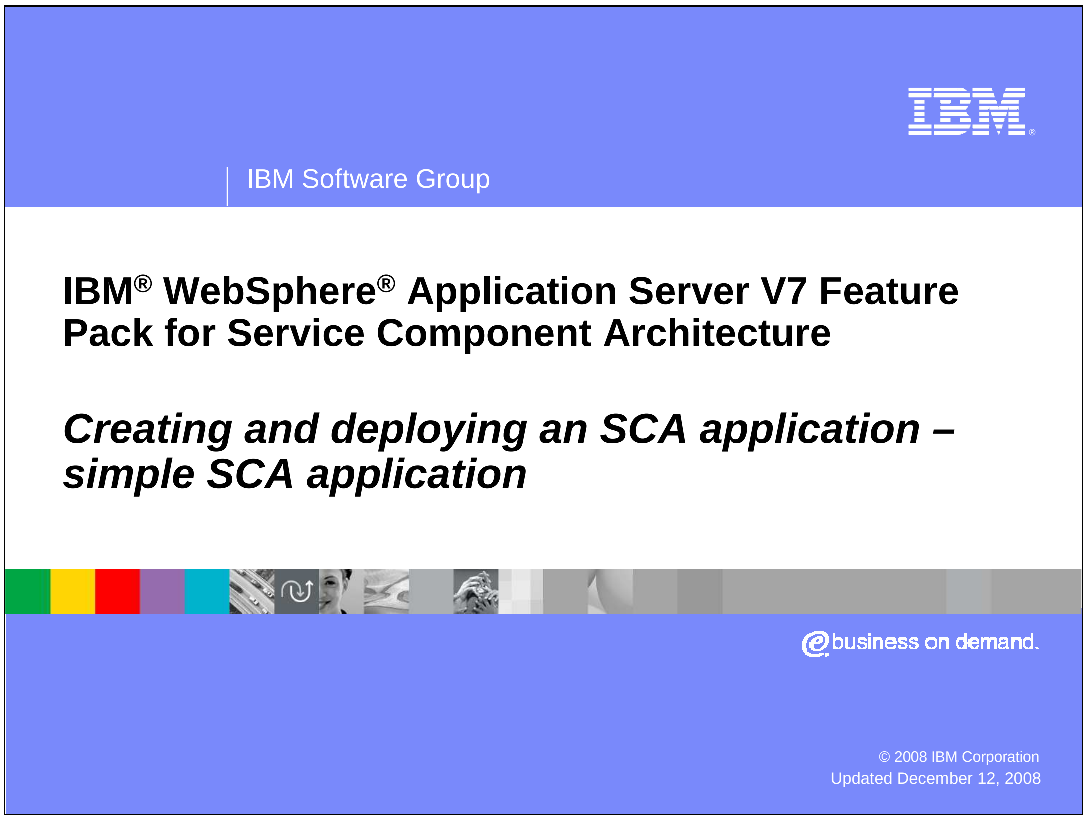


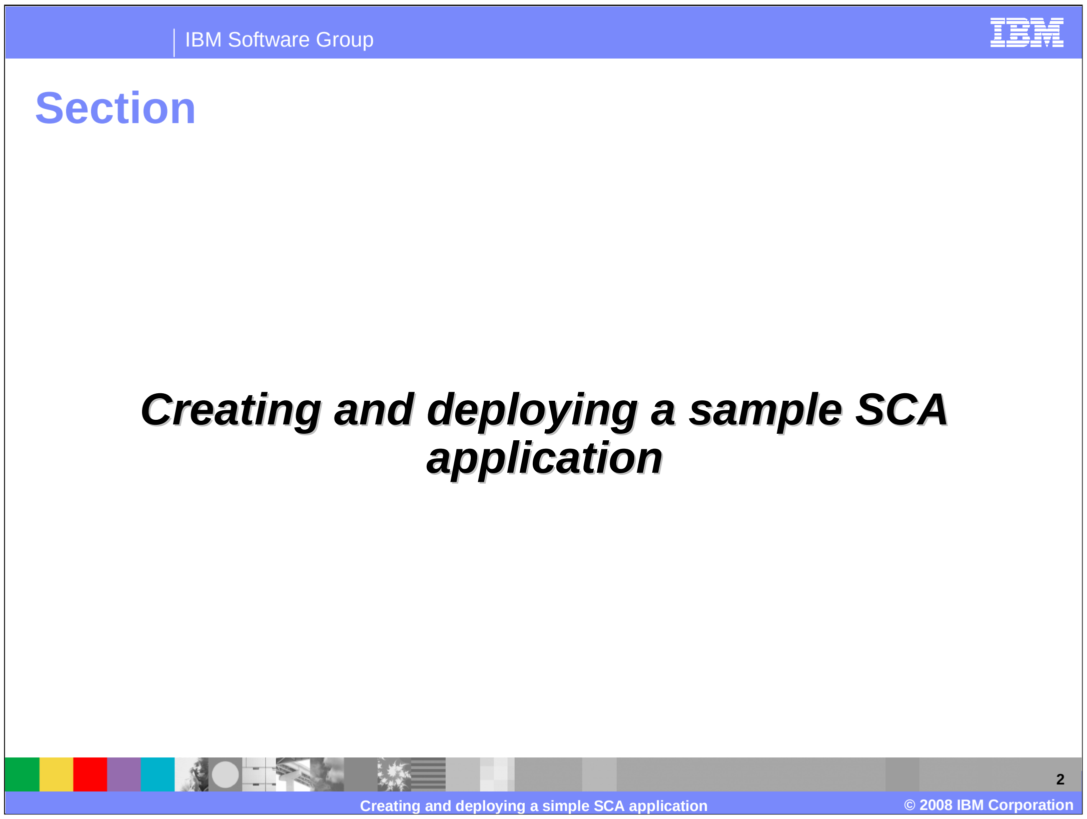
IBM Software Group

IBM® WebSphere® Application Server V7 Feature Pack for Service Component Architecture

Creating and deploying an SCA application – simple SCA application

@business on demand.

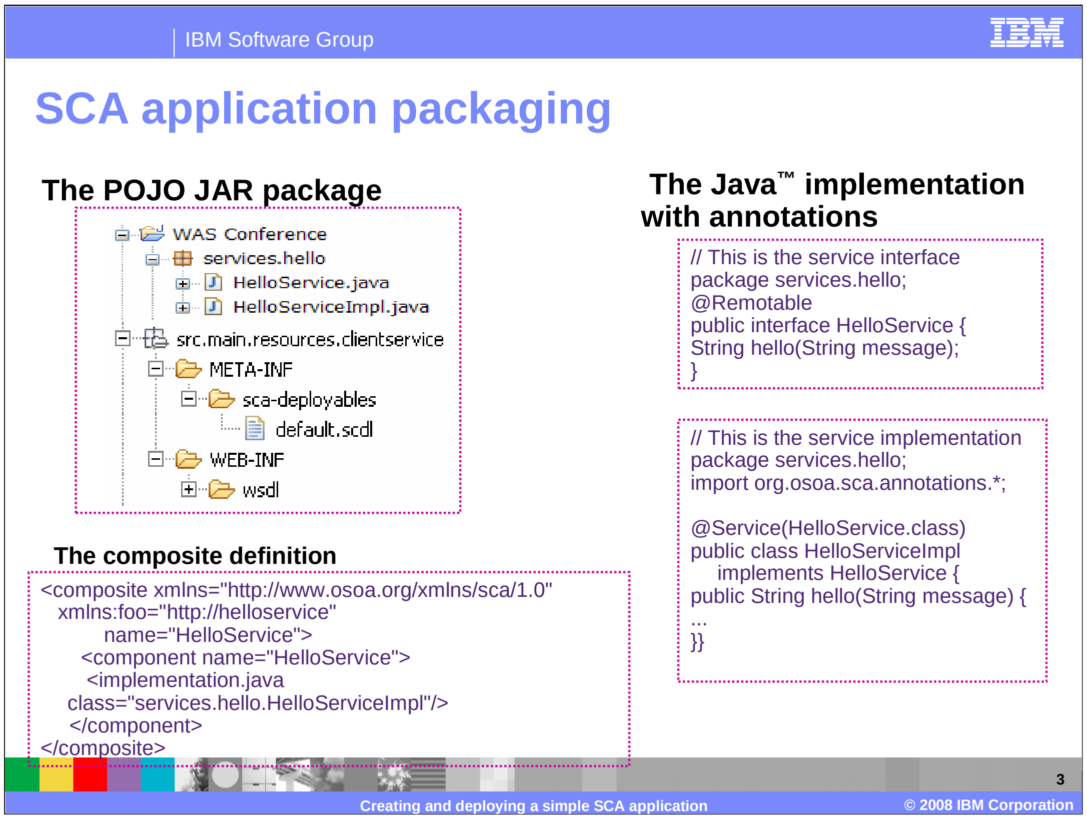
This presentation will show you how to create and deploy a simple SCA application.



Section

Creating and deploying a sample SCA application

This next section discusses how you go about creating and deploying a simple SCA application.



SCA application packaging

The POJO JAR package

```
WAS Conference
  services.hello
    HelloService.java
    HelloServiceImpl.java
  src.main.resources.clientservice
    META-INF
      sca-deployables
        default.scdl
    WEB-INF
      wsdl
```

The composite definition

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:foo="http://helloservice"
       name="HelloService">
    <component name="HelloService">
     <implementation.java
   class="services.hello.HelloServiceImpl"/>
   </component>
</composite>
```

The Java™ implementation with annotations

```
// This is the service interface
package services.hello;
@Remotable
public interface HelloService {
String hello(String message);
}
```

```
// This is the service implementation
package services.hello;
import org.osoa.sca.annotations.*;

@Service(HelloService.class)
public class HelloServiceImpl
    implements HelloService {
public String hello(String message) {
...
}}
```

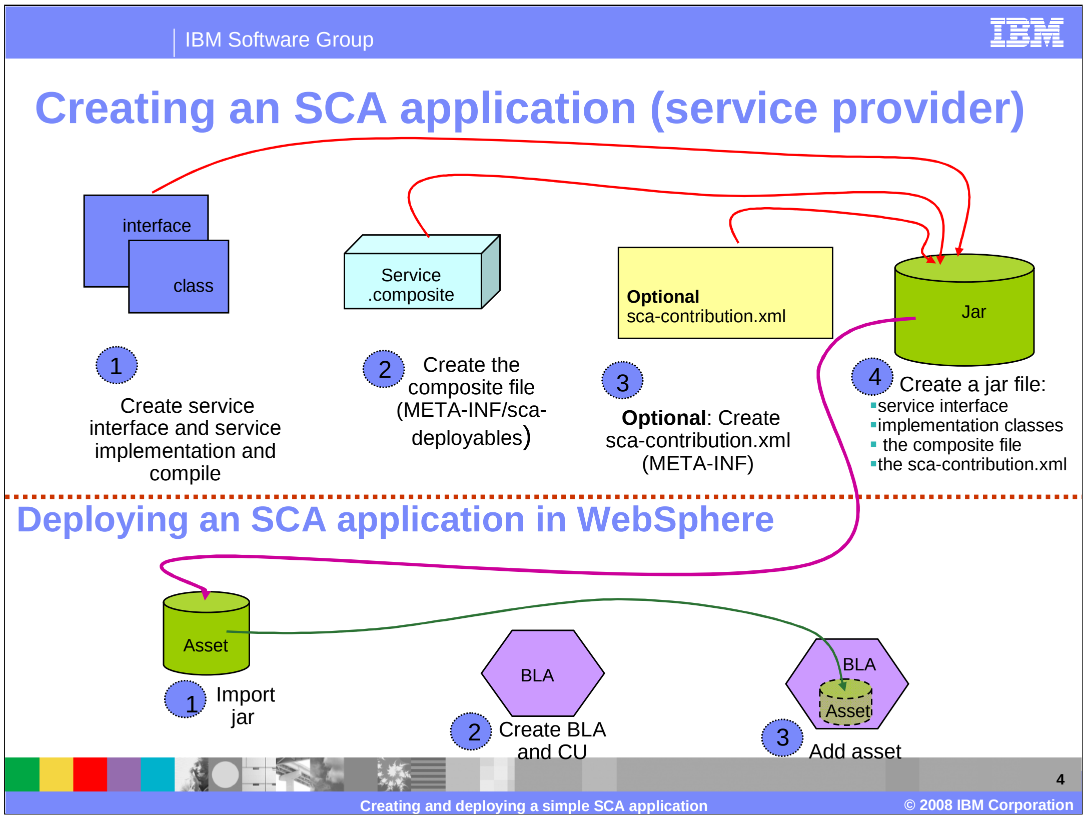
First, look at the packaging to see where things go before creating an application.

Take a look at the POJO JAR package on the top left side. As you can see, when you create a simple service interface class with its implementation class, these classes will go under a package. The **default.scdl** goes under the META-INF directory and, of course, if there was a wsdl involved, it goes under the WEB-INF directory. The composite files would go in the sca-deployables directory.

Also on that picture you can see a sample composite definition, the interface class and the implementation class. Note the use of the @service annotation that was discussed earlier and the @Remotable.

As a re-cap, the @Remotable annotation on a Java interface indicates that the interface is designed to be used for remote communication. Remotable interfaces are intended to be used for **coarse grained** services. Operations parameters and return values are passed **by-value**.

Note that the SCA Packaging model is not dependent on JEE JAR, zip, or WAR files; therefore it can be deployed directly into WebSphere Application Server.



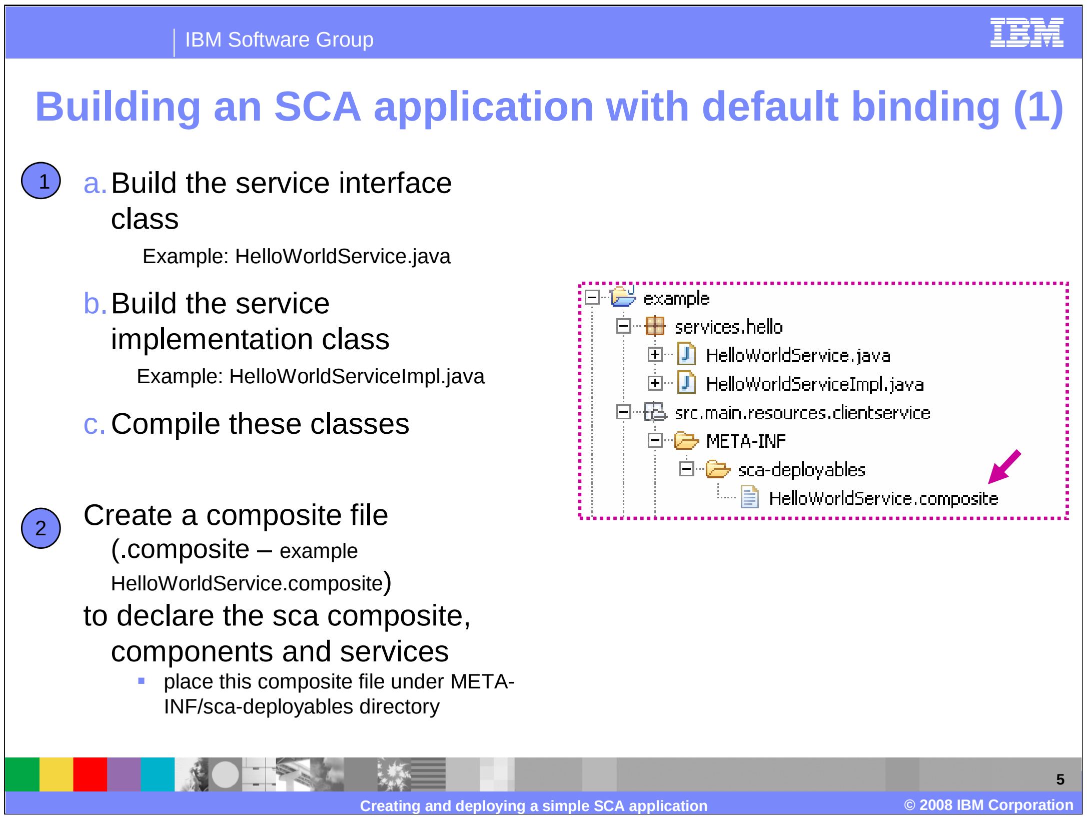
Creating an SCA application (service provider)

interface
class
Service .composite
Optional sca-contribution.xml
Jar
1 Create service interface and service implementation and compile
2 Create the composite file (META-INF/sca-deployables)
3 Optional: Create sca-contribution.xml (META-INF)
4 Create a jar file:
service interface
implementation classes
the composite file
the sca-contribution.xml

Deploying an SCA application in WebSphere

Asset
1 Import jar
BLA
2 Create BLA and CU
BLA
Asset
3 Add asset
4
Creating and deploying a simple SCA application
© 2008 IBM Corporation

This picture is a summary of the steps needed to create and deploy an SCA application. Under creating an SCA application, you can see the implementation classes, the composite file, contribution file and the .jar file. Under deployment, you can see the asset, business level application and the asset added into the business level application. Each of these steps is discussed in the next couple of slides.



Building an SCA application with default binding (1)

1 a. Build the service interface class

Example: HelloWorldService.java

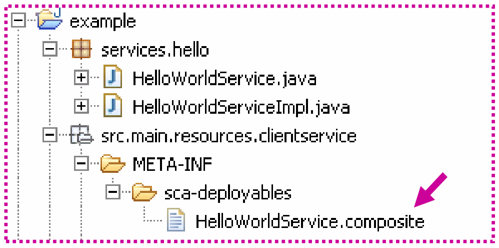
b. Build the service implementation class

Example: HelloWorldServiceImpl.java

c. Compile these classes

2 Create a composite file (.composite – example HelloWorldService.composite) to declare the sca composite, components and services

- place this composite file under META-INF/sca-deployables directory

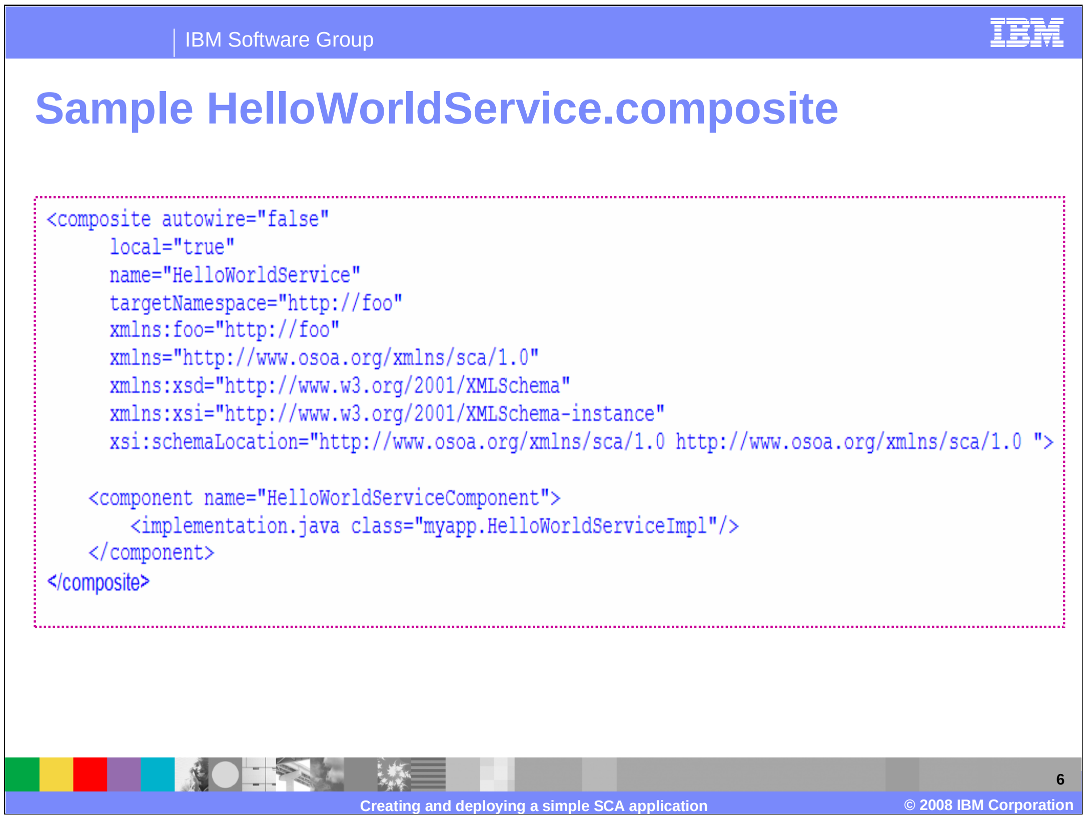


example
services.hello
HelloWorldService.java
HelloWorldServiceImpl.java
src.main.resources.clientservice
META-INF
sca-deployables
HelloWorldService.composite

Now that you have seen the general structure of how files are laid out in SCA, the next step is to create a simple SCA application step by step. The first step is to create the Java classes, the service interface classes and the service implementation classes, and then compile them.

In general, a best practice for application development is to start with WSDL as the interface, run wsimport to generate the Java interface. After that you can write the implementation from the generated Java interface and the generated JAXB objects.

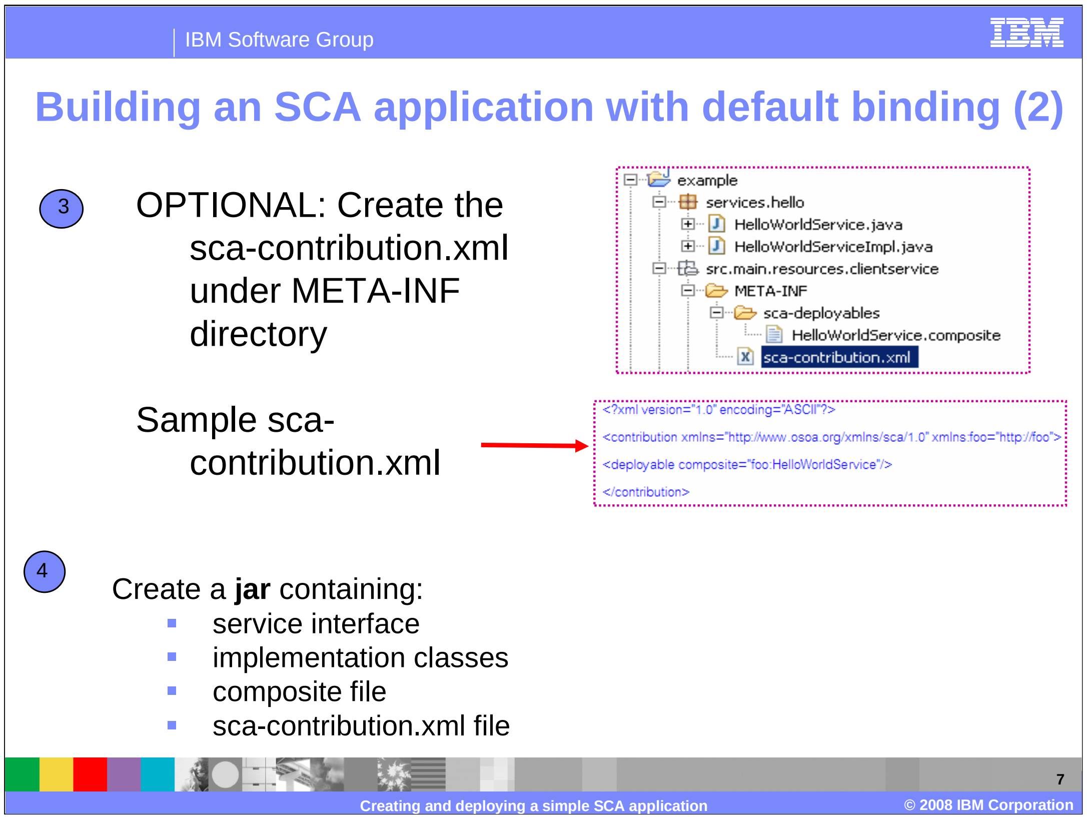
Once you have created your Java classes, the next step is to create a composite file to declare the SCA composite, components, and services. The name of the composite file and the directory where it is stored does not matter if you plan on creating the sca-contribution.xml file as explained later. Otherwise you need to place this composite file under META-INF/sca-deployables directory.



Sample HelloWorldService.composite

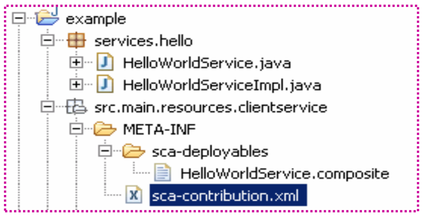
```
<composite autowire="false"
      local="true"
      name="HelloWorldService"
      targetNamespace="http://foo"
      xmlns:foo="http://foo"
      xmlns="http://www.osoa.org/xmlns/sca/1.0"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.osoa.org/xmlns/sca/1.0 http://www.osoa.org/xmlns/sca/1.0 ">

    <component name="HelloWorldServiceComponent">
        <implementation.java class="myapp.HelloWorldServiceImpl"/>
    </component>
</composite>
```

Here is a sample composite file and what it looks like. Note the specified namespace and the component name.



Building an SCA application with default binding (2)

3 OPTIONAL: Create the sca-contribution.xml under META-INF directory



Sample sca-contribution.xml

```
<?xml version="1.0" encoding="ASCII"?>
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0" xmlns:foo="http://foo">
<deployable composite="foo:HelloWorldService"/>
</contribution>
```

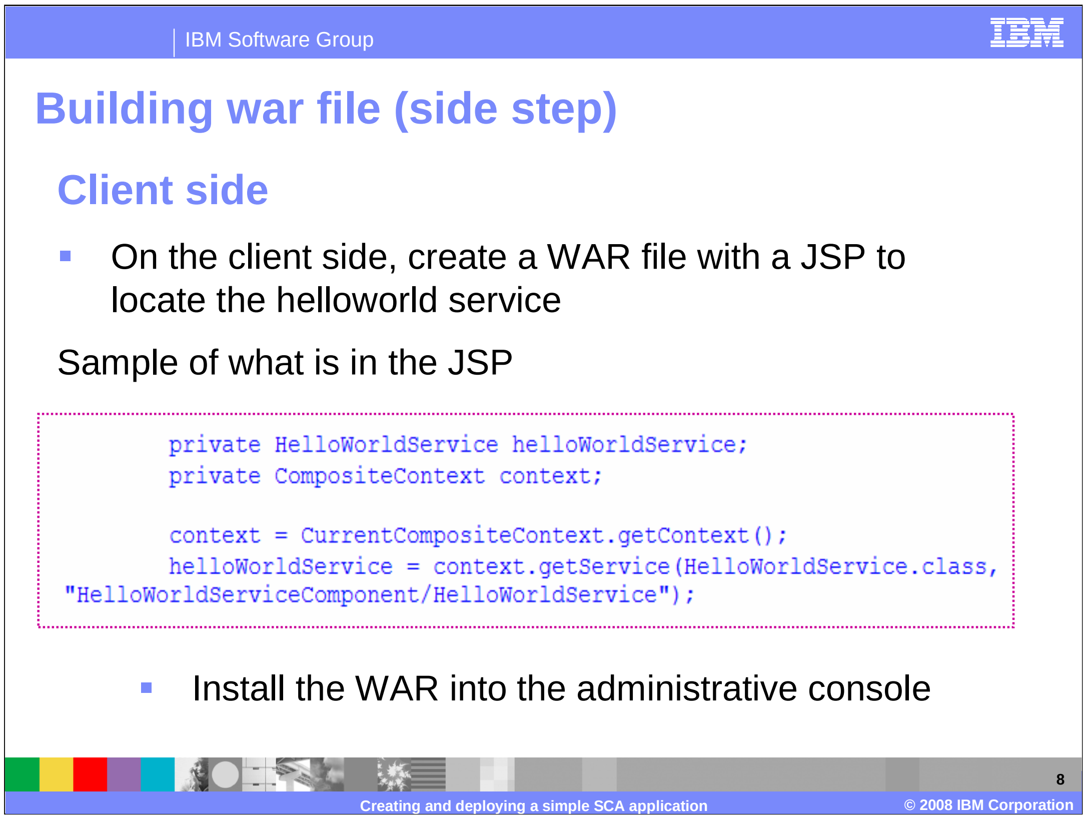
4 Create a **jar** containing:

- service interface
- implementation classes
- composite file
- sca-contribution.xml file

The third step is creating the **sca-contribution.xml** file under the META-INF directory. It is **optional** to create this file if the composite file is created in the sca_deployables directory. You can have composites anywhere in the system, and you can use a contribution file to declare the deployable composites. If you create a contribution file, make sure that the namespace under which the deployable composite is declared is the target namespace defined for that composite in its composite file. In the example shown, HelloWorldService composite is declared under the namespace http://foo, which is the target namespace for that composite declared in the HelloWorldService.composite.

The final step is to create a jar containing service interface and implementation classes, the composite file and the sca-contribution.xml file. This becomes your server side SCA application.

Now that an application has been created, you will see how you run and look at it.



Building war file (side step)

Client side

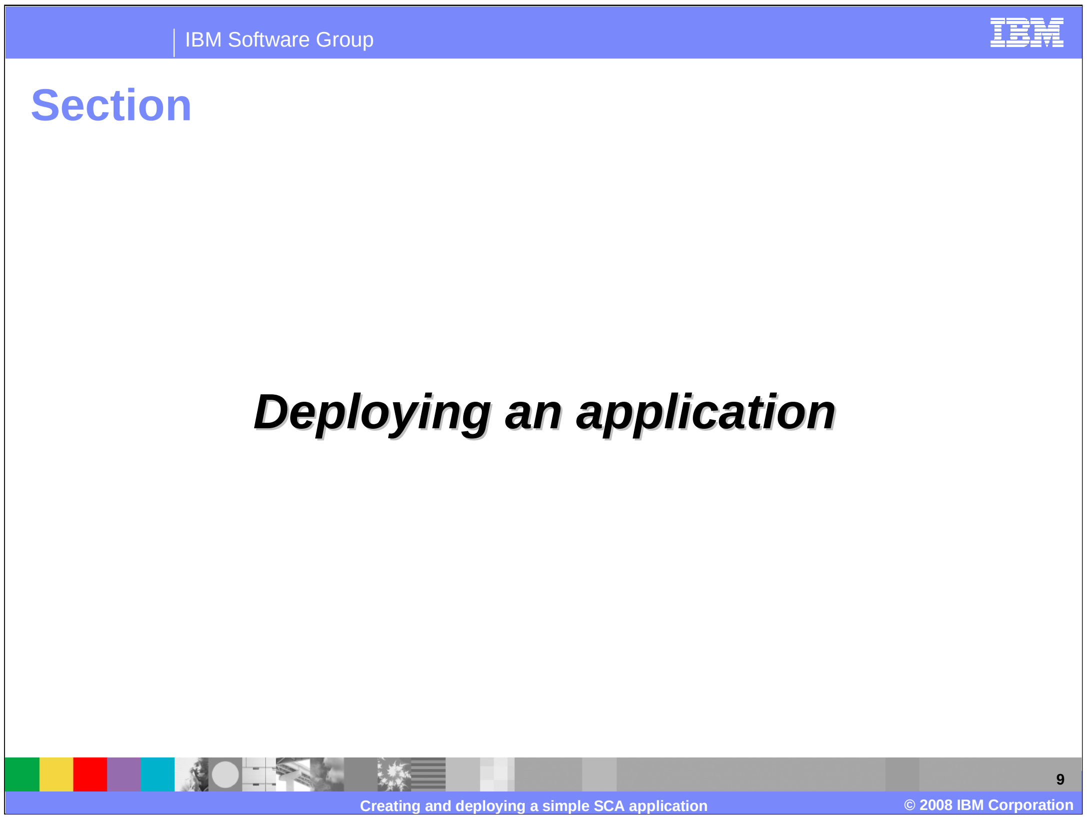
- On the client side, create a WAR file with a JSP to locate the helloworld service

Sample of what is in the JSP

```
        private HelloWorldService helloWorldService;
        private CompositeContext context;

        context = CurrentCompositeContext.getContext();
        helloWorldService = context.getService(HelloWorldService.class,
"HelloWorldServiceComponent/HelloWorldService");
```

- Install the WAR into the administrative console

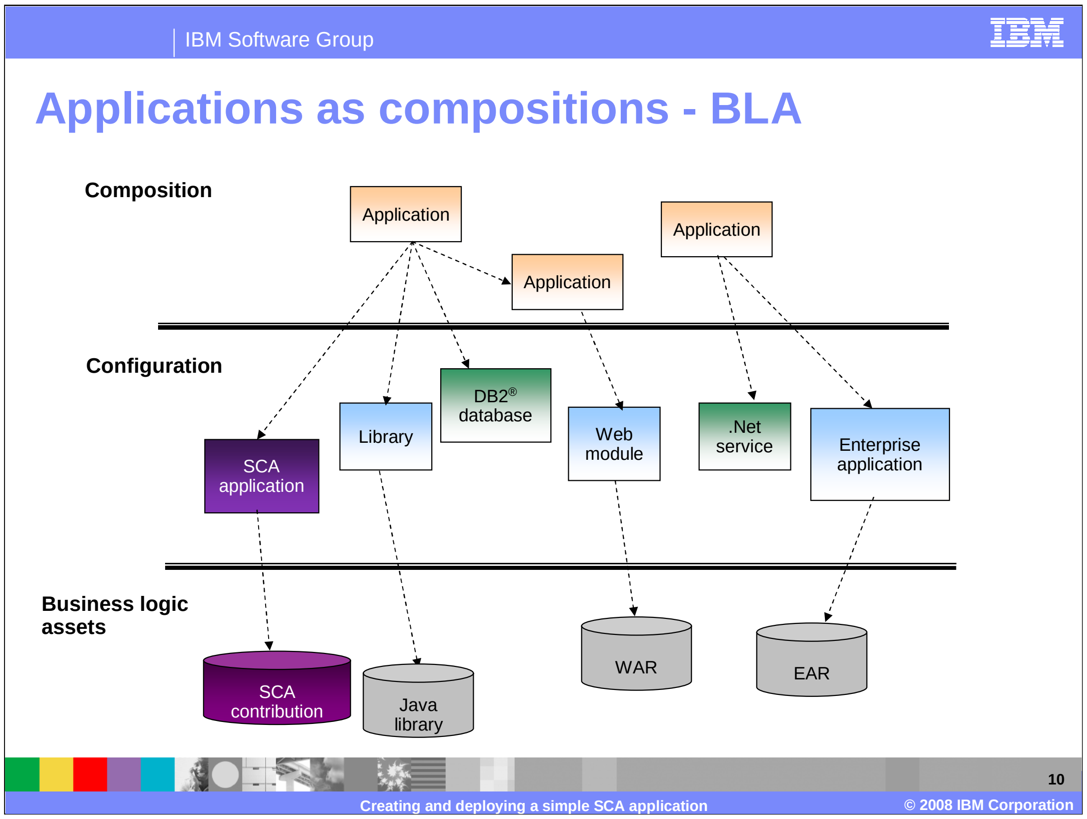
On the client side of things, you need to create a WAR file with a simple JSP to locate the HelloWorld Service. This step can be done anytime during the creation or deployment of an SCA application. The example on the slide shows what is inside the JSP. Note the **context.getService** to locate the HelloWorld Service. Note that context.getService is not supported on the Web services binding. Once you have the WAR file in place, you are now ready to deploy the application. You can install the WAR file through the administrative console just like any JEE application.



Section

Deploying an application

This section will show you how you deploy an SCA application using the administrative console.


IBM Software Group

Applications as compositions - BLA

10

Creating and deploying a simple SCA application

© 2008 IBM Corporation

In the administrative console, there is the concept of business level applications. A business level application, or BLA, is a WebSphere configuration artifact that has these characteristics:

1) It captures the entire definition of an enterprise level application consisting of WebSphere Application Server, though it may not explicitly manage the life cycle of every constituent part of its definition. It is a composition model for defining an application.

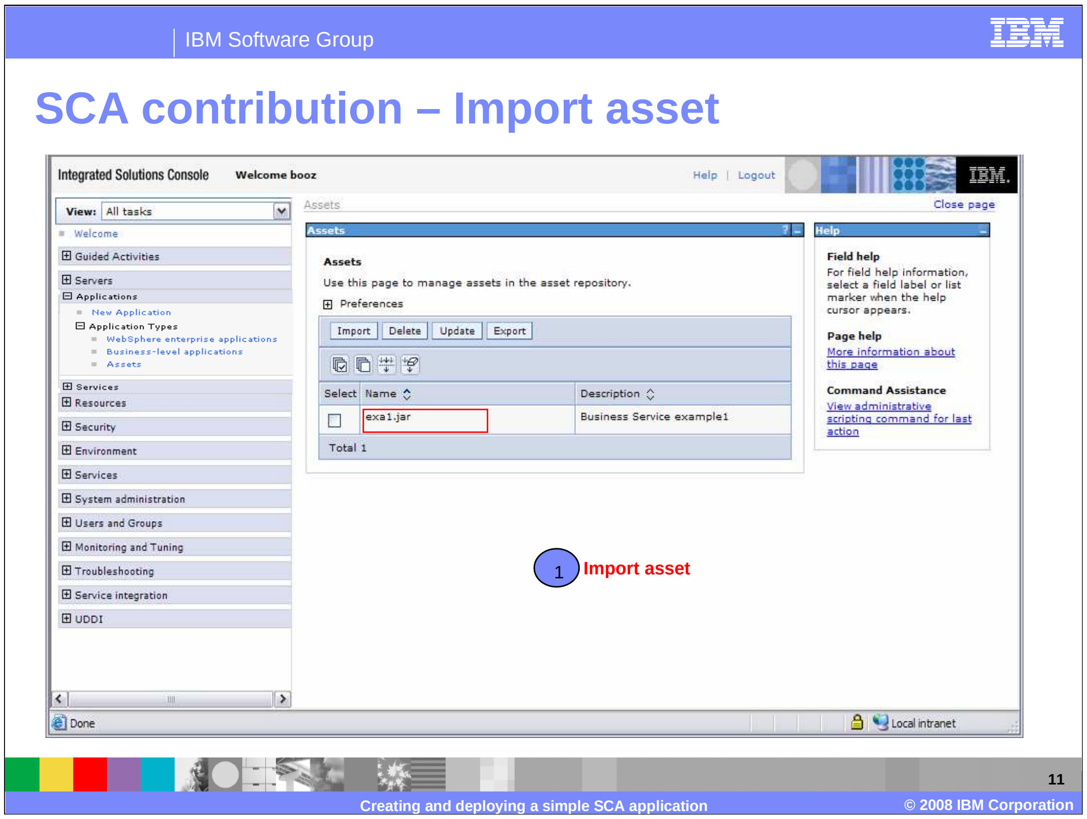
2) It does not represent or contain the application binaries. This provides a clean separation between the administration of binaries versus the administration of the application definition.

3) It also supports recursive composition. The composition at its lowest level consists of configured instances of application binaries that are configured to run in a specific runtime environment such as WebSphere Application Server.

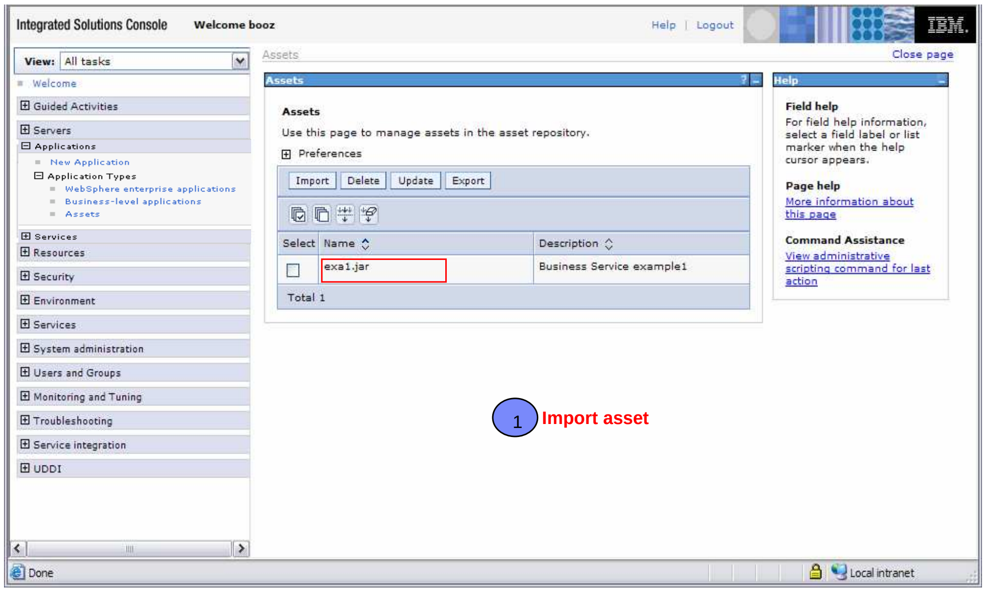
The above three-tier diagram shows the **composition model** for a business level application.

At the bottom are the assets such as the JAR files, WAR files, EAR files and so on and the **SCA contribution** and the **SCA WAR file.** The next level up is the composition units. The composition units are the deployed assets such as the EJB modules, Web modules, Web service modules and so on. This level also contains the **SCA application**. At the top level are the Business-level applications.

As an example, you can take an EAR file (an asset) and install it into WebSphere. This EAR file gets saved in the WebSphere repository as a composition unit, or CU. This CU can then get added to a BLA.



SCA contribution – Import asset



Integrated Solutions Console   Welcome booz   Help | Logout

View: All tasks

- Welcome
- ⊞ Guided Activities
- ⊞ Servers
- ⊟ Applications
  - New Application
  - ⊟ Application Types
    - WebSphere enterprise applications
    - Business-level applications
    - Assets
- ⊞ Services
- ⊞ Resources
- ⊞ Security
- ⊞ Environment
- ⊞ Services
- ⊞ System administration
- ⊞ Users and Groups
- ⊞ Monitoring and Tuning
- ⊞ Troubleshooting
- ⊞ Service integration
- ⊞ UDDI

Assets   Close page

Assets

Use this page to manage assets in the asset repository.

⊞ Preferences

Import | Delete | Update | Export

Select	Name	Description
☐	exa1.jar	Business Service example1
Total 1		

Help

Field help
For field help information, select a field label or list marker when the help cursor appears.

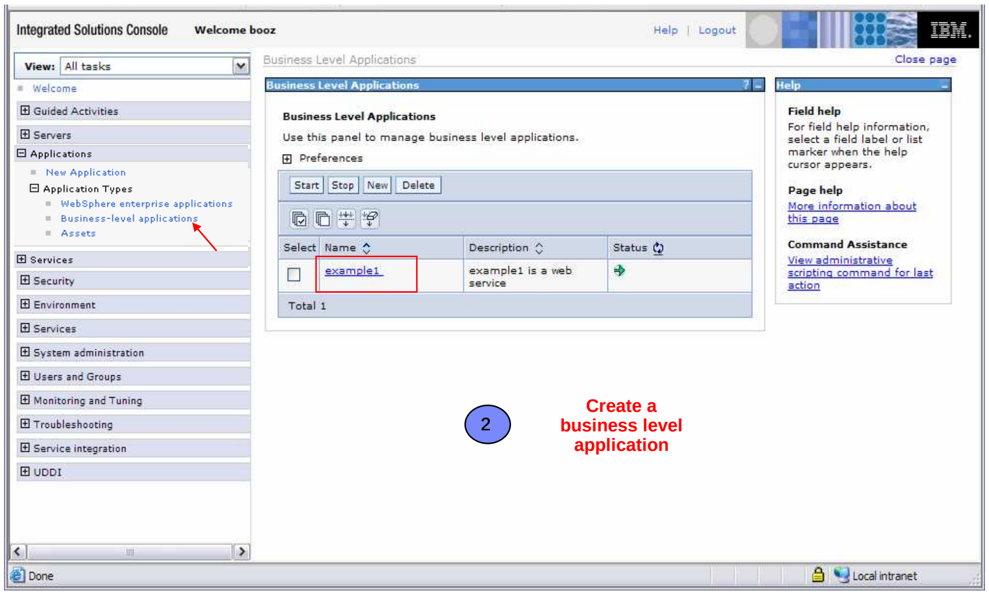
Page help
More information about this page

Command Assistance
View administrative scripting command for last action

Done   Local intranet

Now to continue with the deployment of the SCA application that was created a couple of slides earlier. If the jar you created is called exa1.jar, that JAR file needs to be imported as an asset. That is done in the administrative console through the menu option, Applications->ApplicationTypes->Assets and then select “Import.”

Administrative console experience - BLAs


Integrated Solutions Console Welcome booz

Help | Logout

View: All tasks

- Welcome
- Guided Activities
- Servers
- Applications
  - New Application
  - Application Types
    - WebSphere enterprise applications
    - Business-level applications
    - Assets
- Services
- Security
- Environment
- Services
- System administration
- Users and Groups
- Monitoring and Tuning
- Troubleshooting
- Service integration
- UDDI

Business Level Applications

Close page

Business Level Applications

Use this panel to manage business level applications.

Preferences

Start Stop New Delete

Select	Name	Description	Status
☐	example1	example1 is a web service	➔
Total 1			

Help

Field help

For field help information, select a field label or list marker when the help cursor appears.

Page help

More information about this page

Command Assistance

View administrative scripting command for last action

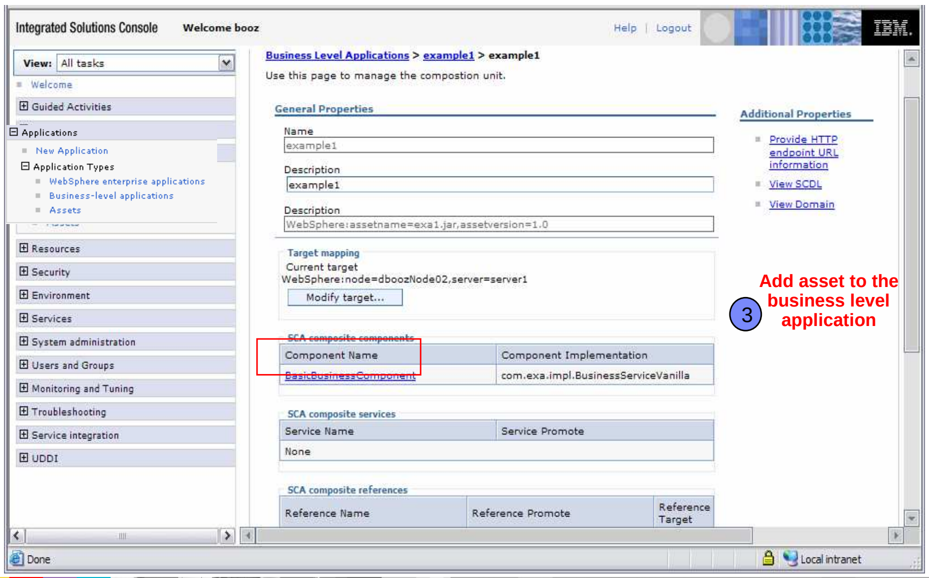
Done

Local intranet

2 Create a business level application

Next you create a business level application from the menu Applications->Application Types->Business level Applications.

Administrative console experience – SCA composite


Integrated Solutions Console Welcome booz Help | Logout

View: All tasks

- Welcome
- Guided Activities
- Applications
  - New Application
  - Application Types
    - WebSphere enterprise applications
    - Business-level applications
    - Assets
- Resources
- Security
- Environment
- Services
- System administration
- Users and Groups
- Monitoring and Tuning
- Troubleshooting
- Service integration
- UDDI

Business Level Applications > example1 > example1

Use this page to manage the compostion unit.

General Properties

Name: example1

Description: example1

Description: WebSphere:assetname=exa1.jar,assetversion=1.0

Target mapping

Current target
WebSphere:node=dboozNode02,server=server1

Modify target...

SCA composite components

Component Name	Component Implementation
BasicBusinessComponent	com.exa.impl.BusinessServiceVanilla

SCA composite services

Service Name	Service Promote
None	

SCA composite references

Reference Name	Reference Promote	Reference Target

Additional Properties

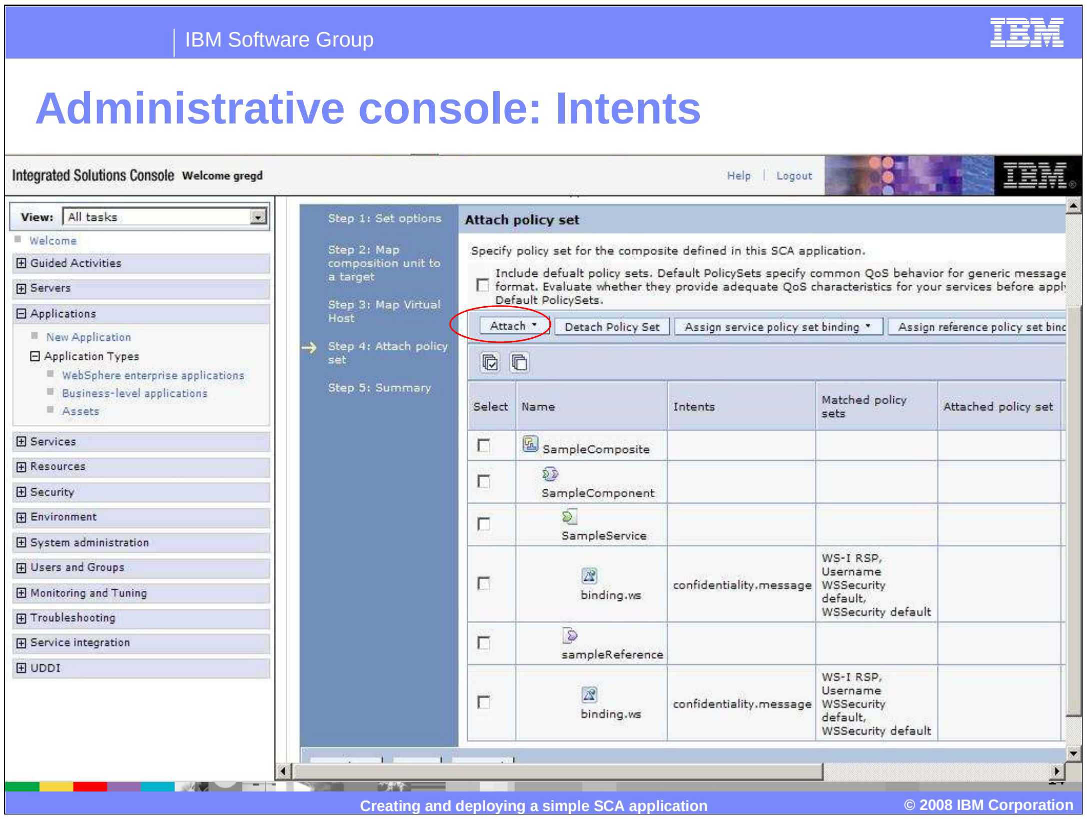
- Provide HTTP endpoint URL information
- View SCDL
- View Domain

Done Local intranet

Next you add the asset under the BLA, which will create the SCA composite components. From the picture, note the SCA composite component created. You can also install the war file from the administrative console so that the service is seen. Once that is done, you are ready to start the SCA application.

This concludes the SCA application creation and deployment process. Bear in mind that this is a very simple case of creating an SCA application. Many other things can come into play like the quality of service when creating SCA applications. For example, you could add authorization policies to composites and configure user roles on the administrative console during deployment. Other presentations look at security and authorization as quality of services.

The next three slides show you a quick snapshot of some of the things you might configure on an SCA application.



Administrative console: Intents

Integrated Solutions Console Welcome gregd

Attach policy set

Specify policy set for the composite defined in this SCA application.

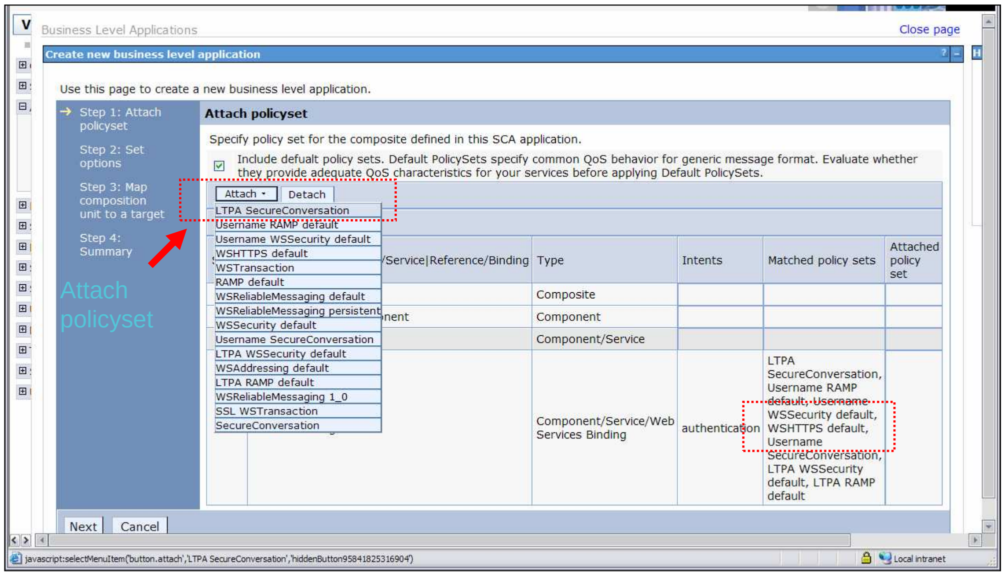
Include default policy sets. Default PolicySets specify common QoS behavior for generic message format. Evaluate whether they provide adequate QoS characteristics for your services before apply Default PolicySets.

Attach
Detach Policy Set
Assign service policy set binding
Assign reference policy set binc

Select	Name	Intents	Matched policy sets	Attached policy set
☐	SampleComposite			
☐	SampleComponent			
☐	SampleService			
☐	binding.ws	confidentiality.message	WS-I RSP, Username WSSecurity default, WSSecurity default	
☐	sampleReference			
☐	binding.ws	confidentiality.message	WS-I RSP, Username WSSecurity default, WSSecurity default	

As an example, from the administrative console, intents can be specified for your application optionally when adding assets with policy set defined to business level applications.

Security policy example – Map to policySet



Business Level Applications

Close page

Create new business level application

Use this page to create a new business level application.

Step 1: Attach policyset

Step 2: Set options

Step 3: Map composition unit to a target

Step 4: Summary

Attach policyset

Attach policyset

Specify policy set for the composite defined in this SCA application.

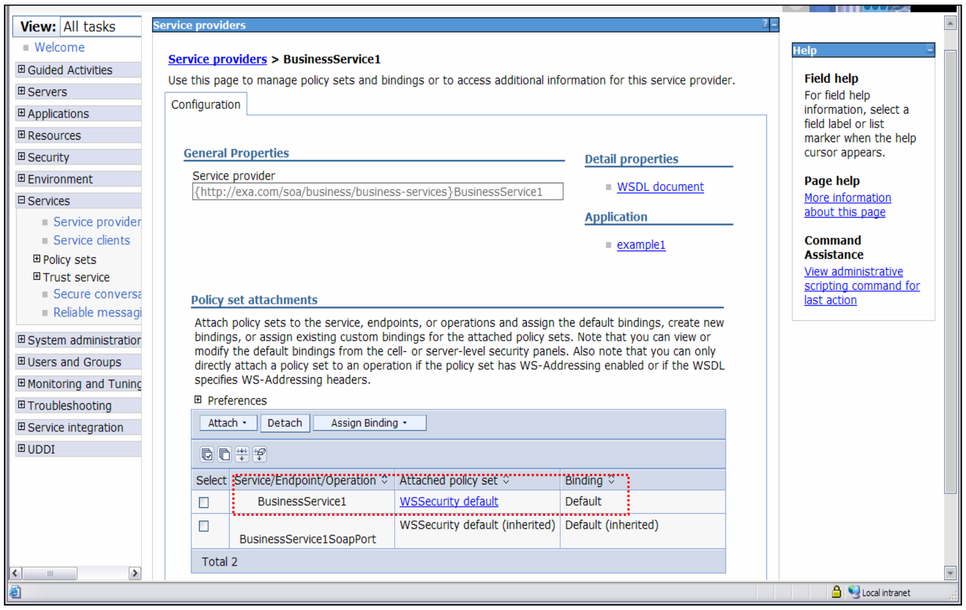
☑ Include defualt policy sets. Default PolicySets specify common QoS behavior for generic message format. Evaluate whether they provide adequate QoS characteristics for your services before applying Default PolicySets.

/Service|Reference/Binding	Type	Intents	Matched policy sets	Attached policy set
	Composite			
nent	Component			
	Component/Service			
	Component/Service/Web Services Binding	authentication	LTPA SecureConversation, Username RAMP default, Username WSSecurity default, WSHTTPS default, Username SecureConversation, LTPA WSSecurity default, LTPA RAMP default	

javascript:selectMenuItem('button.attach','LTPA SecureConversation','hiddenButton95841825316904')

This example shows how the intent is mapped to the policy set on the administrative console using the “**Attach**” button.

Security policy example – Service providers



Service providers

Service providers > BusinessService1

Use this page to manage policy sets and bindings or to access additional information for this service provider.

Configuration

General Properties

Service provider: {http://exa.com/soa/business/business-services}BusinessService1

Detail properties
- WSDL document

Application
- example1

Policy set attachments

Attach policy sets to the service, endpoints, or operations and assign the default bindings, create new bindings, or assign existing custom bindings for the attached policy sets. Note that you can view or modify the default bindings from the cell- or server-level security panels. Also note that you can only directly attach a policy set to an operation if the policy set has WS-Addressing enabled or if the WSDL specifies WS-Addressing headers.

Select	Service/Endpoint/Operation	Attached policy set	Binding
☐	BusinessService1	WSSecurity default	Default
☐	BusinessService1SoapPort	WSSecurity default (inherited)	Default (inherited)

Total 2

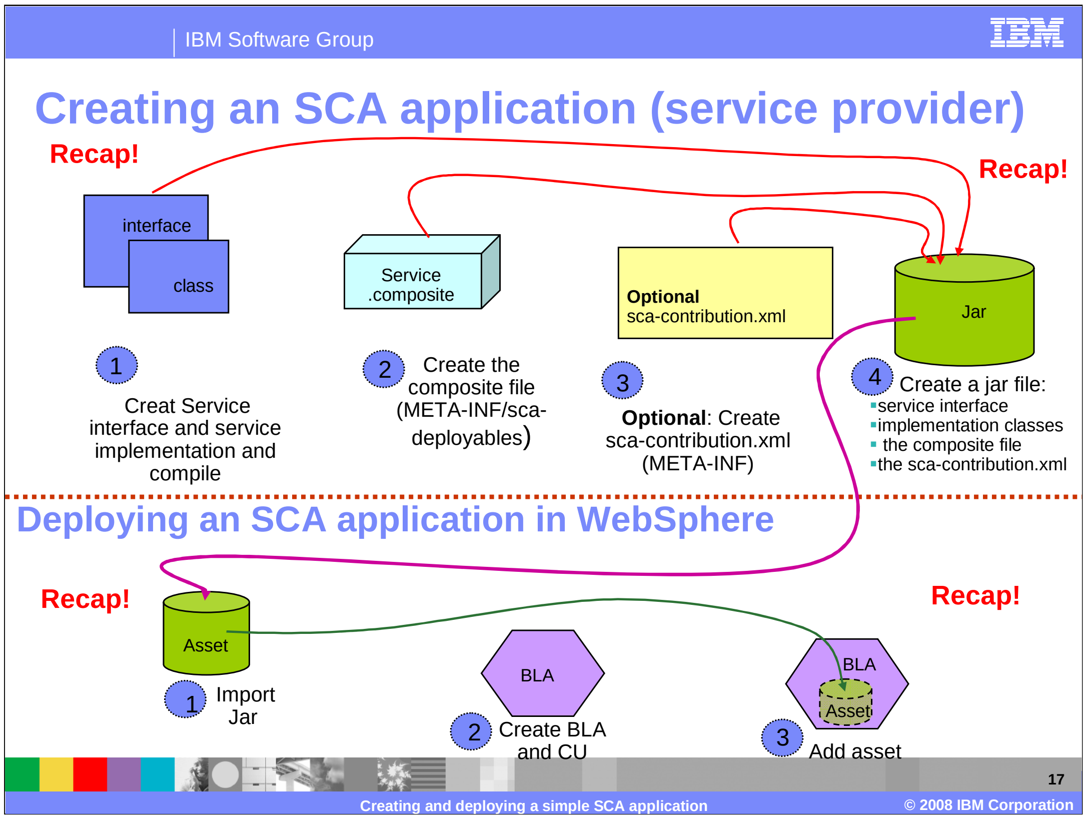
Help

Field help
For field help information, select a field label or list marker when the help cursor appears.

Page help
More information about this page

Command Assistance
View administrative scripting command for last action

You can also see from the service providers the policy set that got attached to the service component.



Creating an SCA application (service provider)

Recap!
interface
class
Service .composite
Optional sca-contribution.xml
Jar
Recap!

1 Creat Service interface and service implementation and compile

2 Create the composite file (META-INF/sca-deployables)

3 **Optional**: Create sca-contribution.xml (META-INF)

4 Create a jar file:
- service interface
- implementation classes
- the composite file
- the sca-contribution.xml

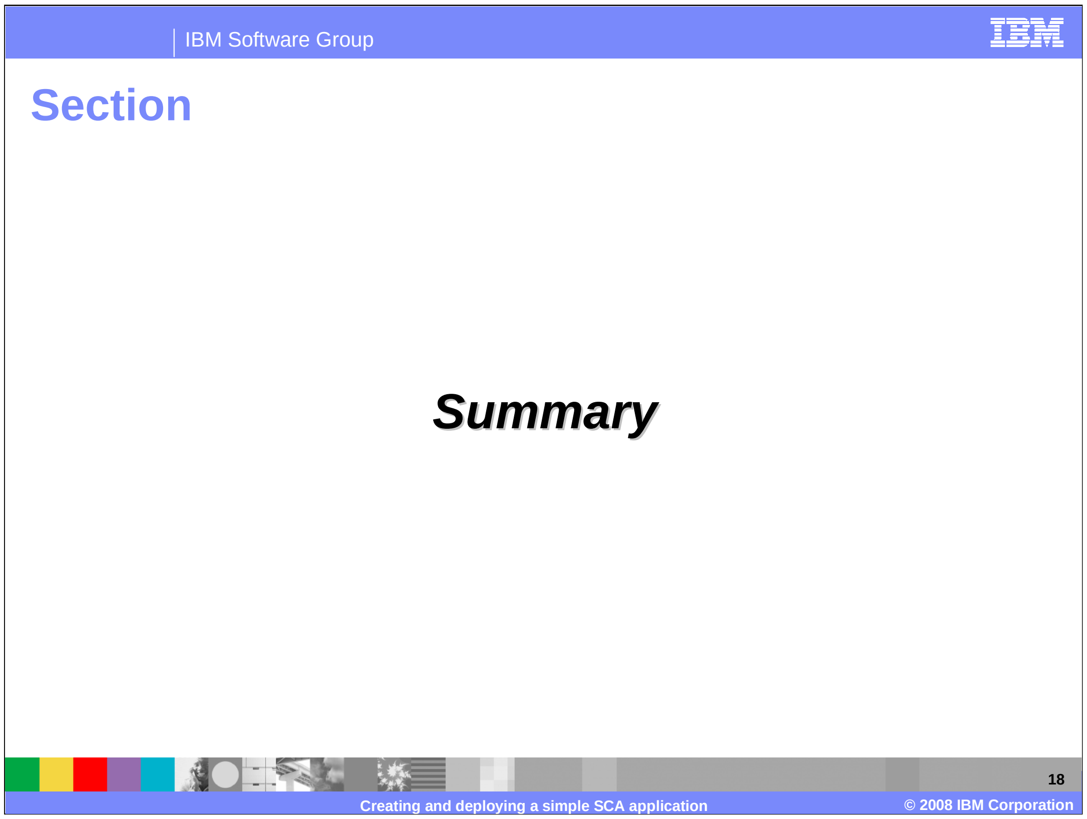
Deploying an SCA application in WebSphere

1 Import Jar

2 Create BLA and CU

3 Add asset

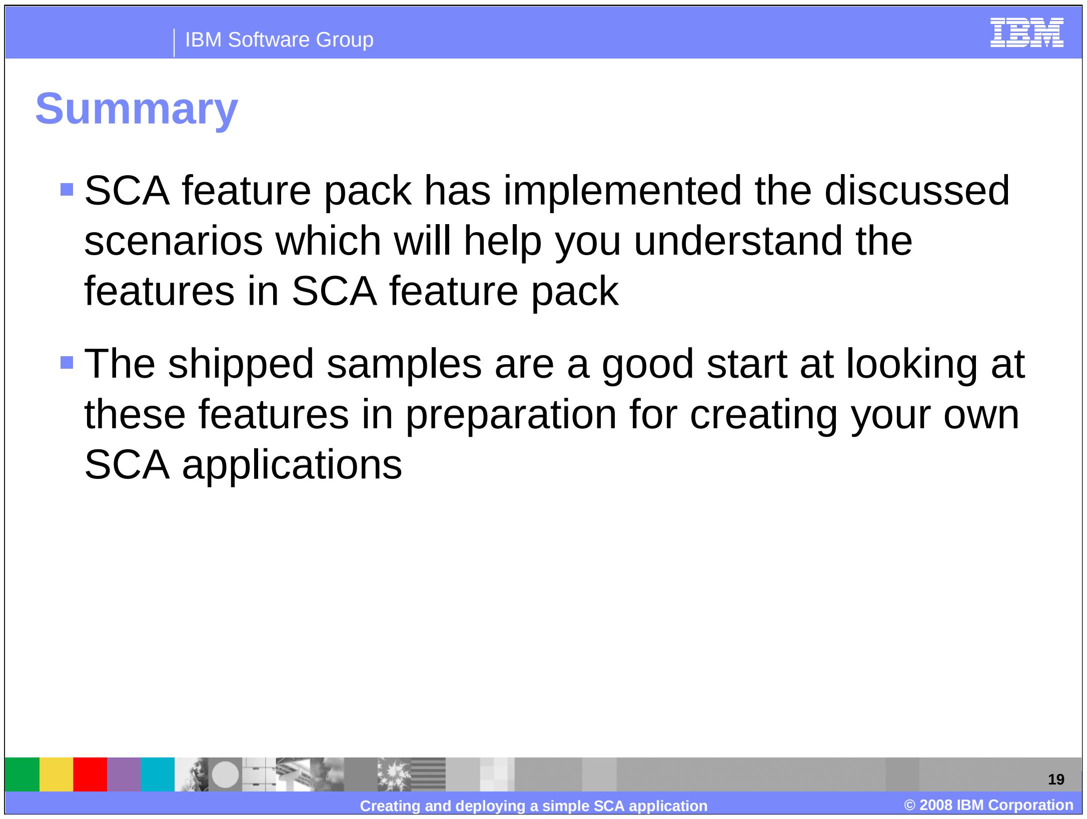
To summarize the steps one more time: Under creating an SCA application, you create the implementation classes, the composite file, contribution file and the jar. Under Deployment, you import the .jar file as an asset, create a business level application, and add the asset into the business level application.



Section

Summary

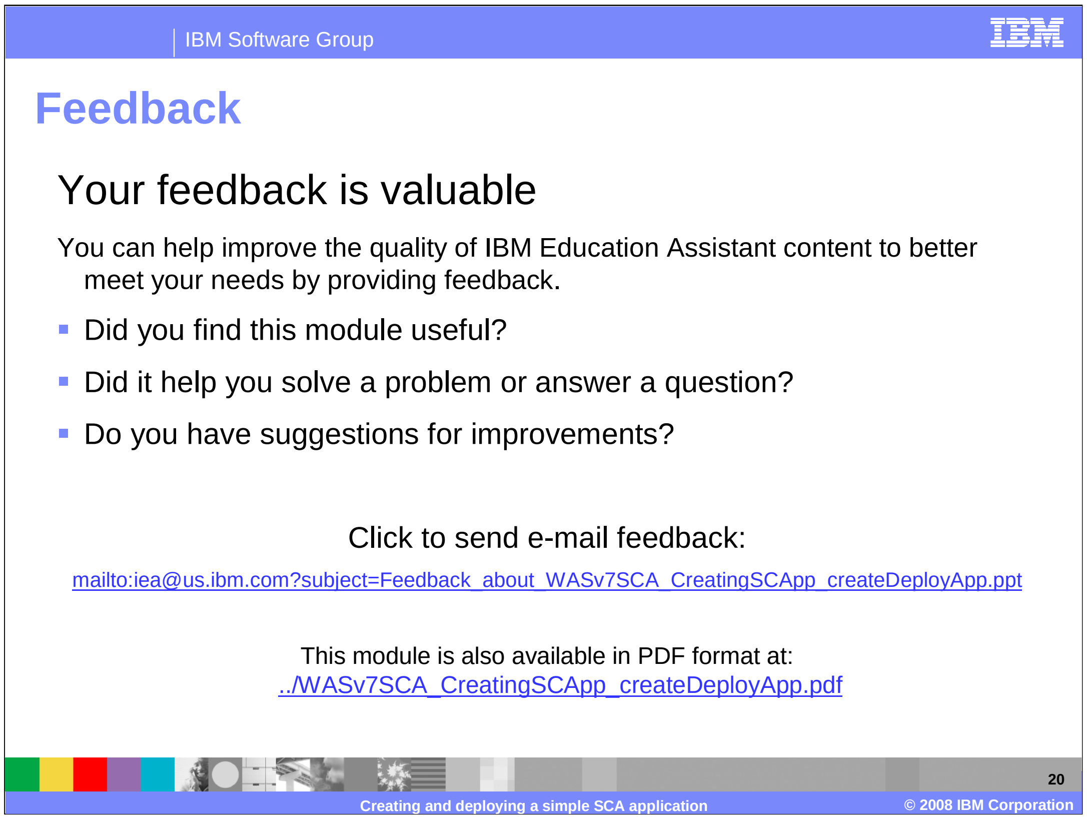
In summary, this presentation looked at the creation and deployment of an SCA application.



Summary

- SCA feature pack has implemented the discussed scenarios which will help you understand the features in SCA feature pack
- The shipped samples are a good start at looking at these features in preparation for creating your own SCA applications

The SCA feature pack allows you to choose from three different development approaches. A good starting point to learn about your options is with the samples that are provided with the product.



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_WASv7SCA_CreatingSCApp_createDeployApp.ppt

This module is also available in PDF format at:
../WASv7SCA_CreatingSCApp_createDeployApp.pdf

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

Trademarks, copyrights, and disclaimers