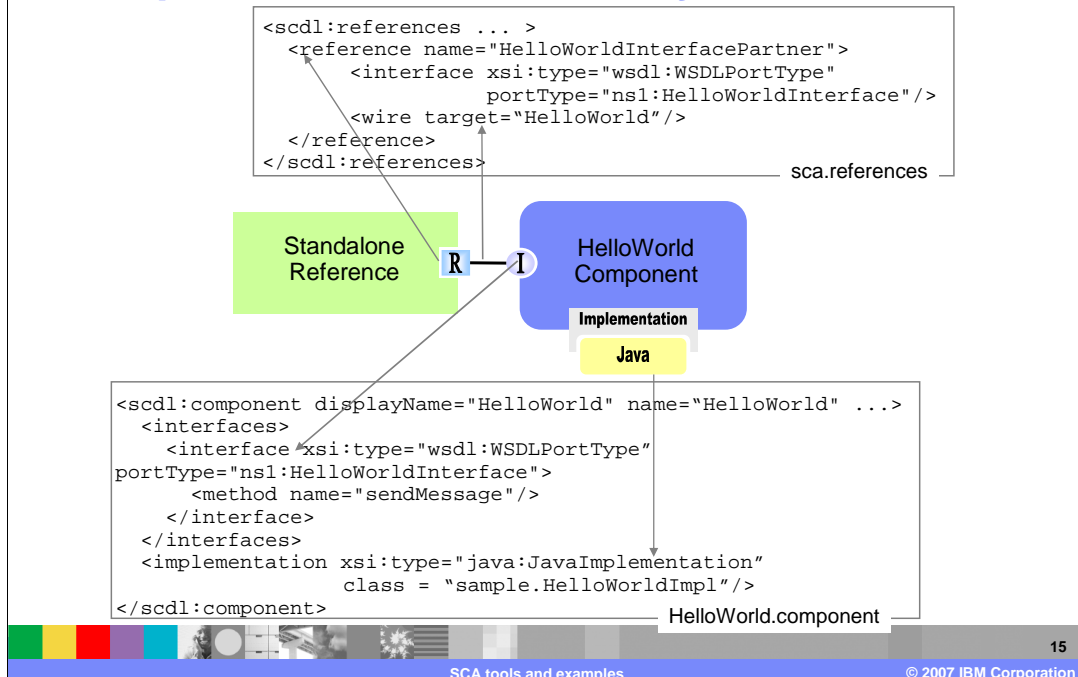
This section will provide several examples that highlight the SCA client programming model.

Example: HelloWorld



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 service component that has a WSDL port type interface and a Java implementation. This service component does not have any references to other service components. Also included in the module is a stand-alone reference that is used by a client JSP to invoke the simple HelloWorld Service component.

Example: 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 type is WSDL port type, and that the implementation type is Java. Notice the reference name in the stand-alone reference definition because 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 the wire definition that has a target that points to the HelloWorld component.

Example: HelloWorld implementation details

The screenshot shows a software development environment window titled "HelloWorldInterface" with a sub-tab "HelloWorldInterface.wSDL". The main area displays the "Define Operation(s)" section, which includes a table defining the "sendMessage" operation. A red arrow labeled "Implementation" points from the "sendMessage" operation in the table to a code block below.

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

```
public class HelloWorldImpl {  
    public String sendMessage(String message) {  
        return "The following message was submitted: " + message;  
    }  
}
```

Below the code block, the filename "HelloWorldImpl.java" is visible.

In the example presented so far, the HelloWorldInterface associated with the HelloWorld component is defined using a WSDL port type interface. That interface definition is highlighted on this slide, along with the Java class used to implement the service component.

Example: 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

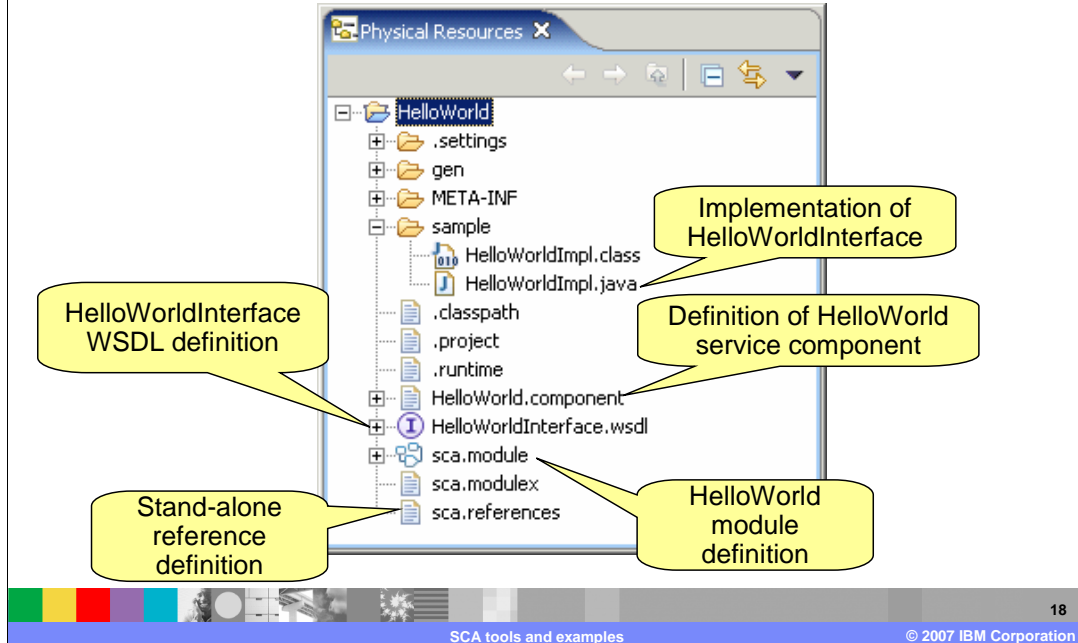
Invoke the
"sendMessage"
service

index.jsp (Client Programming Model)

17

This slide highlights the client code needed to invoke the HelloWorld service component from within a client JSP. The first step is to use the ServiceManager to locate the HelloWorld service. This is done with the locateService method and passing in the reference name for the stand-alone reference that is wired to the HelloWorld service component. The next step is to call the invoke method to invoke the sendMessage operation on the HelloWorld interface. Finally the response from invoking the service can be displayed to you.

Example: 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.

Agenda

- Tools support
- Example
- **Summary and references**

This section will provide a summary of SCA.

Summary

- SCA is the fundamental component model for WebSphere Process Server V6
 - ▶ Programming model the Service Oriented Architecture solution
- SCA helps separate business logic from implementation
 - ▶ Focus is on assembling solutions rather than implementation details
 - ▶ Mitigates need for integration developers to have deep knowledge of Java or J2EE
 - ▶ Aimed at helping J2EE developers become more productive

SCA is the fundamental component model for WebSphere Process Server V6 and provides the basis of the service oriented architecture solution. SCA helps separate business logic from implementation and allows developers to focus on assembling solutions rather than implementation details.

References

- Service data objects (SDO)

- ▶ <http://www-128.ibm.com/developerworks/library/j-commonj-sdowmt/>
- ▶ <http://www-106.ibm.com/developerworks/java/library/j-sdo/>

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](#)



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

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

J2EE, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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.

Information is provided "AS IS" without warranty of any kind. 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 NON-INFRINGEMENT. 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 2007. 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.