Understanding web services specifications, Part 7: WS-Business Process Execution Language

Hernan Silberman
Manas Mandal

May 10, 2007

You have a pretty robust system of services, but what you really want is an application that makes use of them to implement your business processes. Web Services Business Process Execution Language (WS-BPEL) provides you with a standard technology for composing and combining web services so they will do what you want at the macro level, and not just the micro level. This tutorial, the last of a seven-part series, explains how to create a process flow for your services using WS-BPEL.

View more content in this series

Before you start
This tutorial is for developers who build applications that use web services. With web services becoming increasingly popular, developers frequently have to create programs that coordinate the efforts of multiple web services to handle a business process. This article focuses on WS-BPEL, which is an important standard that provides a robust solution to this problem and has become a popular choice among developers.

In order to follow along with this tutorial, you should have a basic understanding of Simple Object Access Protocol (SOAP) and related technologies, such as WSDL. Read the first five tutorials in this series, especially Part 1 and Part 2, for the best understanding.

About this series
This tutorial series teaches the basic concepts of web services by following the exploits of the fictional newspaper, Daily Moon, as the staff uses web services to create a workflow system to increase productivity in this competitive environment.

Part 1 starts simply, explaining the basic concepts behind web services and showing you how to use SOAP, the specification that underlies most of what is to come, connecting the classifieds department with the Content Management System.

Part 2 takes things a step further, explaining how to use Web Services Description Language (WSDL) to define the messages that web services produce, enabling the team to more easily create services and the clients that connect to them.
Part 3 finds the team with a number of services in place and a desire to locate them easily. In response, Universal Description, Discovery, and Integration (UDDI) provides a searchable registry of available services as a way to publicize its own services to others.

In Part 4, Rudy, publisher of the Daily Moon, decides that the paper needs to institute better security procedures for web services that access their internal systems.

Part 5 shows the changes the teams needed to make in order to access those newly secured services.

Part 6 is about building and verifying interoperable web services. The newspaper staff members are already familiar with the importance of reaching the largest possible audience, so they decide to analyze their web services to ensure that anyone who wants to use them will have an easy time doing so.

This Part 7 introduces the Web Services Business Process Execution Language (WS-BPEL) and shows how to use it to create complex applications by coordinating the efforts of individual services.

About this tutorial
In this Part 7 of the web services tutorial series, we find the Daily Moon staff hard at work building an application that automates the process of receiving classified advertisements from their customers. The Daily Moon has a business relationship with a bank, and the staff would love to have an application that receives a classified advertisement and automatically deposits the payment into the bank. Fortunately, the bank that the staff works with has a web service that can be used to make such a deposit. The staff researches this problem and decides to try using WS-BPEL to mash up their internal classified advertisement web service with the bank's web service. With these two services, the staff can model the business process it has in mind and build an application to handle it.

This tutorial teaches you about building sophisticated applications that coordinate the efforts of existing web services. By composing web services in this way, you can quickly build applications of tremendous utility. web services are designed to be easy to find and easy to use; it is natural to want to link web services in order to build powerful new applications.

Prerequisites
The code used in this tutorial is not specific to any particular programming language or environment. The examples provided are the same ones used throughout this tutorial series. To follow along with the examples, you need the following software installed:

Java 2 Standard Edition version 1.4.2 or higher—All of these tools are Java-based, as are the services and clients you build in this tutorial.

Apache Axis2 version 1.0—Axis2 is a full-featured SOAP toolkit that provides implementations of several web service APIs including SOAP and WSDL. A toolkit like Axis2 is invaluable when it comes to web service development. Toolkits of similar scope exist for other programming
languages and environments. The Axis project at Apache has a long history and originated with an IBM effort called SOAP4J.

Apache Geronimo or another application server—This tutorial series uses the Apache Geronimo J2EE server throughout (which is the basis for IBM WebSphere® Community Edition server). You can use other application servers instead, but Geronimo is simple, lightweight, and freely available, so it is a good choice for getting up-and-running quickly.

BPWS4J version 2.1—IBM provides a BPEL runtime, which enables you to run processes that are written using WS-BPEL. BPWS4J is available for a 90-day trial period.

Overview
This tutorial begins with an overview of business processes and the evolution of technology that has become an important factor in executing them.

Understanding business computing today
Most business applications exist to automate or simplify processes that were once accomplished using ink, paper, and people. Accountants once did calculations by hand, recorded results in a ledger using pen and paper, and made routine trips to the bank. Brokers once juggled a similar set of tedious tasks each day, and so did bankers, actuaries, architects, and engineers. Computers continue to have a tremendous impact on business. The maturation and adoption of web services by many companies is the culmination of decades of technological advancement. Many fundamentally important organizations, like banks, are providing their partners with web services they can use to combine their businesses more closely and effectively than ever before.

The benefits of this close cooperation between organizations are evident each time you shop online. It is clear that the future of business computing is in connectivity. The businesses that make themselves available to their partners have a greater chance of succeeding as important components in the business processes of their partners. This notion is not new in business, but the evolution of software capabilities has made it possible to a degree only recently imaginable, and with immediacy that makes the past look tedious indeed.

Addressing disruptive technology
It is hard to keep up with advancements in computer technology. The world of software seems to change on a daily basis. Software developers need to know quite a bit to write modern web services, mostly because the underlying technologies change so quickly. The Internet is the latest disruptive technology to challenge every type of business.

The rapid pace at which technology advances helps highlight the common sense of modern web services, which are defined using language-neutral XML and focus on messages without assuming or prescribing much more. This approach insulates applications from the constant change Internet software developers have learned to expect.

Web services developers are able to hide implementations behind web service descriptions and change them as needed. Consumers of web services rely on web service descriptions to insulate
them from changes in service implementations. In most cases, you have no idea (or need to know) how a service you use is built.

As web services become more and more common, you will eventually find yourself both building services and using services in the same application. An online store is a great illustration of this now common scenario. If you are building a web service like the Daily Moon classified service, you receive information from an individual hoping to buy a classified advertisement. With this advertisement request comes payment information, which you hand off to your bank's web service for processing. It is common these days to build services by composing other useful services.

It would be great to have some way to describe applications that use existing web services to handle important business processes. It would be even better if this method of describing services was language neutral and extended to these applications the same benefits that web services provide their consumers and implementers.

This tutorial introduces the Business Process Execution Language for Web Services (WS-BPEL), which provides such a language-neutral facility for building new web services by composing existing web services.

**The story so far**

This series follows the staff of the fictional Daily Moon newspaper as it moves many of its everyday operations to a web services-based system. In Part 1, the Classifieds Department learned about SOAP by interacting with the Content Management System, and in Part 2, they created their own service, and defined it using Web Services Description Language (WSDL). Then in part 3, they learned how to interact with a UDDI registry, and also how to find data within one, which ultimately led them to create one for the company to enable other organizations to interact with the Daily Moon. Because Rudy insisted that his colleagues Gene and Francis provide a way to prevent unauthorized access to their systems in Part 4, Gene and Francis had to implement WS-Security for their web services. In Part 5, Rudy realized he needed to define policies for web services to enforce that clients accessing the web services of the Daily Moon do so in a prescribed manner, ensuring the security that was built into it back in Part 4 of this series. In Part 6, Phil makes sure the WSDL document and the classified web service it describes are interoperable with other web services.

Now, continuing the story of the Daily Moon staff, this tutorial describes the staff's attempt to build a web service that completely automates the process of purchasing a classified advertisement for inclusion in the newspaper. The tutorial introduces you to the WS-BPEL and the important concept of business process modeling.

**Interoperability overview**

Before going into the details of WS-BPEL, this article guides you to think in terms of business processes. Then this article introduces WS-BPEL and discusses its approach of composing web services to build new applications. You will see how WS-BPEL does this using the web services standards you have learned about over the course of this tutorial series using an example from the fictional Daily Moon newspaper.
**Understanding the goal**
Each time you do work, you are executing a process. Whether you are sorting your DVD collection, planting a new tree, or shipping a package, you can describe the process you go through by identifying the actors, their actions, and the artifacts you employ in the process.

In the business world, understanding process is important for many reasons. The first and most obvious is that understanding business process leads to insights that can improve those processes, potentially saving time and money. Business analysts are constantly scrutinizing business processes, looking for new efficiencies and cost savings. Another important reason for understanding business processes is that these processes are often important intellectual assets. Many companies succeed because of their superior *know-how* instead of any physical assets or products. It is important for these organizations to catalog their processes so that they can understand, improve, and protect them.

It is worthwhile to learn a little more about business processes and how they are modeled, as shown in the following sections.

**Understanding the history**
Thinking in terms of business processes is not a new idea. Businesses have always had to deal with logistics and think about how they do the things they do in order to stay in business and become more efficient. Businesses today focus much of their attention on how to describe the processes they employ, how they design new business processes, how they execute each process, and how they collect information about their business processes for later scrutiny.

Software has had a huge impact on the process-driven view of business. Most business software exists to automate processes that were once done manually. Software has advanced from the first spreadsheets and word processors to elaborate customer relationship management software and service-oriented architectures. As software technology has advanced, more and more business processes have become eligible for complete automation.

As web services have become widely adopted, it has been possible to think about automating business processes that span multiple web services. For example, a company might have an automated order processing system, a billing system, and a fulfillment system, all of which provide web service interfaces. With those web services in place, an application can be built to automate the interactions between them; for example, to automatically accept and fulfill an order from a customer. Web services have been an important technology for integrating enterprise applications in this way.

This same approach can be used to build applications that themselves rely on applications that different organizations own and operate. Today, web services provide a great platform for the latest generation of business process applications across multiple enterprises.

**Exploring business process management**
Today, business process management is the discipline of designing, deploying, and monitoring business processes using various standard technologies. It is not surprising that business process
management today has much to do with web services and the family of technologies discussed in this tutorial series.

One of the biggest advancements in the area of business process management was the invention of standard ways of describing business processes. Initially, if an organization built software to run business processes, it did so using Java or another programming language. There are now standard techniques for describing business processes. There are also runtime environments that execute business processes from these standardized descriptions.

The following sections describe the main activities involved in business process management and some of the tools that will make it a worthwhile approach to describing and executing important business logic.

Discovering and designing processes

As a software professional, surely you have been in a meeting where an expert did his best to describe how an application should work. Usually, he draws diagrams on a white board that identify all of the important actors, actions, and artifacts involved in a business process. These drawings are refined as developers ask questions and as imagined scenarios are played out.

Business process design is an iterative process, so it is important to have some way to write down a process and store it so that it can be called up again later for refinement. Business requirements change over time, so having a repository of business processes and standard tools for viewing and changing them is important. These tools exist. WS-BPEL is one important tool for codifying business processes so that they can be viewed, updated, and executed.

Executing process

Once business processes are defined, it is possible to build software systems that execute them. Business processes written in WS-BPEL become scripts that are fed into a business process management application and executed. There are huge advantages to this approach, the biggest of which is the simplicity with which you can create and update business processes. You see that WS-BPEL is a sort of scripting language for building applications from existing web services. The service descriptions themselves are executable scripts describing a process flow that a WS-BPEL runtime environment can execute.

Monitoring

After business processes are codified into scripts using a standard language and executed in a business process management application, they are incredibly easy to monitor. This monitoring can provide a wealth of reporting information about each business process instance that the application executed. For example, you could ask the process management application about the status of current processes it is working on and processes it has completed. You can use the information the business process application collects to easily answer many business questions you might have to otherwise work hard to find.
Introducing WS-BPEL

The idea of a formal and standard language for describing business processes has been an attractive one for a while now. Until recently, there were many competing standards that aimed to do just this, including Web Services Flow Language (WSFL) from IBM, XLANG from Microsoft, and many others. Eventually, WS-BPEL was created, which merged the best points from WSFL and XLANG. It emerged as the most popular standard in this category.

Overview

WS-BPEL depends on various standard technologies you are probably already familiar with, including:

- WSDL
- XML schema
- Xpath
- Web Services Addressing

WSDL is the most important of these standards because WS-BPEL describes business processes as conversations between web services as they are described in WSDL. In addition, WS-BPEL processes are web services WSDL describes.

Learning the language

WS-BPEL is an XML programming language. In WS-BPEL, a business process is described in an XML document that is the business process execution environment executes. A process in WS-BPEL is broken down into a series of steps called activities. The supported list of activities is listed below.

- invoke -- Invokes an operation on an existing web service
- receive -- Waits for a message from an external entity
- reply -- Generates a message to respond to an external entity
- wait -- Waits for a specified period of time
- assign -- Copies a value from a source to a destination
- throw -- Raises an error
- terminate -- Kills the current process unconditionally
- empty -- Offers a placeholder, which is a no-op

The basic activities can be combined to describe business process algorithms using an additional set of language keywords. These keywords provide the structural character of the WS-BPEL language and are listed below.

- sequence -- Defines an ordered sequence of actions
- switch -- Offers a selection statement that works like case in Java and C++
- while -- Defines a while loop
- pick -- Offers a selection statement that works like if
- flow -- Encloses a collection of steps that should be executed in parallel
With this small set of keywords, you can imagine how business processes are expressed in WS-BPEL. You can see that this concise syntax is quite powerful and easy to use.

**The process composition model**
As already mentioned, business processes can be described by the actors, actions, and artifacts they involve. A web service can be an actor in a business process. Often, multiple web services act together to satisfy a single business process.

**Defining processes as a service**
When you define a business process using WS-BPEL and feed it into a WS-BPEL runtime environment, your business process is exposed as a web service complete with its own WSDL document and ready-to-run implementation based on your blueprint. Your process definition probably prescribes one or more conversations with other web services and, effectively, WS-BPEL enables you to create a new web service by composing various partner services you had available. This is a powerful approach to building applications, especially when you consider that your resulting WS-BPEL web service might itself be used as a component by another business process.

**Starting with plain English**
As an example, look at the *Daily Moon*. Remember that the newspaper is trying to build a fully-automated service to handle its readers' creation of new classified advertisements. The *Daily Moon* staff has already built its own web service for creating classified advertisements. Now staff members have the WSDL from their bank describing the web service they can use to deposit payments for each classified advertisement their users create. They begin by describing their process in plain English.

- An end user application calls the *Daily Moon* with information about creating a classified advertisement and about paying for it.
- The provided payment information is submitted to the bank for processing.
  - If the bank approves the payment, the classified advertisement service is called upon to process the requested classified advertisement. After the classified advertisement has been created, a success message is sent to the caller.
  - If the bank denies the payment, no further processing happens. A message is sent to the caller informing him that the payment failed.

**Using WSDL**
Begin by building a WSDL that describes the endpoint that you will eventually expose and that end users will call to invoke your business process. To create this WSDL, begin by defining the different message types you will use for the Daily Moon classified process. For the example, define:

1. A message to be used to communicate with the classified service (classifiedMessage)
2. A message to be used to communicate with the bank service (bankMessage)
3. A message for the user to kick off the business process (classifiedInfoMessage)
4. A message to use in case an error is encountered during the processing of your business process.
Listing 1 shows the WSDL definition that includes these messages.

**Listing 1. WSDL with message definitions for the Daily Moon business process**

```xml
<definitions
targetNamespace="http://tempuri.org/services/classifieddefinitions"
 xmlns:tns="http://tempuri.org/services/classifieddefinitions"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="classifiedMessage">
    <part name="classifiedName" type="xsd:string"/>
    <part name="amount" type="xsd:int"/>
  </message>
  <message name="bankMessage">
    <part name="userName" type="xsd:string"/>
    <part name="amount" type="xsd:int"/>
  </message>
  <message name="classifiedInfoMessage">
    <part name="classifiedName" type="xsd:string"/>
    <part name="userName" type="xsd:string"/>
    <part name="amount" type="xsd:int"/>
  </message>
  <message name="classifiedErrorMessage">
    <part name="errorCode" type="xsd:int"/>
  </message>
</definitions>
```

Next, import this set of WSDL message definitions into another WSDL document, along with the definitions for the classified and banking services. The resulting WSDL is shown in Listing 2.

**Listing 2. Parent WSDL that imports the message definitions and the WSDLs for the classified and bank web services**

```xml
<definitions
 targetNamespace="http://tempuri.org/services/wsdl/classified-approval"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:lns="http://tempuri.org/services/wsdl/classified-approval"
 xmlns:asns="http://tempuri.org/services/classifiedservice"
 xmlns:apns="http://tempuri.org/services/bankservice"
 xmlns:loandef="http://tempuri.org/services/classifieddefinitions">
  <import namespace="http://tempuri.org/services/classifiedservice"
 location="http://localhost:8080/myservices/service/classifiedservice.wsdl"/>
  <import namespace="http://tempuri.org/services/bankservice"
 location="http://localhost:8080/myservices/service/bankservice.wsdl"/>
  <import namespace="http://tempuri.org/services/classifieddefinitions"
 location="http://localhost:8080/myservices/service/classifieddefinitions.wsdl"/>
  <plnk:partnerLinkType name="bankLinkType">
    <plnk:role name="bank">
      <plnk:portType name="apns:bankPT"/>
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```
In WS-BPEL, any web services involved in a process are referred to as **partner** web services. You can see in Listing 2 the WSDL partnerLinkType definitions to identify two partners: **bank** and **classified**, and how to link them to their respective port types as defined in their WSDL documents. The last thing in Listing 2 is to come up with a service name for our business process.

### Defining the BPEL

Now that you have your WSDL in place, move on to your BPEL definition. The business process you are implementing describes the coordination of two existing web services. Nothing too fancy is going on beyond calling each of the services involved in a particular order and passing around the messages they produce. Now, use your favorite programming language to implement this flow of logic. After making it this far in this tutorial series, you know how easy it would be to write this up as a Java program.

WS-BPEL provides an attractive option to implement your business process algorithms. WS-BPEL plays the role of a scripting language for coordinating a series of web service calls. With a small amount of effort, you can not only script the appropriate web service calls, but you can expose the resulting script as a web service itself. Also, as an XML language, WS-BPEL process definitions are easy to change, much more concise, and easier to follow than the programming code it would take to do the same thing.

### Structuring the document

A BPEL process definition is defined within a `<process>` element. Begin by adding this element to your file, as shown in Listing 3.

#### Listing 3. The main element of a BPEL description

```xml
<process name="bookClassifiedProcess">
<!-- Definition goes here -->
</process>
```

There are three main sections within the process element.

### Creating the first section of the BPEL definition

The first section lists the partners that play a part in the business process. For the *Daily Moon* example, these are:

- The classified advertisement web service
- The bank web service
- The end user that kicks off the business process
Listing 4 shows the relevant partner declarations for the *Daily Moon* business process. Namespace definitions have been left out for the sake of readability.

**Listing 4. Process description with partner links**

```
<partnerLinks>
  <partnerLink name="customer"
    partnerLinkType="lns:bankLinkType"
    myRole="bank"/>
  <partnerLink name="bank"
    partnerLinkType="lns:bankLinkType"
    partnerRole="bank"/>
  <partnerLink name="classified"
    partnerLinkType="lns:bookLinkType"
    partnerRole="classified"/>
</partnerLinks>
```

The first partner corresponds to the customer that initiates this business process. The second partner is the bank web service. The third partner is the classified web service.

**Creating the second section of the BPEL definition**

The second major section of the BPEL definition creates a few placeholders for the messages that are generated and passed among the partners during the execution of the business process. For the *Daily Moon* example, the messages involved are:

- The original message that kicks off the business process, which is referred to as the *request* message.
- The message that will be sent to the bank service to process the payment, which is referred to as the *bankMessage*.
- The message that the bank service responds with, which is referred to as the *bankapprovalInfo*.
- The message to (potentially) send to the classified service, which is referred to as the *classifiedMessage*.
- The message the classified service responds with, which is referred to as the *approvalInfo* message.
- The error message if an error is encountered during the execution of the business process, which is referred to as the *error*.

Listing 5 shows the variable definitions in the BPEL description. In each case, provide a symbolic name and a WSDL message type.

**Listing 5. Variable definitions for holding on to messages involved in the business process**

```
<variables>
  <variable name="request"
    messageType="classifieddef:classifiedInfoMessage"/>
  <variable name="classifiedMessage"
    messageType="classifieddef:classifiedMessage"/>
  <variable name="bankMessage"
    messageType="classifieddef:bankMessage"/>
  <variable name="approvalInfo"
    messageType="apns:approvalMessage"/>
  <variable name="bankapprovalInfo"
    messageType="asns:approvalMessage"/>
  <variable name="error"
    messageType="classifieddef:classifiedErrorMessage"/>
</variables>
```
Creating the third section of the BPEL definition

The third major section in the BPEL definition describes the business process algorithm. This algorithm is expressed using the activities and structural components of the BPEL language that have already been presented.

Translating the business process algorithm is straightforward. You are going to perform a sequence of steps in a specific order, so begin with a sequence element as shown in Listing 6. Most business processes are modeled as sequences.

Listing 6. Business processes modeled as sequences of activities

```xml
<sequence>
    <!-- Do something interesting -->
</sequence>
```

The first activity according to your algorithm is to accept or receive a request from the customer. Use a `<receive>` element to express this, as shown in Listing 7.

Listing 7. Sequence showing the receiving of a customer message

```xml
<sequence>
    <receive name="receive1" partnerLink="customer" portType="asns:bankPT" operation="depositToBank" variable="request" createInstance="yes">
    <!-- Do something interesting -->
    </receive>
    <!-- Do something interesting -->
</sequence>
```

The important parts of this receive action establish which partner you are receiving from and which variable to store the received message in. Essentially, this statement says "wait for depositToBank to be invoked on port type BankPT, and, when it happens, store the message in the variable request".

The next thing you need to do is copy information from the request message into a few of the other messages you are going to use. Listing 8 shows a series of assign activities that accomplish this.

Listing 8. Assign activities copying state from one message to another

```xml
<assign name="assignclassified">
    <copy>
        <from variable="request" part="classifiedName"/>
        <to variable="classifiedMessage" part="classifiedName"/>
    </copy>
    <copy>
        <from variable="request" part="amount"/>
        <to variable="classifiedMessage" part="amount"/>
    </copy>
</assign>

<assign name="assignbank">
    <copy>
```
The assign actions above prepare the classifiedMessage and bankMessage by copying over the classifiedName, amount, and userName from the initial request.

You are now ready to invoke the bank web service to process the payment from the customer. Listing 9 shows the definition of an invoke activity on the bank web service.

**Listing 9. An invoke activity prescribing a call to the bank web service**

```xml
<invoke name="invokeBank"
     partnerLink="bank"
     portType="asns:bankPT"
     operation="depositToBank"
     inputVariable="bankMessage"
     outputVariable="bankapprovalInfo">
</invoke>
```

This is easy to follow. It simply sends bankMessage to bankPT and invokes the depositToBank operation, storing the response in bankapprovalInfo.

The next step depends on the result of the above invokeBank action. We use a switch to handle the conditional logic and a special BPEL function called bpws:getVariableData to check the status of the bank approval message. Listing 10 shows the switch statement covering the possible outcomes.

**Listing 10. A switch statement to handle the success or failure of the bank transaction**

```xml
<switch>
  <case condition="bpws:getVariableData('bankapprovalInfo','accept')=1">
    <invoke name="invokeClassified"
        partnerLink="classified"
        portType="apns:classifiedPT"
        operation="bookClassified"
        inputVariable="classifiedMessage"
        outputVariable="approvalInfo">
  </invoke>
  </case>
  <otherwise>
    <assign name="assignMessage">
      <copy>
        <from expression="'Bank payment failed'"/>
        <to variable="approvalInfo" part="accept"/>
      </copy>
    </assign>
  </otherwise>
</switch>
```

Note that the switch statement in Listing 10 sets an error string on the approvalInfo message if the bank process did not succeed.
All that remains is to respond to the customer. That is easy enough, as shown in Listing 11. Being able to respond to the customer is one reason why the customer is listed as a partner. More complicated usage scenarios involving callbacks are another important reason for listing the initiator of a process as a partner.

**Listing 11. Reply activity, sending a reply to the customer upon completion of the business process**

```xml
<reply name="reply"
   partnerLink="customer"
   portType="asns:bankPT"
   operation="depositToBank"
   variable="approvalInfo"/>
```

The only additional item is a fault handler, which demonstrates how BPEL can react when a web service call produces a fault. Listing 12 shows the definition of a fault handler to handle faults thrown by the classified service. If such a fault is encountered, the handler responds to the customer immediately with an error message.

**Listing 12. Definition of a fault handler, prescribing termination of process execution for a fault**

```xml
<faultHandlers>
   <catch faultName="classifiedFault"
       faultVariable="error">
      <reply partnerLink="customer"
         portType="apns:bankPT"
         operation="depositToBank"
         variable="error"
         faultName="invalidRequest"/>
   </catch>
</faultHandlers>
```

Listing 13 shows the complete BPEL file implementing the business process for the *Daily Moon* to automatically receive a classified advertisement and charge a customer for it using their in-house classified web service and their bank’s provided web service.

**Listing 13. The complete BPEL process definition**

```xml
<process name="bookClassifiedProcess"
  targetNamespace="http://tempuri.org/classifiedProcess"
  suppressJoinFailure="yes"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
  xmlns:lns="http://tempuri.org/services/wsdl/classified-approval"
  xmlns:classifieddef="http://tempuri.org/services/classifieddefinitions"
  xmlns:asns="http://tempuri.org/services/bankservice"
  xmlns:apns="http://tempuri.org/services/classifiedservice">

  <variables>
    <variable name="request" messageType="classifieddef:classifiedInfoMessage"/>
    <variable name="classifiedMessage" messageType="classifieddef:classifiedMessage"/>
    <variable name="bankMessage" messageType="classifieddef:bankMessage"/>
    <variable name="approvalInfo" messageType="apns:approvalMessage"/>
    <variable name="bankapprovalInfo" messageType="asns:approvalMessage"/>
    <variable name="error" messageType="classifieddef:classifiedErrorMessage"/>
  </variables>
```
<variables>
</variables>

<partnerLinks>
  <partnerLink name="customer" partnerLinkType="lns:bankLinkType" myRole="bank"/>
  <partnerLink name="bank" partnerLinkType="lns:bankLinkType" partnerRole="bank"/>
  <partnerLink name="classified" partnerLinkType="lns:bookLinkType" partnerRole="classified"/>
</partnerLinks>

<faultHandlers>
  <catch faultName="classifiedFault" faultVariable="error">
    <reply partnerLink="customer" portType="apns:bankPT" operation="depositToBank" variable="error" faultName="invalidRequest"/>
  </catch>
</faultHandlers>

<sequence>
  <receive name="receive1" partnerLink="customer" portType="asns:bankPT" operation="depositToBank" variable="request" createInstance="yes">
  </receive>

  <assign name="assignclassified">
    <copy>
      <from variable="request" part="classifiedName"/>
      <to variable="classifiedMessage" part="classifiedName"/>
    </copy>
  </assign>

  <assign name="assignbank">
    <copy>
      <from variable="request" part="userName"/>
      <to variable="bankMessage" part="userName"/>
    </copy>
  </assign>

  <invoke name="invokeBank" partnerLink="bank" portType="asns:bankPT" operation="depositToBank" inputVariable="bankMessage" outputVariable="bankapprovalInfo"/>

  <switch>
    <case condition="bpws:getVariableData('bankapprovalInfo','accept')=1">
      <invoke name="invokeClassified" partnerLink="classified" portType="apns:classifiedPT" operation="bookClassified" inputVariable="classifiedMessage"/>
    </case>
  </switch>
</sequence>
Putting it together

There are many different applications involved in the execution of a typical WS-BPEL business process. To see an example of the *Daily Moon* WS-BPEL in action, for example, have the following pieces running:

- The classified advertisement web service
- The bank's web service
- A WS-BPEL runtime environment that will run the WS-BPEL process

Setting it up

1. Run both web services and your BPEL runtime environment on the same J2EE server.
2. Download Geronimo and install it. The example uses the Geronimo server from Apache, which is freely available and simple to set up.
3. Download the IBM Business Process Execution Language for Web Services Java Run Time (or BPWS4J). BPWS4J is fully-featured platform for running business processes described in BPEL. Download version 2.1 of the bpws4j engine.
4. Copy bpws4j.war from the BPWS4J distribution, and drop it into the deploy directory of your Geronimo server. Geronimo notices this new Web application and automatically deploys it. You should now be able to see the BPWS4J administration console, as is shown in Figure 1 using the URL http://localhost:8080/bpws4j/admin.
As the big buttons show, this administrative application can be used to deploy, list, and remove WS-BPEL business processes from the BPWS4J runtime environment.

Before you deploy your business process, deploy your two web services: the classified web service and the bank web service. This tutorial comes with a code bundle that includes both of these web services bundled into a single file called `myservices.war`. Make a copy of this bundle, and drop it into Geronimo's deploy directory. Geronimo notices this new bundle and deploys both of the web services.

### Deploying a business process

The details of deploying a business process differ from one WS-BPEL environment to another, but they should all be fairly simple. Deploying a business process on BPWS4J takes a few mouse clicks.

1. Click the Deploy button in the BPWS4J administrative Web application. You are asked for the WSDL and BPEL documents you created that define an endpoint for your business process and the business process itself, respectively. This is shown in Figure 2.
2. Browse to the bookclassified.wsdl and bookclassified.bpel files, and select Continue Deployment. You are asked for the WSDLs for each partner service involved in the business process, which, of course, are your classified and bank services. This is shown in Figure 3.

**Figure 3. Providing WSDLs for all partner web services**

3. Find the WSDLs, and select Start Service the Process. BPWS4J attempts to begin serve your business process. Figure 4 shows the successful startup of the *Daily Moon* business process.
Calling the business process

The next step is to test your service by calling it. Of course, this means making a simple SOAP call to the service described in the WSDL you created for your service. This article includes a Java program to make this call and a simple batch script you can use to test it with. To make this test easy, your bank service accepts any payments over 30 and declines anything else. Run the business process once with a value of 40, and expect it to succeed. Run it once with a value of 20, and expect it to fail. The results are shown in Listing 14.

Listing 14. Calling your business process

```
C:\bpel\client\testcustomer.cmd http://localhost:8080/bpws4j/axisengine
classifiedName userName 40
success
C:\bpel\client\testcustomer.cmd http://localhost:8080/bpws4j/axisengine
classifiedName userName 20
Bank payment failed
```

Summary

This tutorial has introduced you to WS-BPEL and the concepts at work in business process management. In a world full of web services, it makes sense to have a standard technology for scripting interactions between them, to compose them into even more powerful web services that support important business processes.

WS-BPEL is an example of what is possible with standard technologies like those presented in this tutorial series. These standards give application developers a powerful set of tools to integrate the applications of an enterprise business and to build the newest breed of applications that span multiple enterprises.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 7 BPEL source code</td>
<td>wsbpelcode.zip</td>
<td>1874KB</td>
</tr>
</tbody>
</table>
Related topics

- **RSS feed for this series**: Request notification for the upcoming articles in this series. (Find out more about RSS feeds of developerWorks content.)
- See **Business processes in a web services world** (developerWorks, August 2002) for an overview of BPEL4WS, including the basic language elements and the concepts underlying the language.
- See **Business Process with BPEL4WS: Learning BPEL4WS** (developerWorks, August 2002) for a series of articles that gives readers an understanding of the different components of the BPELWS language, and teaches them how to create their own complete processes.
- See **Transactions in the world of web services** (developerWorks, August 2002) for a series of articles that provides a high-level overview of the web service specifications for WS-Coordination and WS-Transaction.
- See **Automating business processes and transactions in web services** (developerWorks, August 2002) for an introduction to BPELWS, WS-Coordination, and WS-Transaction.
- See **Describe business process activities as web services** for a series of short articles that introduce readers to the industry’s various web services standards.
- Learn more about WS-BPEL using the **WS-BPEL 2.0 Tutorial**.
- Read a **WS-BPEL Tutorial** from Eclipse.org.
- Visit the **JBoss WS-BPEL wiki**.
- Go to **Wikipedia BPEL Wiki** to get a succinct summary of BPEL.
- Read the **Introduction to XML** tutorial (developerWorks, August 2002) to get a good grounding in XML.
- Read the **Online banking with Apache Geronimo and Axis2 tutorial series** to see an example of Axis2 at work.
- Check out the **developerWorks Java zone** to learn more about programming in Java.
- Learn all about XML at the **developerWorks XML zone**.
- Download **Java 2 Standard Edition version 1.4.2 or higher**.
- **The IBM SOA Web site** offers an overview of SOA and how IBM can help you to get there.

© Copyright IBM Corporation 2007
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)