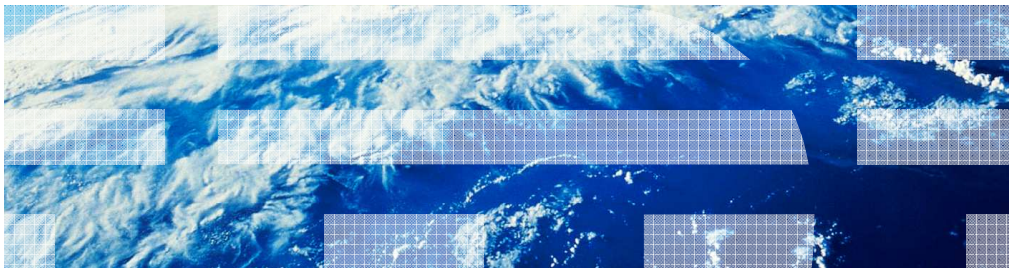


# IBM WebSphere Application Server Communications Enabled Applications

web services development



The Communications Enabled Applications feature of WebSphere Application Server Feature allows you to access telephony services with web service clients. This presentation goes through the steps needed to create and deploy an application that can manage telephone calls using the web services interface, including how to configure the application server. Formerly, this capability required building SIP servlets and a detailed understanding of the SIP specification. This functionality greatly reduces the amount of code required.

## Agenda

- Manage telephone calls using a web services client
- Configuration
- Development
- web service sample
- Configuring external web service providers to use the Communications Enabled Applications (CEA) functionality

This presentation covers managing telephone calls using the Communications Enabled Applications (CEA) web service, configuring your server and system, and developing an application that uses the CEA web service. It then covers the web service sample that is included in the CEA samples package that you can download from the WebSphere Application Server Samples site. The presentation ends with a brief overview of how to configure external web service providers to use the CEA functionality.

## ***Manage telephone calls using a web services client***

First is a brief overview of the Communications Enabled Applications (CEA) web service and an overview of the steps needed to create and deploy an application that can manage telephone calls using the web services interface.

## CEA web service (1 of 2)

- The Communications Enabled Applications (CEA) feature of WebSphere Application Server allows you to integrate telephony services into new and existing applications using its web services interface
- The CEA capability lets you:
  - Open a session to start monitoring a telephone
  - Get notifications about telephone activity
  - Make telephone calls between two phones
  - End an active telephone call
  - Close a session to stop monitoring a telephone

The Communications Enabled Applications feature of WebSphere Application Server allows you to integrate telephony services into new and existing applications using its web services interface. This functionality lets you open a session to start monitoring a telephone and get notifications about telephone activity. You can also make telephone calls between two telephones, end an active telephone call, and close a session to stop monitoring a telephone.

## CEA web service (2 of 2)

- The CEA web service
  - Communicates over the HTTP protocol used on the Web
  - XML messages follow the SOAP standard
  - Description of operations offered by the service are written in web services Description Language (WSDL)

A web service is designed to support interaction over a network and is frequently just web application programming interfaces that can be accessed over a network, and run on a remote system hosting the requested services. A typical web service application communicates over the HTTP protocol used on the Web. Within the Communications Enabled Applications feature, a Web service uses XML messages that follow the SOAP standard where there is a machine-readable description of the operations offered by the service written in the web services Description Language (WSDL). The WSDL file can be interpreted by Web service tools to generate the Web services client code needed to communicate with the Web service. As a result, an application developer need only call the correct set of Java APIs to manage telephone calls in an application.

## Steps to accessing telephony services with WS

- Enable the CEA system application
- Install IP-PBX
- Configure the IP-PBX location
- Restart your server
- Develop an application
- Install and start your application

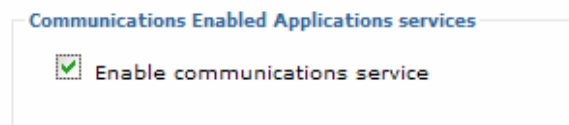
In order to access telephony services with web services you need to follow these steps. First, enable the CEA system application in your WebSphere Application Server. Second, install and configure your IP-PBX and restart your server. Then develop your application and install and start your application.

## ***Configuration***

The next few slides will walk through the configuration steps that need to be done in order to access telephony services with web services.

## Enable CEA system application

- Update the configuration for each server running communications enabled applications
  - In the administrative console, click Servers > Server Types > WebSphere application servers > *server\_name* > Communications Enabled Applications (CEA)
  - Ensure the check box labeled **Enable communications service** is checked



First, enable the CEA system application. For each server running communications enabled applications, update the configuration to ensure that communications service is enabled. In the administrative console, click Servers > Server Types > WebSphere application servers > *server\_name* > Communications Enabled Applications (CEA). Ensure the check box labeled **Enable communications service** is checked as shown here.



## IP-PBX

- The CEA capability requires an IP private branch exchange (PBX) as part of your infrastructure
- Install and start the sample IP-PBX application (commsvc.pbx.ear) included in the CEA samples package available on the WebSphere Application Server Samples site
  - [http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-nd-mp&topic=welcome\\_samples](http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-nd-mp&topic=welcome_samples)
- Along with the sample IP-PBX, two soft phones are needed to test the application

The Communications Enabled Applications capability requires an IP private branch exchange (PBX) as part of your infrastructure. An IP-PBX is a business telephone system designed to deliver voice over a data network and interoperate with the Public Switched Telephone Network (PSTN). A sample IP-PBX application is included in the application server installation. The sample IP-PBX is in the form of an application enterprise archive (EAR) file and is for test purposes only. The details of installing and configuring a vendor-specific IP-PBX are not provided. The IP-PBX must support the ECMA TR/87 protocol. Along with the sample IP-PBX, two soft phones are needed to test the application.

To set up a sample IP-PBX application that you can use for unit testing in the absence of an official PBX, start the application server where you deploy the sample IP-PBX application. Then install the SIP IP-PBX sample application. Then start the application.

## Configure the IP-PBX location

- In the administrative console, click Servers > Server Types > WebSphere application servers > server\_name > Communications Enabled Applications (CEA)
- Use the CEA settings page to select the Use SIP CTI (ECMA TR/87) gateway for telephony access option and configure these fields:
  - Host name or IP address
  - Port
  - Protocol (TCP)
  - Superuser name
- Restart the application server

Telephony access method

Use SIP CTI (ECMA TR/87) gateway for telephony access

\* Host name or IP address  
localhost

\* Port  
5060

\* Protocol  
TCP

Extract user name from request

Superuser name  
ceauser

After installing and starting the IP-PBX, configure the IP-PBX location. In the administrative console for the server where the Communications Enabled Applications (CEA) system application is running, click Servers > Server Types > WebSphere application servers > *server\_name* > Communications Enabled Applications (CEA). On the CEA settings page, select “Use SIP CTI (ECMA TR/87) gateway for telephony access” and configure the host name or IP address, port, protocol, and superuser name fields. Be sure to set the fields based on the server that is running the PBX application.

Use the host name field to provide the fully-qualified host name or IP address of the SIP CTI gateway that the CEA services connect to; the default is localhost. The port field specifies the port number of the SIP CTI gateway for connection communication services. For the TCP Protocol, the port is the SIP\_DEFAULTHOST for the server that is running the PBX application; the default is 5060. If you are not using the default port then enter your port number here. In the protocol field, provide the protocol to use when connecting to the SIP CTI gateway; the default is TCP. The superuser name field specifies the name that is used when opening a new session to the gateway; the default is root. Be sure to restart the application server.

# ***Development***

Now that you have your application server configured you can now integrate telephony services into new and existing applications.

## Development process

- Obtain WSDL files and associated schema file
- Generate through JAX-WS a client using ControllerService.wsdl
- Write code to call methods against the web service client
- Adding notification
  - Generate through JAX-WS a service implementation class using CeaNotificationConsumer.wsdl
  - Implement the notify method to receive and process notification messages
- The rest is handled by the runtime

The development process is outlined here. In communications enabled applications, a web service uses XML messages that follow the SOAP standard where there is a machine-readable description of the operations offered by the service written in WSDL. The WSDL file can be interpreted by web service tools to generate the web services client code needed to communicate with the web service. The Communications Enabled Applications (CEA) feature provides two WSDL descriptions of the operations offered by the Web service. You will first obtain the WSDL files ControllerService.wsdl and CeaNotification.wsdl and associated schema files. The ControllerService.wsdl file generates through JAX-WS the Web services client code needed to communicate with the Web service. Generated files include openSession, closeSession, makeCall, and endCall. As a result, you need only call the correct set of Java APIs to manage telephone calls in an application. Your code should include method calls to open a session, make a call, end a call, and close a session.

The CEA Web service support is built on WS-Notification. The CeaNotificationConsumer.wsdl follows WS-Notification allowing the CEA Web service to participate in publish and subscribe messaging patterns. The WSDL describes the consumer service. The CeaNotificationConsumer.wsdl file generates through JAX-WS a service implementation class, CeaNotificationConsumerSOAPImpl.java and a NotificationConsumer.java file. As a result, you just need to Implement the notify() method to receive and process notification messages, notifying you of the call status. The rest is handled by the runtime; neither the client code nor the provider code are required to write or parse SOAP messages.

## Obtain the WSDL files

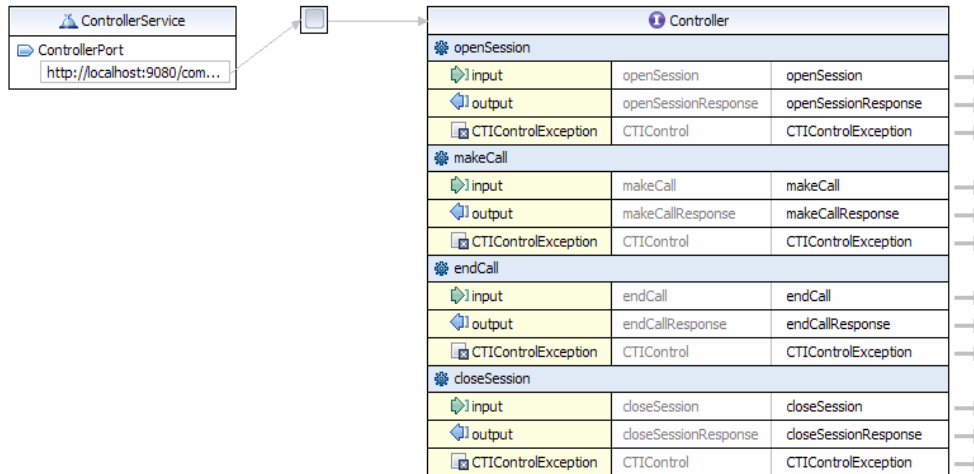
- Point your web browser to
  - <http://host:port/commsvc.rest/ControllerService?wsdl>
  - <http://host:port/commsvc.rest/CeaNotificationConsumer?wsdl>
  - WSDL file created in the file system during installation is read by the web services infrastructure
- Save the file and import it into your application
- Import associated schema file
  - [http://host:port/commsvc.rest/ControllerService/WEB-INF/wsdl/ControllerService\\_schema1.xsd](http://host:port/commsvc.rest/ControllerService/WEB-INF/wsdl/ControllerService_schema1.xsd)

The Communications Enabled Applications (CEA) feature includes WSDL descriptions of the operations offered by the web service. In order to obtain the WSDL files, point your web browser to the two URLs shown here.

In these URLs, “host” is the IP address or host name on which the Web container is listening, and “port” is the port number on which the Web container is listening.

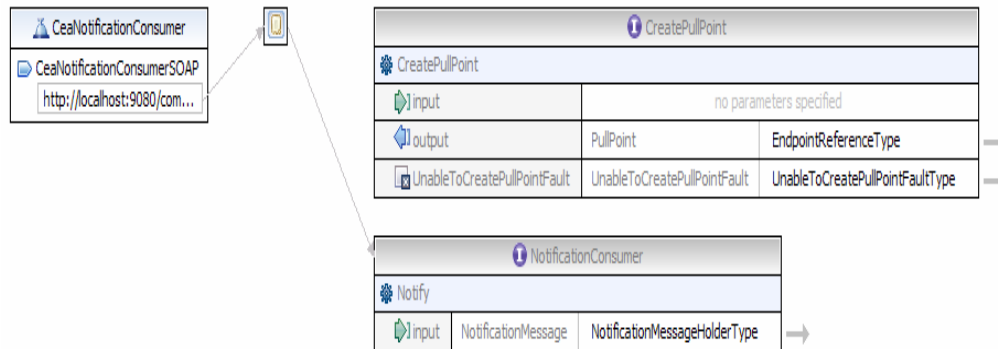
When you do this, the WSDL file created in the file system during installation is read by the web services infrastructure, and the correct host and port are configured in the WSDL file sent to the browser. Save the file and import it into your application. You will also need to import the associated schema file, which is the last URL listed on this slide.

## ControllerService.wsdl



The ControllerService.wsdl file contains description of the operations offered by the service. Using web service tools, you can generate the web services client code needed to communicate with the web service. Generated files include openSession, makeCall, endCall, and closeSession. On this slide is a visual display of the WSDL file.

## CeaNotificationConsumer.wsdl



The CeaNotificationConsumer.wsdl follows WS-Notification, allowing the CEA web service to participate in publish and subscribe messaging patterns. The WSDL describes the consumer service. The CeaNotificationConsumer.wsdl file generates through JAX-WS a service implementation class, CeaNotificationConsumerSOAPImpl.java and a NotificationConsumer.java file. As a result, you just need to Implement the notify() method to receive and process notification messages, notifying you of the call status. On this slide is a partial visual display of the WSDL file.

## Generating Java artifacts

- JAX-WS tools support generating Java artifacts when starting with a WSDL file
  - Create a service client from a WSDL
  - Create a skeleton bean from a WSDL
- Rational Application Developer has the web services tools that use the WebSphere JAX-WS runtime environment
- Command line tools available

The Java API for XML web Services (JAX-WS) is a Java programming language API for creating web services. JAX-WS uses annotations to simplify the development and deployment of web service clients and endpoints. JAX-WS represents remote procedure calls or messages using XML-based protocols such as SOAP, but hides SOAP's innate complexity behind a Java-based API. Developers use this API to define methods, then code one or more classes to implement those methods and leave the communication details to the underlying JAX-WS API. Clients create a local proxy to represent a service, then invoke methods on the proxy. The JAX-WS runtime system converts API calls and matching replies to and from SOAP messages.

JAX-WS tools support generating Java artifacts when starting with a WSDL file. You can create a service client from a WSDL file. Web service clients are created from a WSDL document which describes where the Web service is deployed and what operations this service provides. You can also create a skeleton bean from a WSDL file. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

Rational Application Developer has the Web services tools that use the WebSphere JAX-WS runtime environment. There are also command line tools available.



## Rational Application Developer web service wizard

- Switch to the Java EE perspective
- Right click your WSDL file
  - choose web Services > Generate client or
  - Generate Java bean skeleton
- Select the stages of web service client to Develop, select your server, runtime, and service project
- For Generate Java bean skeleton: select Top down Java bean web service as your web service type
- Files are generated



Using the WebSphere JAX-WS runtime environment Rational Application Developer has the web services tools that allow you to either create a service client from a WSDL file or create a skeleton bean from a WSDL file. In order to use the web service wizard in Rational Application Developer follow these steps. First switch to the Java EE perspective (Window > Open Perspective > Java EE). Import your WSDL file and associated schema file. Right click your WSDL file and choose web services > Generate client or Generate Java bean skeleton. For ControllerService.wsdl choose generate client and for CeaNotificationConsumer.wsdl choose generate Java bean skeleton.

On the web services page select the stages of web service development that you want to complete using the slider. This will set several default values on the remaining wizard panels. You will want to set the slider to develop, which will develop the WSDL definition and implementation of the Web service. This includes such tasks as creating the modules that will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate. Select your server, runtime, and service project.

If creating a Java bean skeleton from a WSDL be sure to select Top down Java bean Web service as your Web service type. Top-down Web services development involves creating a Web service from a WSDL file. Go through the rest of wizard selecting other details and click Finish. Your Java files will then be generated.

## Command line

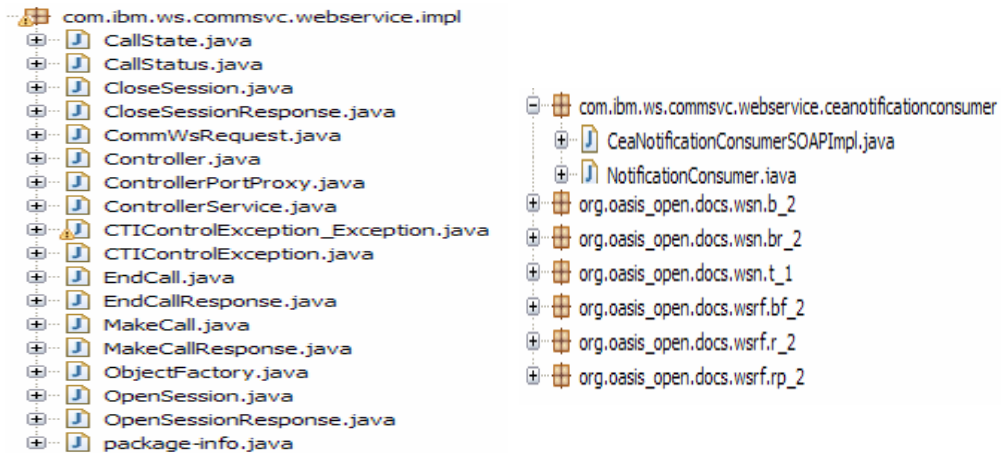
- wsimport (top down) generates:
  - Service endpoint interface (SEI)
  - Service class
  - Exception class that is mapped from the wsdl:fault class (if any)
  - Java Architecture for XML Binding (JAXB) generated type values
- <WAS\_HOME>\bin\wsimport -keep -b <WAS\_HOME>\util\ibm-wsn-jaxb.xml -wsdllocation <WSDL\_LOC> <WSDL\_FILE>
  - Example:
    - C:\opt7\ibm\websphere\appserver\bin\wsimport -keep -b C:\opt7\ibm\websphere\appserver\util\ibm-wsn-jaxb.xml -wsdllocation "WEB-INF/wsdl/CeaNotificationConsumer.wsdl" CeaNotificationConsumer.wsdl

There are two main command line tools for working with JAX-WS to develop web services. WSImport is a top down development tool that will create the necessary beans, service client, service endpoint interface, and wrappers from a provided WSDL file. The WSGen command will create a WSDL document and wrappers when needed from Java code with the proper web service annotations.

Use wsimport, to process a WSDL file and generate portable Java artifacts that are used to create a web service client. Using the wsimport tool you can create Service endpoint interface (SEI), and Service class. You also create the exception class that is mapped from the wsdl:fault class (if any), and Java Architecture for XML Binding (JAXB) generated type values which are Java classes mapped from XML schema types. Use the **-verbose** option to see a list of generated files when you run the command. Use the **-keep** option to keep generated Java files. Use the **-wsdlLocation** option to specify the location of the WSDL file. Use the **-b** option if you are using WSDL or schema customizations to specify external binding files that contain your customizations. You can customize the bindings in your WSDL file to enable asynchronous mappings or attachments.

## Generated files

- ControllerService.wsdl
  - web services client code
- CeaNotificationConsumer.wsdl
  - Java service implementation class and other generated Java files



This picture shows the web services client code that was generated from the `ControllerService.wsdl` file and the Java service implementation class and other oasis files that were generated from the `CeaNotificationConsumer.wsdl` file.

## ControllerService.java

```
@WebServiceClient(name = "ControllerService", targetNamespace = "http://impl.webservice.commsvc.ws.ibm.com/",
    wsdlLocation = "WEB-INF/wsdl/ControllerService.wsdl")
public class ControllerService
    extends Service
{
    ...
    @WebEndpoint(name = "ControllerPort")
    public Controller getControllerPort() {
```

JAX-WS uses annotations to simplify the development and deployment of web service clients and endpoints. Here is some of the code from the generated Java file `ControllerService.java`. This file was generated from `ControllerService.wsdl`. Notice the annotations. The `@WebServiceClient` is used to annotate a generated service interface. The information specified in this annotation is sufficient to uniquely identify a `wsdl:service` element inside a WSDL document. This `wsdl:service` element represents the web service for which the generated service interface provides a client view. The `@WebEndpoint` is used to annotate the `getPortName()` methods of a generated service interface. The information specified in this annotation is sufficient to uniquely identify a `wsdl:port` element inside a `wsdl:service`.

## What to code

- Write client side code to call methods against the web service client
- For notification
  - Implement the notify method to receive and process notification messages
    - Notify method is in CeaNotificationConsumerSOAPImpl.java
  - Add client side code to set the NotifyCallback in the web service request object
  - Create a method to update the telephone session with call status information that arrived in a WS-Notification

After generating your Java code from the WSDL files you will write client side code to call methods against the web service client. You will want to write client code to open a session, make a call, end a call, and close a session. For notification you will need to implement the notify method to receive and process notification messages. The notify() method is in the CeaNotificationConsumerSOAPImpl.java file. After coding the notify method you will want to add client side code to set the NotifyCallback in the web service request object. You will also want to create a method to update the telephone session with call status information that arrived in a WS-Notification.

## Sample code

```
public accessWebService () {
    // Access the web services client
    controllerService = ControllerService();
    if (controllerService!= null) {
        controllerPort = controllerService.getControllerPort();
    } ... }
// Open a session to monitor/control a phone
public void openSession(String addressOfRecord, String notifyCallback) {
    // Build a request object
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setAddressOfRecord(addressOfRecord);
    wsRequest.setNotifyCallback(notifyCallback);
    W3CEndpointReference EPR = controllerPort.openSession(wsRequest);
    controllerPortWithEPR = EPR.getPort(Controller.class, new AddressFeature(true));
}
```

Here is some sample client code. This code shows a sample `accessWebService()` method that gets access to the web service client. Also, shown here is a sample `openSession()` method. `openSession()` is called in order to start monitoring a telephone. In this method you will first build the web service request object, then access the web service, and call the web service to open the session. You will use the endpoint reference (EPR) to create a new object to make Web service calls on. The EPR includes information that allows the Web service to map requests to this session. The EPR must be used in all other APIs called related to the session monitoring that telephone. The EPR is critical for multiple reasons. First, it allows for Web service interface to be simpler eliminating the need to pass a state object as a parameter in all follow up requests. The EPR itself has enough information for the Web service to track it. It is also used in a clustered environment to ensure follow on requests go back to the same container monitoring the telephone.

Notice that `notifyCallback` is set. This is the URL needed to contact in order to trigger a call notification (WS-Notification). For the URL the host and port must be where the Web service client resides. The context root must match that of the WAR including this Web services client, and the name at the end must match the service name in the `CeaNotificationConsumer.wsdl`.

## Sample code continued

```
// Make a call
public void makeCall(String calleeAddressOfRecord) {
    // Build a request object
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setPeerAddressOfRecord(calleeAddressOfRecord);
    // Make the call, using the EPR returned in openSession()
    controllerPortWithEPR.makeCall(wsRequest);
}
// End an active call
public void endCall() {
    // End the call, using the EPR returned in openSession()
    controllerPortWithEPR.endCall(wsRequest);
}
// Close the session monitoring the phone
public void closeSession() {
    // Close the session, using the EPR returned in openSession()
    controllerPortWithEPR.closeSession();
}
```

This is sample code to make a call, end a call, and close a session. In this code, the callee is the person being called. Note the use of the EPR in each of the methods.

## Implement notify method

- Extract the list of notification messages
- Loop through the messages
- Get the message content as a DOM Element
- Build a CallStatus object out of the notification
- Loop through and match the text to a member of the CallStatus object and set it
- Update the status of the associated client state object

For notification you will need to implement the `notify()` method to receive and process notification messages. The `notify()` method is in the `CeaNotificationConsumerSOAPImpl.java` file. These steps list what a sample `notify` method should contain. First you extract the list of notification messages. Next you loop through the messages and get the message content as a DOM Element. You then build a `CallStatus` object out of the notification by looping through and matching the text to a member of the `CallStatus` object and setting it. Finally, you update the status of the associated client state object. The `notify()` method is called automatically by the server's notification broker when a notification takes place.



## New application

- Bundle up your application that includes JAX-WS “annotated” classes, WSDL, XSD schema, and any client side code you created
- Install your sample
- Take the application-specific action that triggers the call to the web service API
- In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone
  - This can be a URI (uniform resource indicator) of a telephone
    - A SIP URI, for example, has the format of sip:username@serviceprovider, which represents the address of your telephone on the Internet
- Answer the source telephone when it rings
- Answer the destination telephone when it rings

After developing your application, bundle up your application that includes JAX-WS “annotated” classes, WSDL, XSD schema, and any client side code you created. Install your sample in your application server. Take the application-specific action that triggers the call to the web service API. In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone. This can be a URI (uniform resource indicator) of a telephone. A SIP URI, for example, has the format of sip:username@serviceprovider, which represents the address of your telephone on the Internet. Answer the source telephone when it rings and answer the destination telephone when it rings.

## ***web service sample***

The next two slides show the web service sample included in the Communications Enabled Applications (CEA) samples package that you can download from the WebSphere Application Server Samples site.



## web services sample

- Download the sample:  
[http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-nd-mp&topic=welcome\\_samples](http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-nd-mp&topic=welcome_samples)
- Install the EAR file: <WAS\_HOME>\feature\_packs\cea\samples\webservice.sample\commsvc.ws.sample.ear
- Visit:  
<http://host:port/commsvc.ws.sample/CommWebServiceServlet>

### CEA Web Service Sample

#### Open a session to monitor a phone

Phone address of record:

27

Web services development

© 2011 IBM Corporation

A web services sample is included in the Communications Enabled Applications (CEA) samples package that you can download from the WebSphere Application Server Samples site. You can view the code and see the generated Java files and sample client side code. In order to run the web service sample, you will need to install the sample. After installing and starting the application, browse to the URL shown here. In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone. This can be a URI of a telephone. Enter the URI value in the telephone address of record field and select Open session. Remember to complete the configuration steps earlier in the presentation before running this sample.

## Making a call

**1**

### Phone Status

- Address of record: sip:Customer@localhost
- Call status: CALL\_STATUS\_CLEARED
- Caller: null
- Callee: null
- Call ID: null

Refresh call status

---

Peer address of record:

**3**

---

- Address of record: sip:Customer@localhost
- Call status: CALL\_STATUS\_ESTABLISHED
- Caller: sip:Customer@localhost
- Callee: sip:CSR@localhost
- Call ID: local.1245717055562\_80

Refresh call status

---

**2**

### Phone Status

- Address of record: sip:Customer@localhost
- Call status: CALL\_STATUS\_DELIVERED
- Caller: sip:Customer@localhost
- Callee: sip:CSR@localhost
- Call ID: local.1245717055562\_80

Refresh call status

---

**4**

### Phone Status

- Address of record: sip:Customer@localhost
- Call status: CALL\_STATUS\_CLEARED
- Caller: null
- Callee: null
- Call ID: null

Refresh call status

---

Peer address of record:

28      Web services development      © 2011 IBM Corporation

Starting in the left corner you will see that there is not a call going on yet, but your address of record is filled in. Next enter a URI for the callee - the number that you want to call - and click "Make call". At the top right you will now see that the status is updated showing the caller, callee and the call status stating that the call was delivered. On the bottom left after both the phones are answered and after refreshing the status the call status will now state that the call is established. Finally, if you click "End the call," you will see on the bottom right that the call status is cleared.

## ***Configuring external web service providers***

This section shows the steps needed to configure an external web service provider.

## Using IP-PBX

- The CEA feature provides web telephony services in the form of REST APIs in addition to a web service
- When invoked, a common core technology interacts with an IP-PBX to monitor and control phones
  - This core technology can be substituted with an external web service to manage all communications with the IP-PBX

The Communications Enabled Applications feature provides web telephony services in the form of REST APIs in addition to a web service. When invoked, a common core technology interacts with an IP-PBX to monitor and control telephones. This core technology can be substituted with an external web service to manage all communications with the IP-PBX.

## External web service provider


- If external provider creates a web service that supports the CEA WSDL, application can be configured to use that provider
  - External web service provider must be deployed and running on a server accessible from the application server
    - WSDL file for the external service must be known and accessible using an HTTP request
  - This configuration replaces the need for the existing CEA web service, but the existing service can be used for REST requests

The Communications Enabled Applications (CEA) web service interface is described by a WSDL file. If an external provider creates a web service that supports this WSDL, then the communications enabled application can be configured to use that provider. To use an external web service provider, it must be deployed and running on a server accessible from the application server. The location of the WSDL file for the external service must be known and accessible by using an HTTP request. Like the setup required when using the web service provided by the CEA feature, you must start and configure an IP private branch exchange (PBX) as well.

This configuration replaces the need for the existing CEA Web service, but the existing service can be used for REST requests. As REST requests are received, the application server uses a Web services client to communicate with the external Web service provider. The external Web service provider manages all communications with the IP-PBX.

## Configure the external web service

- Install and configure the external web service
- Configure the location of the vendor web service WSDL
  - In the administrative console, click Servers > Server Types > WebSphere application servers > *server\_name* > Communications Enabled Applications (CEA)
  - Under Telephony access method, select the Use a third-party web services provider for telephony access option
  - Enter the HTTP URL of the third-party WSDL
  - Save the settings and restart the server so that the new changes are applied to the run time

 Use a third-party Web services provider for telephony access

Third-party Web services provider's WSDL

You will need to install and configure the external web service. For example, if the external web service is delivered as an application that is deployed on WebSphere Application Server, you must install and configure the service on the local server. Configure the location of the vendor web service WSDL. In the administrative console, click Servers > Server Types > WebSphere application servers > *server\_name* > Communications Enabled Applications (CEA). Under Telephony access method, select “Use a third-party web services provider for telephony access”. Enter the URL of the third-party WSDL in the “Third-party Web services provider's WSDL” field. Save the settings and restart the server so that the new changes are applied to the run time.



## ***Summary and references***

This section provides a summary and references.

## Summary

- The Communications Enabled Applications feature of WebSphere Application Server lets you integrate telephony services into new and existing applications using the web services interface
  - Obtain WSDL files and associated schema file
  - Generate client code using ControllerService.wsdl
  - Write code to call methods against the web service client
  - Add Notification
- web services sample
- Configure external web service providers

This presentation talked about managing telephone calls using the Communications Enabled Applications (CEA) web service, configuring your server and system, and developing an application that uses the CEA web service. When developing an application, the main steps are to obtain the WSDL files and associated schema files, then generate the Java code. With the generated Java code you can just write client code to call methods against the web service client. The CEA web service is based on WS-Notification. After you generate the code needed for notification, you have to code the notify() method. The CEA feature includes a Web service sample; be sure to view the code and run the sample. You can also use an external Web service provider to use communications enabled applications. The external Web service provider's role is to manage all communications with the IP-PBX.

## References

- web service
  - <http://www.w3.org/TR/ws-arch/>
- SOAP
  - <http://www.w3.org/TR/soap/>
- WSDL
  - <http://www.w3.org/TR/wsdl>
- WS-Notification
  - [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsn](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn)
- JAX-WS
  - <http://www.jcp.org/en/jsr/detail?id=224>
- OASIS
  - <http://www.oasis-open.org/>

This slide lists some useful references.



## 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 email feedback:

[mailto:iea@us.ibm.com?subject=Feedback\\_about\\_WASv8\\_CEA\\_WebServicesDevelopment.ppt](mailto:iea@us.ibm.com?subject=Feedback_about_WASv8_CEA_WebServicesDevelopment.ppt)

This module is also available in PDF format at: [..WASv8\\_CEA\\_WebServicesDevelopment.pdf](..WASv8_CEA_WebServicesDevelopment.pdf)

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



## Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, Rational, and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

Java, and all Java-based trademarks and logos are trademarks of Oracle and/or its affiliates.

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

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2011. All rights reserved.