Boost application development with Amazon Web Services, Part 1: How to use the Amazon E-Commerce Service

Skill Level: Intermediate

Eric Giguerre (ericgiguere@ericgiguere.com)
Software Developer
Studio B

24 Jun 2005

This tutorial provides an overview of Amazon Web Services (AWS). AWS exposes raw product information and key parts of Amazon.com technology to third-party developers for use in their applications. After describing how AWS works in general, the tutorial focuses on the main AWS service, called the Amazon E-Commerce Service (ECS). As part of this tutorial, you will develop a small Web application that uses ECS to display book and music information.

Section 1. Before you start

About this tutorial

This tutorial provides an overview of Amazon Web Services (AWS). AWS exposes raw product information and key parts of Amazon.com technology to third-party developers for use in their applications. After describing how AWS works in general, the tutorial focuses on the main AWS service, called the *Amazon E-Commerce Service* (ECS). As part of this tutorial, you will develop a small Web application that uses ECS to display book and music information.

This tutorial is written primarily for Java(TM); developers who are looking to incorporate AWS into their applications or to build new applications around AWS, though much of the material isn't specific to the Java programming language. No

knowledge of Web services is required.

Prerequisites

As part of this tutorial, you need to register with Amazon.com to obtain an AWS subscription ID. Registration is free, and the details are provided later. You cannot use any of Amazon's Web services without a subscription ID.

To run the code in this tutorial, you need a Java software development kit (JDK), version 1.4.2 or later.

Section 2. Introducing Amazon Web Services

Expose the data

A company's databases of information have traditionally been considered proprietary data meant to be kept safe from prying eyes. But large e-commerce companies like Amazon.com, eBay, and Google are taking the opposite view and *opening up* their data repositories to interested outsiders. A recent article in *Business 2.0* described this trend as "the great giveaway." What would entice these large companies to do such a thing? What do online businesses have to gain from exposing their critical information to others? Why would they release competitive information for all to see?

The people in charge of these companies know what they're doing, of course, and do not give outsiders wholesale access to their entire databases. Privacy and confidentiality issues aside, too much information resides in those databases that would be of interest to competitors. As such, nobody outside each company can access the raw data. Instead, access to specific parts of their databases is tightly controlled using Web services.

But why would these companies expose any of their data through Web services? The fact is, these companies have *too much* information in their databases. Despite its size, a company like Amazon.com must still devote most of its resources toward its core business -- the business of selling things online. The company can't afford to mine each last bit of information from its databases.

What these companies are really doing is exposing *platforms* for developers to

create new and compelling applications that extend the core functionality of an online business. They hope that developers will use these platforms to partner with them and drive more business their way in a symbiotic relationship that benefits both parties.

Amazon Web Services

When it was first introduced in July 2002, AWS was a simple way for developers to access a limited amount of product information contained within the Amazon databases. Since then, AWS has expanded into three separate service areas:

- Amazon E-Commerce Service (ECS)

- Alexa Web Information Service (AWIS)

- Amazon Simple Queue Service (ASQS)

ECS is the longest-running service. For a long time, it was known simply as *AWS,* because the other two services are relatively recent introductions. In fact, both AWIS and ASQS are still in beta.

Amazon E-Commerce Service

ECS peers into Amazon's product database and lets developers extract detailed information about any item sold by Amazon or through third-party sellers who are using the Amazon site as their online storefront. Developers can even use ECS to build electronic shopping carts and to transfer the items in those shopping carts to the Amazon.com Web site for payment.

As previously mentioned, the ECS had been known as *AWS.* As such, ECS is currently at version 4.0. Previous versions are referred to as *AWS 1.0, AWS 2.0,* and *AWS 3.0.* The switch from AWS 3.0 to ECS 4.0 was more than a name change, however, because Amazon took the opportunity to rework many of the Web services it was exposing. AWS 3.0 and ECS 4.0 currently co-exist, but the use of AWS 3.0 has been deprecated.

Note that ECS 4.0 exposes information in product databases from six Amazon sites:

- Amazon.com (United States)

- Amazon.de (Germany)

- Amazon.ca (Canada)

- Amazon.fr (France)

- Amazon.co.uk (United Kingdom)

- Amazon.co.jp (Japan)

Developers can interact with the database that best suits their needs.

The information available through ECS 4.0 goes beyond simple product information, however. Amazon features such as wish lists and customer reviews are also accessible.

Access to ECS 4.0 is free to anyone who registers for an AWS subscription ID and who agrees to the terms of the AWS licensing agreement (the main restriction is that you cannot invoke any Amazon Web Service more than once per second).

Alexa Web Information Service

Although best known for running an online store, Amazon.com also runs Alexa Internet, a portal for navigating and searching the Web that melds Google and Amazon data and technology. Alexa's claim-to-fame is collecting statistics on Internet usage and traffic, primarily by monitoring browser activity through the Alexa Toolbar, a browser extension.

AWIS lets developers query some of the information that Alexa collects when it crawls Web sites and monitors browser activity, such as:

- Basic site information (who owns the domain, how does the site rank)

- Who links to a site (other Web sites)

- Related sites

Developers can also use AWIS to search the Web and to explore the sites listed in the Open Directory, a free Web directory that categorizes sites by topic and serves as the basis for a number of other directories run by Google and other companies.

Access to AWIS is free during the beta period, though developers are restricted to no more than 10,000 requests per day. Pricing for access after the beta period has yet to be determined.

Amazon Simple Queue Service

While ECS and AWIS provide access to Amazon and to information about specific Web sites, ASQS is a bit different. Instead of providing access to Amazon.com's *databases,* ASQS lets you build applications that piggyback on Amazon.com's *technology.*

As you can probably tell from its name, ASQS is a queuing system. A *queuing system* lets applications use a messaging paradigm: Instead of talking directly to each other, applications leave*messages* (data) for each other in simple databases called *queues.* In this case, the queues run on Amazon's own computers and emphasize reliability and scalability. The queues are exposed using Web services.

ASQS is a specialized service. Like AWIS, access to ASQS is free during its beta period, and limits are placed on how much data can be stored in a queue at any given time. Pricing for access after the beta period has yet to be determined.

Section 3. Work with AWS

The subscription ID

Even though AWS is free, you can't use the services without first obtaining an AWS subscription ID. You can obtain the ID using the subscription form on the Amazon.com Web site. Each subscription ID is associated with a single e-mail address. If you already have an Amazon account (for making book purchases, for example), you can attach a subscription ID to that account or create a separate, new account just for using AWS.

Before continuing with the tutorial, register with Amazon. You won't be able to run any of the code without a valid subscription ID.

After registering, Amazon sends your new subscription ID to the e-mail address you supplied. Save this e-mail message: You'll need to cut-and-paste the subscription ID later into Java source files. Be sure to read the AWS licensing agreement before continuing.

The Associate tag

You'll encounter references in documentation and in code samples to Amazon's

Associates program. The Associates program is an affiliate program that Web developers can join to make money from referring customers to Amazon.com. The main Associates page describes the program in detail.

When you join the Associates program, you're assigned another identifier, separate from the AWS subscription ID. This *associates tag* can be included in many AWS calls. If someone ends up buying something at Amazon.com through your application, the Associates tag ensures that Amazon credits you correctly for the sale. Note, however, that an Associates tag is *not* required to use AWS.

AWS documentation

Amazon provides extensive documentation for each of its AWS services from its online documentation page. Separate information is provided for each service, including documentation for both ECS 3.0 and ECS 4.0.

Bookmark the ECS 4.0 Developer Guide for later reference. The documentation is also available in PDF format for printing, but be warned that it's more than 450 pages long! Several third-party books also cover AWS; see Resources for titles.

AWS samples and discussion boards

A variety of code samples are available for download from the AWS Web Services Downloads page. Java programmers should download the **Java SOAP and REST** sample -- a self-contained example that demonstrates how to use ECS 4.0. You'll use this sample later in this tutorial. Samples for other programming languages are also available, but this tutorial deals only with the Java language.

For further help, Amazon sponsors a discussion board for AWS users to share information with each other and to communicate with the AWS development team.

Section 4. Dive into ECS

Understanding ECS

ECS is the Amazon service that most programmers use. It has great appeal for

developers for several reasons:

- It provides access to the vast amount of information contained in Amazon's product databases.

- It can be used as an opportunity to make money through the Amazon Affiliates program.

- Access to ECS is free of charge.

Note that ECS is a *read-only* service: You can't submit information to Amazon using ECS. Instead, you must use alternate channels and APIs to add or modify product listings, customer reviews, and so on. The only exception is that you can create and fill shopping carts for users and submit those carts to Amazon for presentment and approval by the user.

ECS can be used by stand-alone Java applications or by Web applications. The *ECS Developer Guide* lists a variety of possible uses for ECS:

- Presenting detailed information about Amazon products on your own site, including up-to-date prices

- Monitoring statistics for how well products are doing or how competitors are pricing their products

- Building a niche online store using Amazon's own technology

- Developing software products and services for other ECS users

Most of these uses are commercial in nature, but it doesn't have to be that way. If you're building software to organize book or music collections, for example, you can quickly download cover art for a given book or album using a few simple ECS calls. (Note: Be sure to read the AWS licensing terms closely for restrictions on using cover art and other copyrighted materials that Amazon provides.)

ECS is also a great way to experiment with Web services in general, because the services that AWS provides are easily understood (everyone can relate to browsing and purchasing books and menus) and are free to use.

Access ECS: REST versus SOAP

You can access all services available through ECS in one of two ways:

- As Representational State Transfer (REST) requests

- As SOAP requests

The first way -- through REST -- is the most simple to understand. REST uses HTTP as its underlying transport. You pass REST commands and options to ECS by using URL query parameters; ECS returns an XML document containing the result. This process makes it possible to invoke ECS services directly from a Web browser, as in this example:

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService&SubscriptionId=
                                xxxx&Operation=ItemLookup&ItemId=0321321146
```

The result is an XML document describing the book with ISBN 0321321146:

```
<?xml version="1.0" encoding="UTF-8"?>
<ItemLookupResponse
        xmlns="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
  <OperationRequest>
    <HTTPHeaders>
      <Header Name="UserAgent" Value="Mozilla/5.0 Gecko/20041001 Firefox/0.10.1"/>
    </HTTPHeaders>
    <RequestId>0PJ4DG95DYN2JGQ7ZSYE</RequestId>
    <Arguments>
      <Argument Name="Service" Value="AWSECommerceService"/>
      <Argument Name="Style" Value="http://www.ericgiguere.com/tohtml.xsl"/>
      <Argument Name="SubscriptionId" Value="xxxx"/>
      <Argument Name="ItemId" Value="0321321146"/>
      <Argument Name="Operation" Value="ItemLookup"/>
    </Arguments>
    <RequestProcessingTime>0.0185689926147461</RequestProcessingTime>
  </OperationRequest>
  <Items>
    <Request>
      <IsValid>True</IsValid>
      <ItemLookupRequest>
        <ItemId>0321321146</ItemId>
      </ItemLookupRequest>
    </Request>
    <Item>
      <ASIN>0321321146</ASIN>
      <DetailPageURL>http://www.amazon.com/exec/obidos/redirect?tag=ws%26link_code=xm2
      %26camp=2025%26creative=165953%26path=
      http://www.amazon.com/gp/redirect.html%253fASIN=0321321146
      %2526location=
      /o/ASIN/0321321146%25253FSubscriptionId=1WCK32SN05PGBJ0YV302</DetailPageURL>
      <ItemAttributes>
        <Author>Eric Giguere</Author>
        <ProductGroup>Book</ProductGroup>
        <Title>Make Easy Money with Google : Using the AdSense Advertising Program</Title>
      </ItemAttributes>
    </Item>
  </Items>
</ItemLookupResponse>
```

You can even ask ECS to apply an Extensible Stylesheet Language Transformation (XSLT) stylesheet to the XML it returns, which is useful for transforming the Amazon

data into a different format.

The more conventional approach to Web services is to use SOAP. For each of its Web sites, Amazon supplies Web Services Description Language (WSDL) documents that you use to import the ECS services into whatever Web services toolkit you're using.

Which approach you take -- REST or SOAP -- is entirely up to you. Amazon supports both systems equally well.

Operations

Two general types of operation are available to ECS: `lookups` and `searches`.

You use a `lookup` to find detailed information about things like products, merchants, and even customers. When looking up products, you can request information for up to 10 products. The previous example looked up information for a single book by its ISBN using the `ItemLookup` request. (Products are identified in a number of ways, chiefly by the Amazon Standard Identification Number (ASIN). For books, ASIN and ISBN values are equivalent.) The following `lookup` operations are available in ECS 4.0:

- `BrowseNodeLookup`

- `CustomerContentLookup`

- `ItemLookup`

- `ListLookup`

- `SellerLookup`

- `SellerListingLookup`

- `SimilarityLookup`

- `TransactionLookup`

As you might guess, `ItemLookup` is the most common `lookup` operation.

A `search` operation searches the Amazon databases for information using criteria you supply. For example, you can get a list of all the books in the Amazon database written by a specific author. The following `search` operations are available in ECS 4.0:

How to use the Amazon E-Commerce Service

- `CustomerContentSearch`

- `ItemSearch`

- `ListSearch`

- `SellerListingSearch`

The shopping cart operations are:

- `CartAdd`

- `CartClear`

- `CartCreate`

- `CartGet`

- `CartModify`

Finally, you can use a `Help` operation to obtain help for other operations.

Options control how much information an operation returns and how that information is returned. As already mentioned, you can apply an XSLT stylesheet to the XML (for REST requests only) simply by passing the URL of the stylesheet (which must be available on a public Internet site) to Amazon as part of the request. You can use *response groups* to control the level of detail returned. Response groups tell AWS which subset of the possible data to return -- for example, do you just want basic information about a book (title, author, ranking, price) or do you want detailed information (customer reviews, availability from third-party stores, images).

Content caching

To ensure that its systems are available for everyone's use, Amazon limits how quickly you can make successive requests to AWS. Each IP address is limited to one request per second. If your application is going to make several requests in a short span of time, you'll have to spread them out over several seconds, which can lead to unacceptable performance, especially if the Amazon servers are very busy. Thus, caching becomes an important part of any AWS-based application.

When making REST requests, for example, you can use a simple cache built around the `java.util.HashMap` class, where the URL of the request acts as the key. Every time you make a successful AWS request, place the response data (ideally already transformed into whatever format makes sense for your application so that you don't have to keep re-parsing the XML) in the cache.

Note that Amazon restricts the amount of time you can cache information. According to the AWS licensing agreement, most data can be cached for no more than 24 hours, though some of the static product information -- things like author names, product names, and release dates -- can be cached for up to three months. Pricing and availability information can be cached for up to a week, but if you don't refresh the information on an hourly basis, you must include a timestamp with the information whenever it's displayed. Refer to Section D of the AWS licensing agreement for all the specifics.

Section 5. Try out AWS

Run the sample application

The sample Java application for AWS (which is unnamed, so I refer to it as the *ECS Explorer*) is a simple tool for exploring AWS operations. You can invoke each possible operation directly from the tool using either REST or SOAP. To use the tool, unzip the `ws-aws1code.zip` file that accompanies this tutorial (see Downloads). This file creates a `TopRank` directory containing the source, libraries, and a precompiled version of the tool. There is no Ant build file, unfortunately, so if you make changes, you need to follow the instructions in the README.txt file for recompiling the source.

To run the ECS Explorer, use the following command in the `TopRank` directory:

```
java -jar TopRank.jar
```

Java 1.4.2 or higher is required to run the tool. When run, the tool opens a single window, as shown in Figure 1.

Figure 1. The ECS Explorer sample Java tool
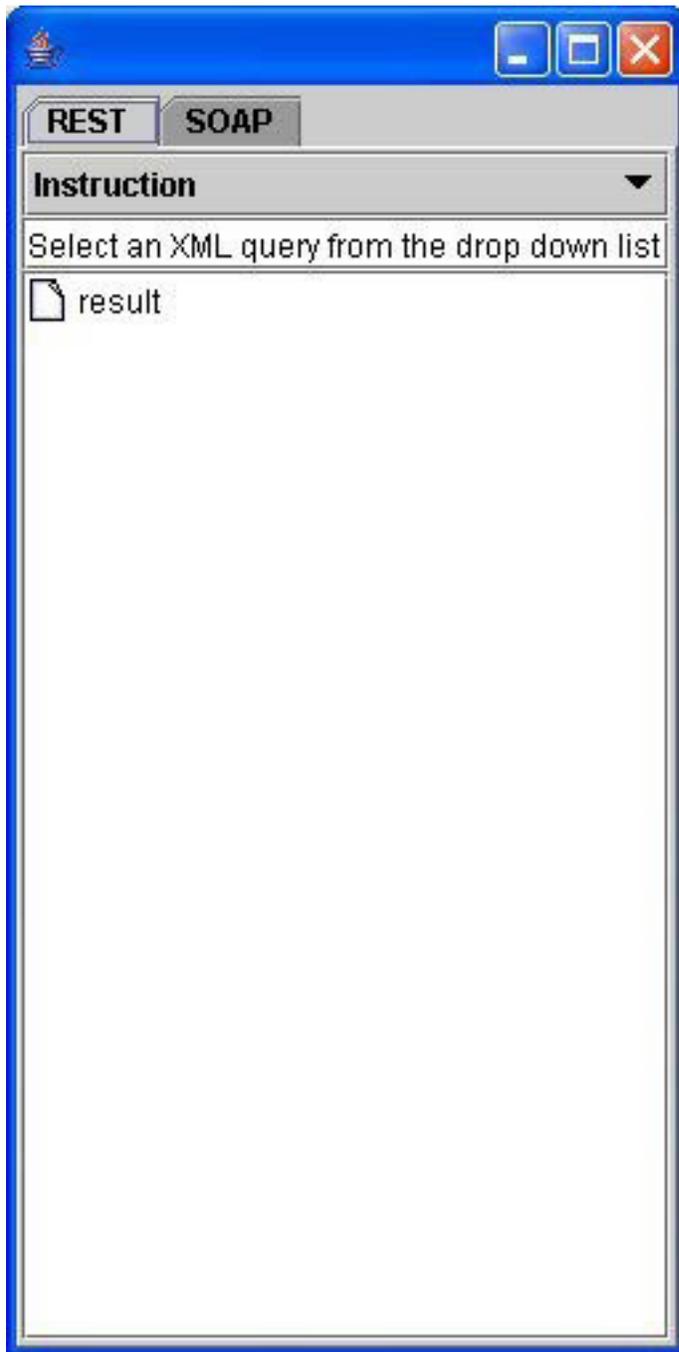
The tool is simple to use. The tabs switch between using REST or SOAP to make the requests, and the drop-down list immediately below the tabs lets you select the operation to perform, as shown in Figure 2.

Figure 2. Selecting the operation
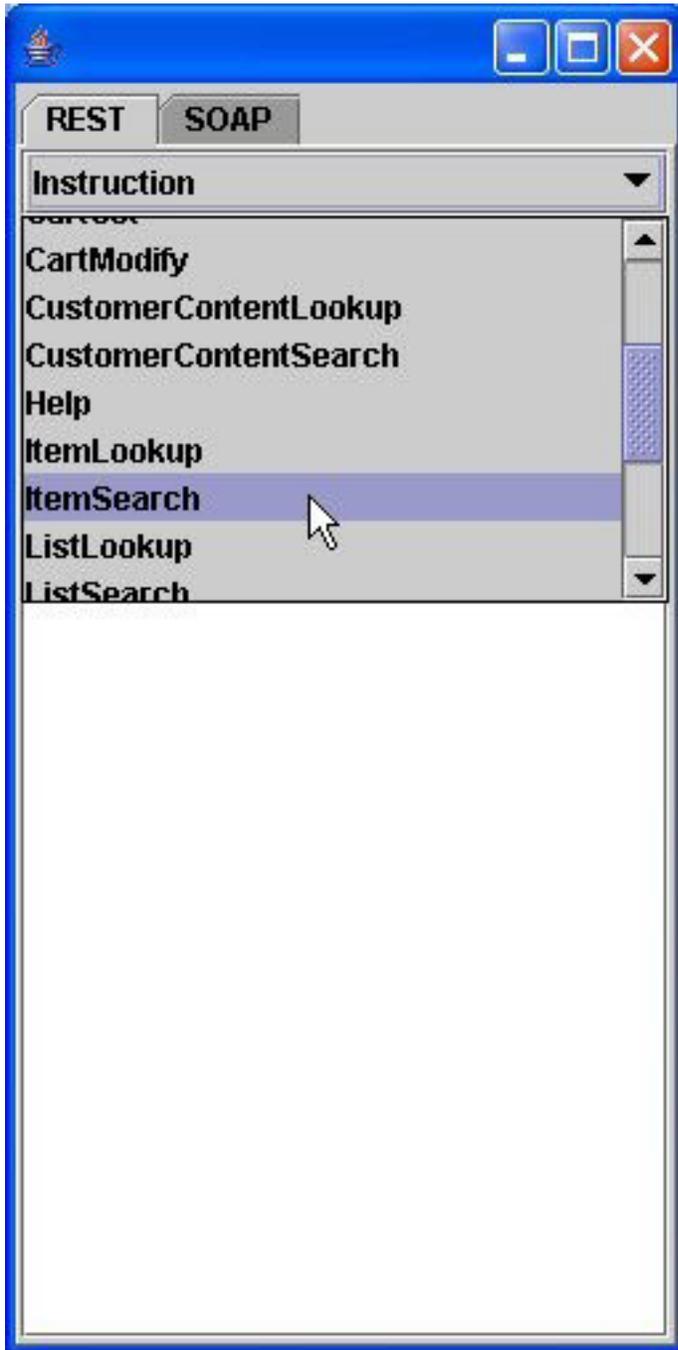
When you select an item in the list, the top part of the tool window changes to show the properties that you can set for the request. The common properties -- `SubscriptionID`, `AssociateTag`, and `Validate` -- are always shown first, followed by a **Send** button and the operation-specific properties (you might want to

expand the application window to see all the properties). You fill in the appropriate property values (note that the `SubscriptionID` property is always required) and click **Send** to make the request. The result of the operation request is shown below the properties using a tree control, as shown in Figure 3.

**Figure 3. Viewing the result of an operation**
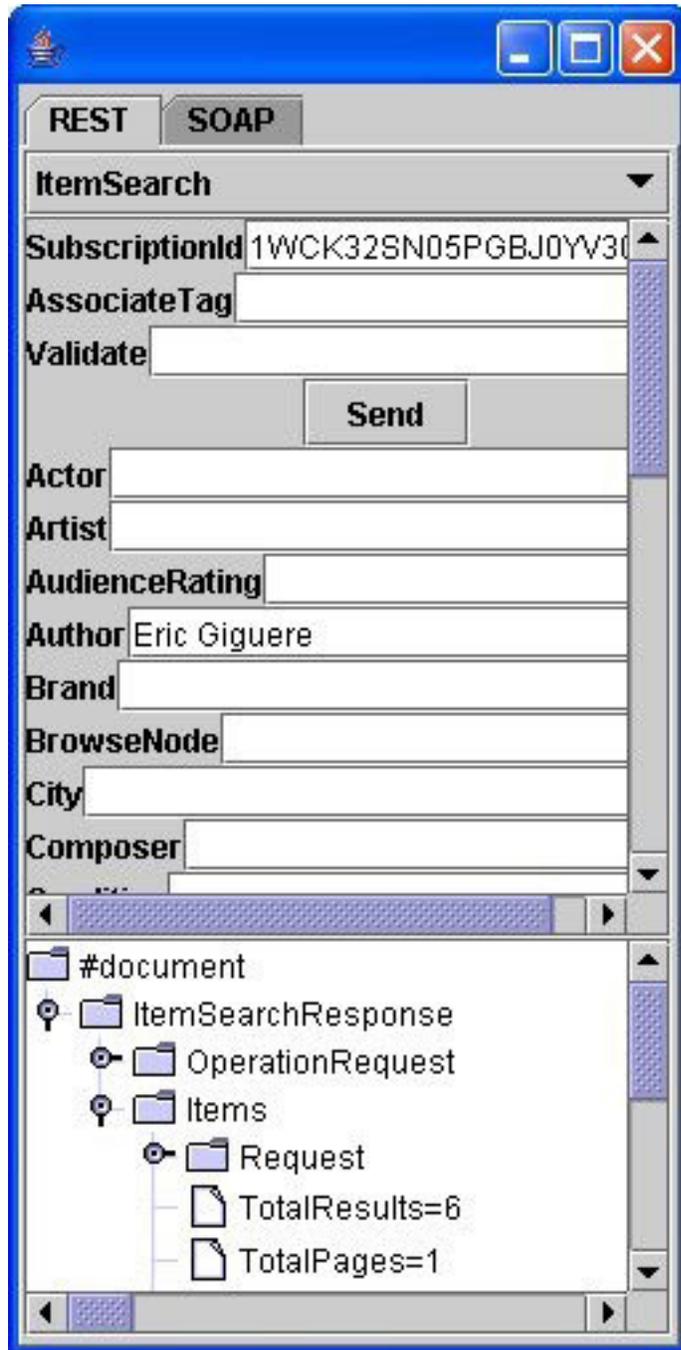
Explore AWS operations

AWS includes extensive online help through the Web services themselves, which makes it simple to explore each operation.

Two levels of help are available. The first is the `Help` operation, which lists the required and optional parameters for a given operation as well as the response groups that are available, as shown in Figure 4.
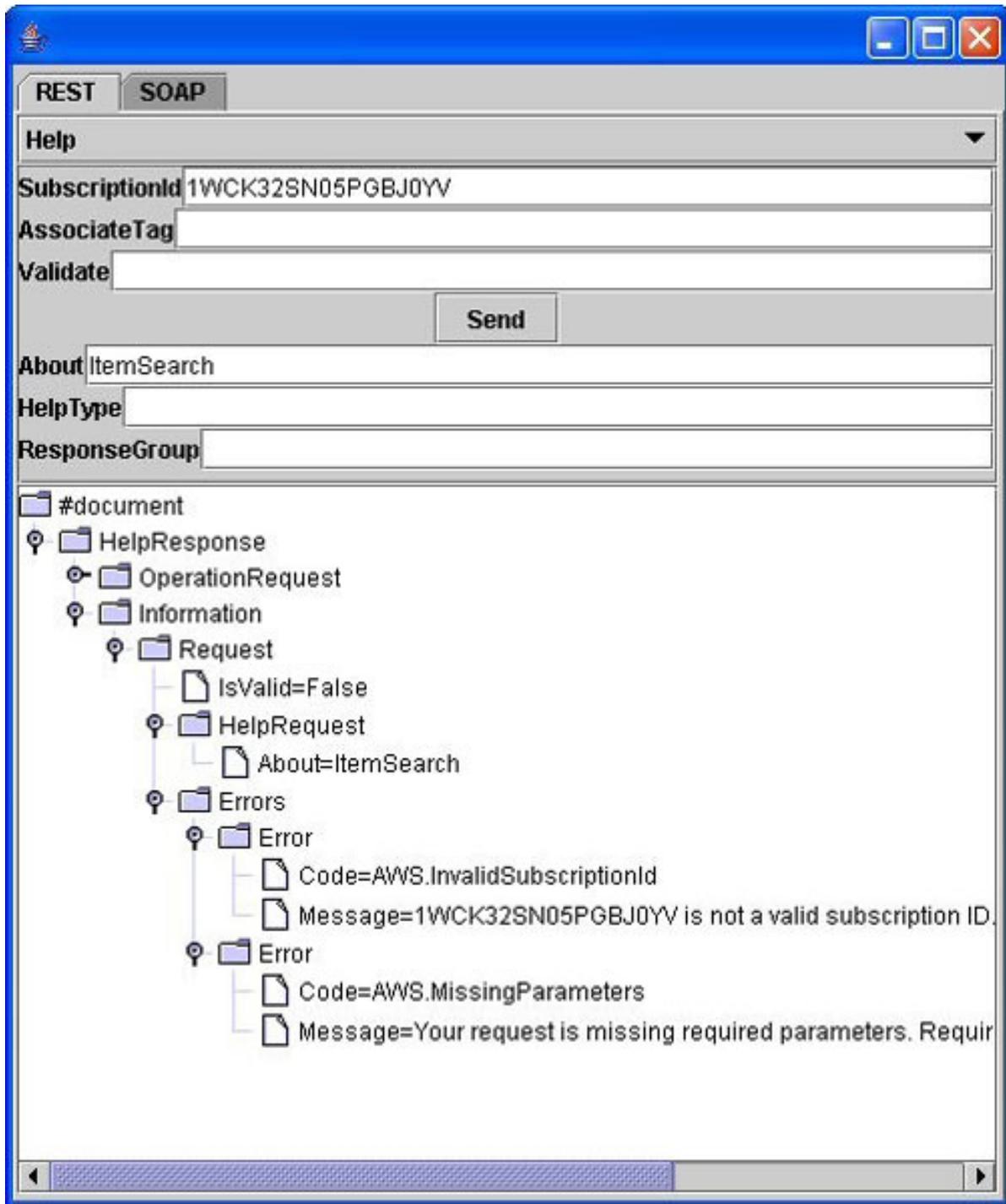
Figure 4. The Help operation
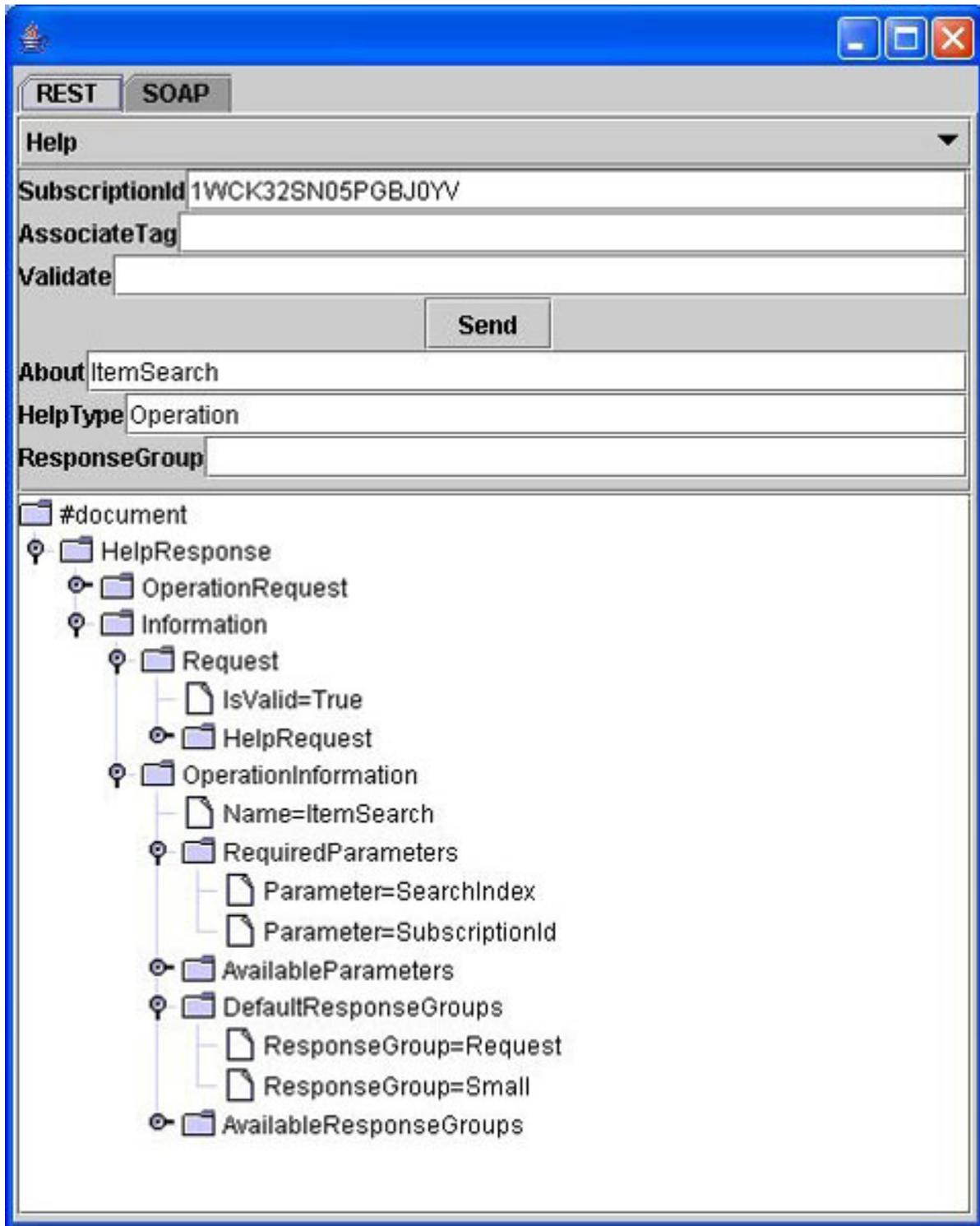
The second form of help is a validity check found in every response. If you look

closely at the response shown above in Figure 4, you'll see that there is an `IsValid` property. When its value is *true*, you know that the response is valid. If set to *false*, as shown in Figure 5, the operation request is not valid.

Figure 5. An invalid operation

What's really nice, however, is that invalid operations include the details about what's wrong with the request. The operation shown in Figure 5 returns the following

error messages:

```
1WCK32SN05PGBJ0YV is not a valid subscription ID.
Please retry with a valid subscription ID.
Your request is missing required parameters. Required parameters include HelpType.
```

From this feedback, it's easy to see that there are two problems with the request: an invalid subscription ID and a missing request property. Don't know what values are allowed for the `HelpType` property? Just set it to something and resubmit the operation. You'll get back this message:

```
The value you specified for HelpType is invalid.
Valid values include ResponseGroup, Operation.
```

This kind of information makes debugging really easy. You can even test operations for validity without actually running them by setting the `Validate` property to *true* and checking the result of the `IsValid` property in the response.

Responses

The data that an AWS operation returns is an XML document defined using the XML schema description (XSD) found at http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.xsd. You can load the XSD directly into most Web browsers for viewing.

Each AWS operation has a corresponding XSD element defined. The response element's name consists of the operation name suffixed with *Response*. For example, the `ItemLookup` operation returns an XML document matching the `ItemLookupResponse` element:

```xml
<xs:element name="ItemLookupResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:OperationRequest" minOccurs="0" />
      <xs:element ref="tns:Items" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Most of the response data for `ItemLookup` is contained in the `Items` element:

```xml
<xs:element name="Item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ASIN" type="xs:string"/>
      <xs:element ref="tns:Errors" minOccurs="0"/>
      <xs:element name="DetailPageURL" type="xs:string" minOccurs="0"/>
```

```
      <xs:element name="SalesRank" type="xs:string" minOccurs="0"/>
      <xs:element name="SmallImage" type="tns:Image" minOccurs="0"/>
      <xs:element name="MediumImage" type="tns:Image" minOccurs="0"/>
      <xs:element name="LargeImage" type="tns:Image" minOccurs="0"/>
      <xs:element ref="tns:ItemAttributes" minOccurs="0"/>
      <xs:element ref="tns:OfferSummary" minOccurs="0"/>
      <xs:element ref="tns:Offers" minOccurs="0"/>
      <xs:element ref="tns:VariationSummary" minOccurs="0"/>
      <xs:element ref="tns:Variations" minOccurs="0"/>
      <xs:element ref="tns:CustomerReviews" minOccurs="0"/>
      <xs:element ref="tns:EditorialReviews" minOccurs="0"/>
      <xs:element ref="tns:SimilarProducts" minOccurs="0"/>
      <xs:element ref="tns:Accessories" minOccurs="0"/>
      <xs:element ref="tns:Tracks" minOccurs="0"/>
      <xs:element ref="tns:BrowseNodes" minOccurs="0"/>
      <xs:element ref="tns:ListmaniaLists" minOccurs="0"/>
      <xs:element ref="tns:SearchInside" minOccurs="0"/>
      <xs:element ref="tns:PromotionalTag" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
```

As you can see, this is a *large* data structure. Most of the nested elements, however, are optional -- their `minOccurs` attributes are set to 0. This is where the AWS concept of a *response group* comes into play: Response groups (specified on a per-request basis using the `ResponseGroup` request property) determine which of the optional elements actually get included in the response. The Request response group, for example, returns only the `ASIN` element (it's required) and drops all the optional elements. The Small response group, on the other hand, returns the `DetailPageURL` and `ItemAttributes` elements in addition to the `ASIN` element. Unforunately, there is no formal specification for which elements actually appear in which response group; all you have to go by are the examples in the *ECS Developer Guide* and your own experiments with the Java tool. Also, different operations support different response groups; you can find those lists in the Developer Guide, as well.

Note that you can list multiple response groups in the `ResponseGroup` property. Just separate them with commas. The response elements from each response group will be included in the final response.

Section 6. Use AWS in your applications

Use the right site

Whether you're using REST or SOAP, your first task is to decide which Amazon site

you're going to communicate with. Each of the six sites -- referred to as *locales* --
uses different base URLs (for REST) or endpoints (for SOAP).

The REST base URLs are:

- **United States (US):**
  http://webservices.amazon.com/onca/xml?Service=AWSECommerceService

- **United Kingdom (UK):**
  http://webservices.amazon.co.uk/onca/xml?Service=AWSECommerceService

- **Germany (DE):**
  http://webservices.amazon.de/onca/xml?Service=AWSECommerceService

- **Japan (JP):**
  http://webservices.amazon.co.jp/onca/xml?Service=AWSECommerceService

- **Canada (CA):**
  http://webservices.amazon.ca/onca/xml?Service=AWSECommerceService

- **France (FR):**
  http://webservices.amazon.fr/onca/xml?Service=AWSECommerceService

The SOAP endpoints are almost identical:

- **US:**
  http://webservices.amazon.com/onca/soap?Service=AWSECommerceService

- **UK:**
  http://webservices.amazon.co.uk/onca/soap?Service=AWSECommerceService

- **DE:**
  http://webservices.amazon.de/onca/soap?Service=AWSECommerceService

- **JP:**
  http://webservices.amazon.co.jp/onca/soap?Service=AWSECommerceService

- **CA:**
  http://webservices.amazon.ca/onca/soap?Service=AWSECommerceService

- **FR:**
  http://webservices.amazon.fr/onca/soap?Service=AWSECommerceService

The WSDL locations for each SOAP endpoint are:

- **US:**
  http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

- **UK:**
  http://webservices.amazon.com/AWSECommerceService/UK/AWSECommerceService.ws

- **DE:**
  http://webservices.amazon.com/AWSECommerceService/DE/AWSECommerceService.ws

- **JP:**
  http://webservices.amazon.com/AWSECommerceService/JP/AWSECommerceService.wsd

- **CA:**
  http://webservices.amazon.com/AWSECommerceService/CA/AWSECommerceService.ws

- **FR:**
  http://webservices.amazon.com/AWSECommerceService/FR/AWSECommerceService.ws

Finally, the XML schemas are available from these locations:

- **US:**
  http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.xsd

- **UK:**
  http://webservices.amazon.com/AWSECommerceService/UK/AWSECommerceService.xsd

- **DE:**
  http://webservices.amazon.com/AWSECommerceService/DE/AWSECommerceService.xsd

- **JP:**
  http://webservices.amazon.com/AWSECommerceService/JP/AWSECommerceService.xsd

- **CA:**
  http://webservices.amazon.com/AWSECommerceService/CA/AWSECommerceService.xsd

- **FR:**
  http://webservices.amazon.com/AWSECommerceService/FR/AWSECommerceService.xsd

If you're wondering why there are separate WSDL and XSD files for each locale, it's because not all operations are available in every locale. For example, item lookup by UPC is currently available only in the US locale.

Make the request: Create a helper class

According to Jeff Barr, Amazon's Web services evangelist, most developers are accessing AWS through the REST interfaces rather than through SOAP, so I restrict my code examples to REST. If you want to use SOAP instead, just import the correct WSDL into your development tool and have it generate the appropriate bindings for you.

In Java language, a REST request is coded in two steps. First, you must form the full URL for the request, starting with the appropriate base URL and appending request properties as HTTP query parameters. Second, you make an HTTP request with the URL and read the XML response from the resulting input stream. Built-in Java HTTP support makes the latter almost trivial.

To form the request URL, create a simple helper class called `ECSHelper`:

```java
import java.util.*;

// A helper class for building REST URLs for
// Amazon E-Commerce Service (ECS) operations.
// Can be used with J2SE/J2EE or J2ME.

public class ECSHelper {

    public ECSHelper( String subID ){
        this( subID, null, "US" );
    }

    public ECSHelper( String subID, String assocTag ){
        this( subID, assocTag, "US" );
    }

    public ECSHelper( String subID, String assocTag, String locale ){
        _locale = locale.toUpperCase();
        _subID = subID;
        _assocTag = assocTag;
    }

    public void clear(){
        _properties.clear();
    }

    public String getLocale(){
        return _locale;
    }

    public String getProperty( String name ){
        return (String) _properties.get( name );
    }

    public void setLocale( String locale ){
        _locale = locale.toUpperCase();
    }

    public void setProperty( String name, String value ){
        if( value == null || value.length() == 0 ){
            _properties.remove( name );
        } else {
            _properties.put( name, value );
        }
    }

    public String toString(){
        StringBuffer b = new StringBuffer();

        b.append( (String) _baseURL.get( _locale ) );
        b.append( "&SubscriptionId=" );
        b.append( _subID );
```

```
        if( _assocTag != null ){
            b.append( "&AssociateTag=" );
            b.append( _assocTag );
        }

        Enumeration e = _properties.keys();
        while( e.hasMoreElements() ){
            String name = (String) e.nextElement();
            String value = (String) _properties.get( name );

            b.append( '&' );
            b.append( name );
            b.append( '=' );
            b.append( value );
        }

        return b.toString();
    }

    private String    _assocTag;
    private String    _locale;
    private Hashtable _properties = new Hashtable();
    private String    _subID;

    // Build the table of base URLs

    private static final Hashtable _baseURL = new Hashtable();

    static {
        String prefix = "http://webservices.amazon.";
        String suffix = "/onca/xml?Service=AWSECommerceService";

        _baseURL.put( "US", prefix + "com" + suffix );
        _baseURL.put( "UK", prefix + "co.uk" + suffix );
        _baseURL.put( "DE", prefix + "de" + suffix );
        _baseURL.put( "JP", prefix + "co.jp" + suffix );
        _baseURL.put( "CA", prefix + "ca" + suffix );
        _baseURL.put( "FR", prefix + "fr" + suffix );
    }
}
```

Note that `ECSHelper` uses the `Hashtable` class to store data mappings instead of `HashMap`: This is done deliberately so that the class can be used in Java 2 Platform, Micro Edition (J2ME) applications as well as Java 2 Platform, Standard Edition (J2SE)/Java 2 Platform, Enterprise Edition (J2EE) applications. Also, the class needs to do more error checking and encode parameter values (replacing spaces with *%20*, for example -- easily done in J2SE using the java.net.URLEncoder.encode method, but harder to do with J2ME).

Make the request: Create an instance of ECSHelper

Within your application, then, all you need to do is create an instance of `ECSHelper` by passing in your subscription ID:

```
ECSHelper helper = new ECSHelper( "1XVJDKFAA232323" );
```

You can also optionally pass in an Associate tag and a two-character locale. If no locale is specified, it defaults to *US* and the main Amazon.com site.

Now, set the desired request properties and call `toString()` to obtain the final URL:

```
helper.setProperty( "Operation", "ItemLookup" );
helper.setProperty( "ItemId", "0321321146" );
helper.setProperty( "ResponseGroup", "Large" );
String url = helper.toString();
```

With the URL in hand, the application can make the HTTP request. For J2SE/J2EE applications, this is done with the `URL` and `URLConnection` classes:

```
public java.io.InputStream invokeECS( String url )
                            throws java.io.IOException {
    java.net.URL u = new java.net.URL( url );
    java.net.URLConnection conn = u.openConnection();

    return conn.getInputStream();
}
```

In J2ME -- and more specifically with the Mobile Information Device Profile (MIDP) -- you use the `HttpConnection` interface:

```
public java.io.InputStream invokeECS( String url )
                            throws java.io.IOException {
    javax.microedition.io.HttpConnection conn =
        (javax.microedition.io.HttpConnection) javax.microedition.io.Connector.open( url );
    try {
        return conn.openInputStream();
    }
    finally {
        conn.close(); // close it because no finalizers in J2ME
    }
}
```

Assuming that no exception has occurred, you're now ready to process the response stream.

Process the response

For debugging purposes, you might simply want to read the input stream and dump its contents to the console or a log file using a method like this:

```
public void dumpResponse( java.io.InputStream in,
                          java.io.PrintStream out )
                          throws java.io.IOException {
    java.io.InputStreamReader r = new java.io.InputStreamReader( in );
```

```
    int ch;

    while( ( ch = r.read() ) ) != -1 ){
        out.write( (char) ch );
    }
}
```

Typically, though, you want to parse the response with an XML parser. There are
many ways to do this, of course. The standard way in J2SE/J2EE is to convert the
document into a Document Object Model (DOM) tree using the facilities of the Java
API for XML Processing (JAXP):

```
public org.w3c.dom.Document toDOM( java.io.InputStream in )
        throws javax.xml.parsers.ParserConfigurationException,
                org.xml.sax.SAXException,
                java.io.IOException {
    javax.xml.parsers.DocumentBuilderFactory factory =
                javax.xml.parsers.DocumentBuilderFactory.newInstance();
    javax.xml.parsers.DocumentBuilder builder = factory.newDocumentBuilder();

    return builder.parse( in );
}
```

You then navigate the DOM tree in the usual manner. You'll probably find, however,
that the XML document is easier to manipulate using JDOM, an open source project
that defines Java-friendly APIs for processing XML documents. You can use JDOM
in conjunction with or as a replacement for the standard DOM classes. Download the
JDOM binaries from the jdom.org site, and place the jdom.jar file in your classpath.
You can then use this code to parse the document:

```
public org.jdom.Document toJDOM( java.io.InputStream in )
                        throws org.jdom.JDOMException,
                                java.io.IOException {
    org.jdom.input.SAXBuilder builder = new org.jdom.input.SAXBuilder();
    return builder.build( in );
}
```

Printing a JDOM document tree is incredibly easy, by the way, using JDOM's
XMLOutputter class:

```
org.jdom.Document doc = toJDOM( in );
org.jdom.output.Format format = org.jdom.output.Format.getPrettyFormat();
org.jdom.output.XMLOutputter out = new org.jdom.output.XMLOutputter( format );
out.output( doc, System.out );
```

Extracting information from the document is just a matter of referring to the ECS
schema and finding the appropriate element in the tree. For example, here's how
you extract the name of a book author:

```
org.jdom.Document doc = toJDOM( in );
```

```
// Find the root element and the root element's namespace

org.jdom.Element e = doc.getRootElement();
org.jdom.Namespace ns = e.getNamespace();

// Navigate to Items/Item/ItemAttributes/Author
// within the same namespace

e = e.getChild( "Items", ns )
     .getChild( "Item", ns )
     .getChild( "ItemAttributes", ns )
     .getChild( "Author", ns );

// Get the author name

String author = e.getText();
```

Things are slightly more complicated if you're making ECS requests from a J2ME platform. Most J2ME devices do not yet include an on-board XML parser, and because application size is a real issue with these devices, it's very important to use a small and lightweight parser to process the response. A *pull parser*, like the one described in the article Add XML parsing to your J2ME applications is your likeliest choice.

Section 7. A sample application

The TopRank application

To finish this tutorial, you're going to build a simple Java application with ECS 4.0. The application, *TopRank*, lets you list the top-ranked books by a given author on any of the Amazon sites, as determined by the Amazon sales rank for each book. The application also prints the book titles and calculates an average ranking for the author's books.

Understanding the ItemSearch operation

The TopRank application is built around ECS's `ItemSearch` operation, which lets you search for products in the Amazon database according to a number of criteria. The author name is your only search criteria, although you'll want the data sorted by sales rank, if possible.

The first step is to launch ECS Explorer to test the `ItemSearch` operation. After entering your subscription ID, set the `Author` property to `Stephen King`, then scroll down the property list to set the `SearchIndex` property to `Books`. Click **Send** to invoke the operation and return information about Stephen King's books. The result is shown in Figure 6.

Figure 6. Initial ItemSearch results
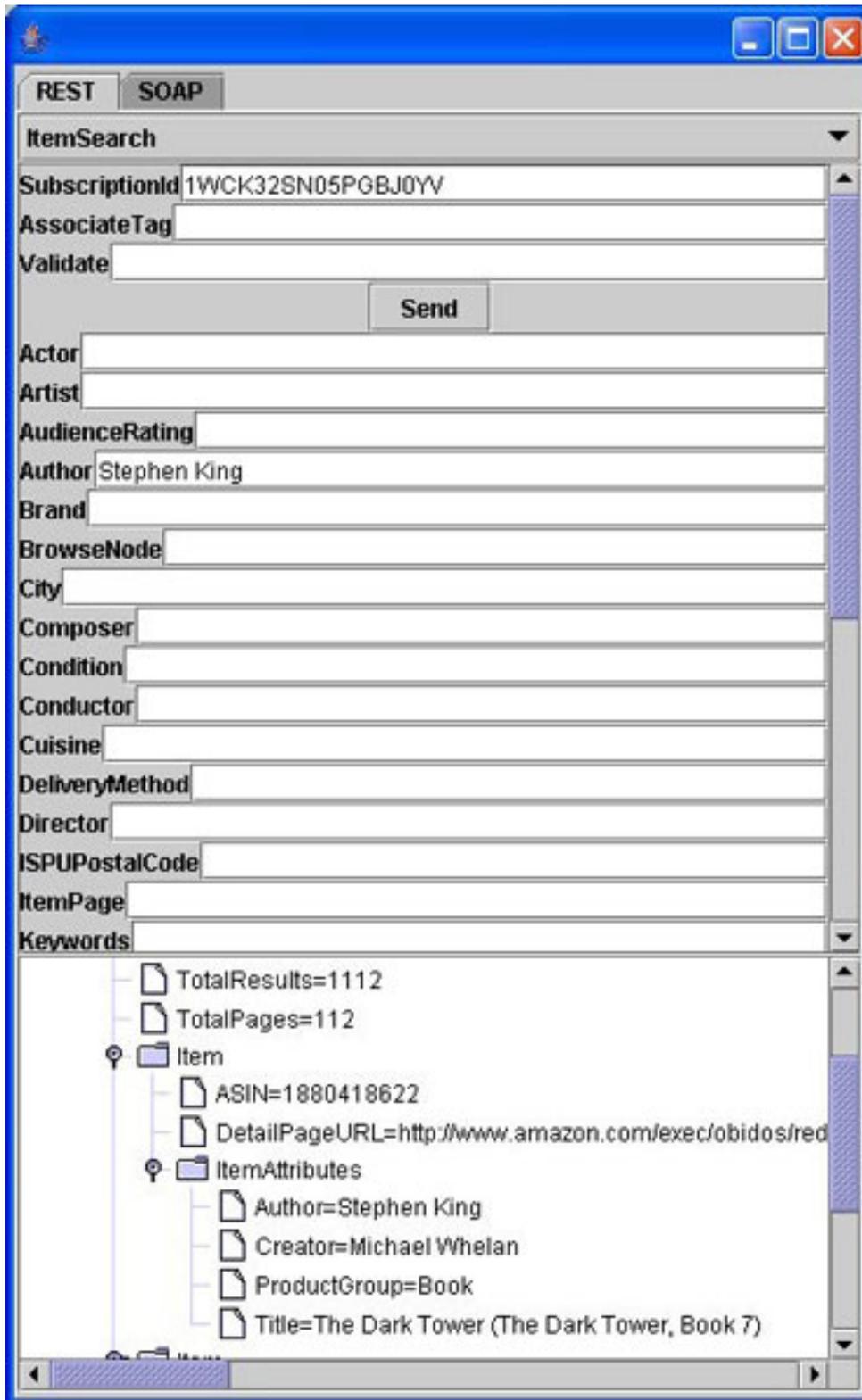
There's a problem, however: The sales rank is not included in the item information. After perusing ECS documentation, you discover that sales rank is returned if you use the SalesRank response group. So, set `ResponseGroup` to `SalesRank`, then resubmit the operation, as shown in Figure 7.

Figure 7. Returning sales rank information
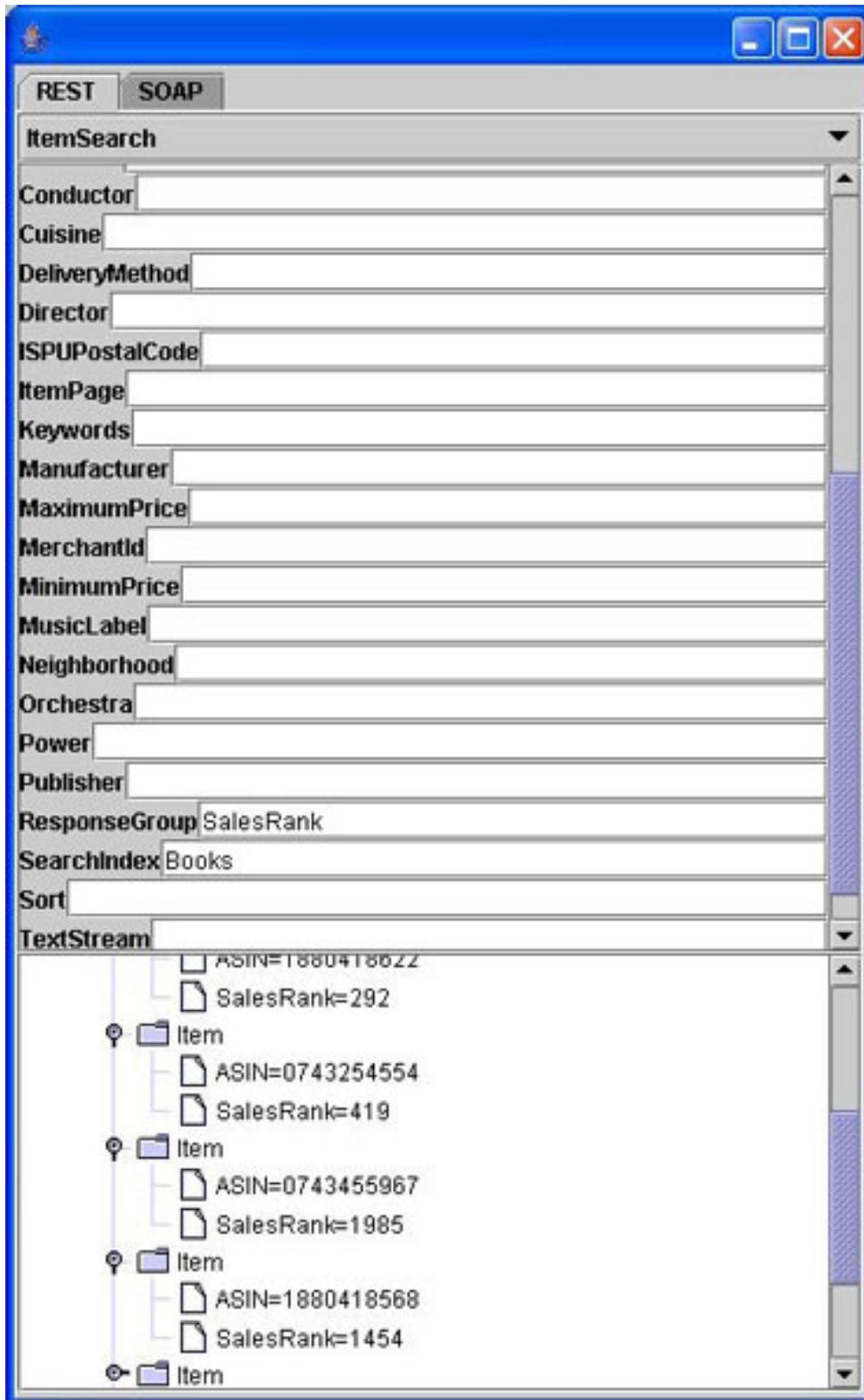
Now you have the sales rank, but you've lost all of the other information about the books! Referring back to the documentation again, you discover that the book title is returned using the Small response group, so set the `ResponseGroup` property to `SalesRank,Small`. Also, set the `Sort` property to `salesrank` so that the data are returned in sorted order. After resubmitting, you get the results shown in Figure 8.

Figure 8. Sales rank and title information sorted by sales rank
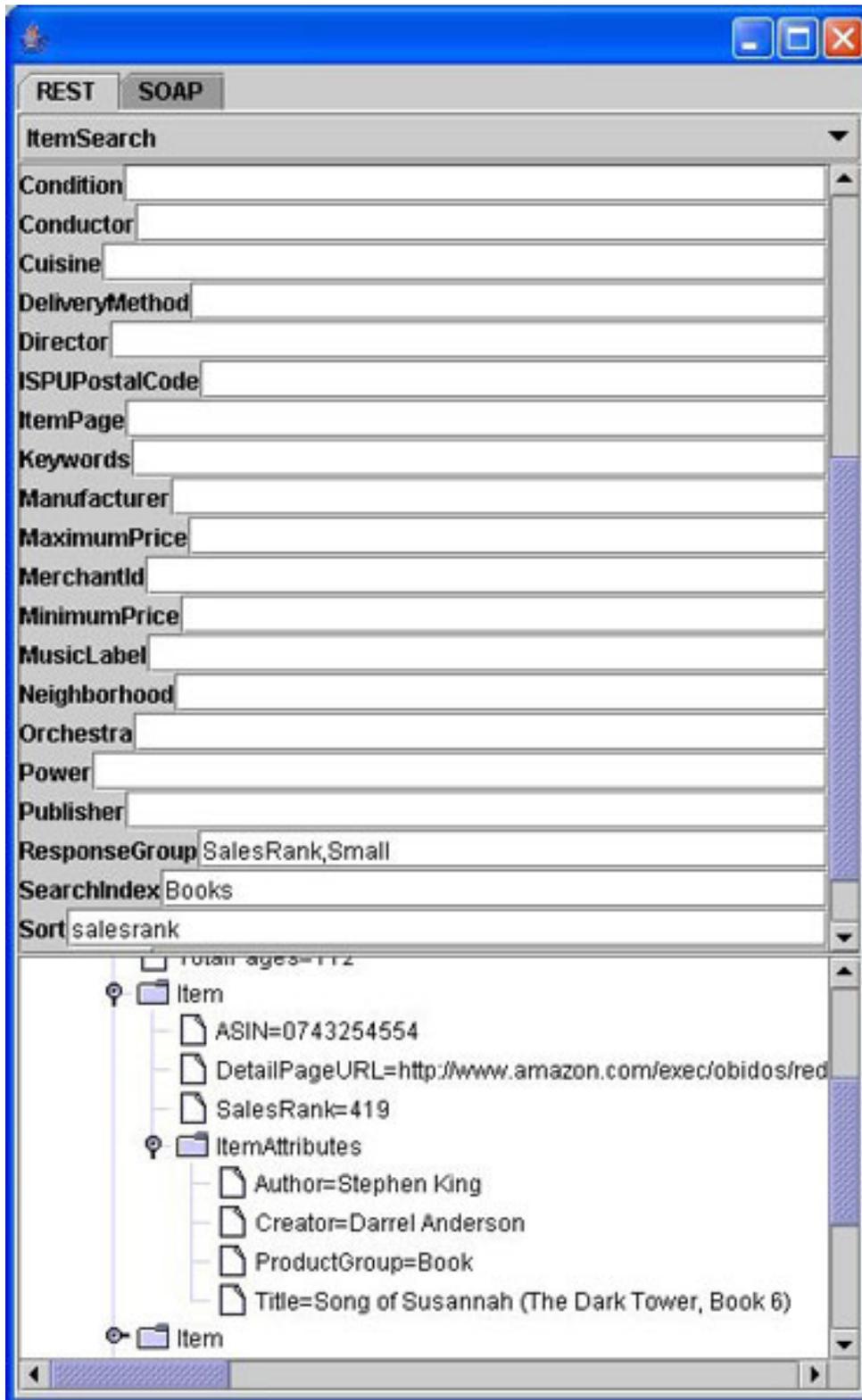
There's just one remaining problem: The operation only returns information about 10 books at a time, and of course, Stephen King has written many more books than that. By setting the `ItemPage` property, however, you can scroll through the data with separate requests, which is what this application does.

Build the application

TopRank is a console application, so all you need in terms of building tools is the JDK and Ant. (For more information about Ant, see Resources.) Besides JDOM, you'll use an open source project called Java Simple Argument Parser (JSAP) to handle your command-line options and Log4J for logging.

Common request properties -- like `SubscriberId` -- are stored in a separate Java properties file, TopRank.properties, which resides in the src directory alongside the source code. Be sure to edit this file before building the project.

The application itself is rather straightforward when you know what to expect in the ItemSearchResponse document. The ECSHelper class and JDOM handle most of the work. The only tricky part is the need to throttle access to the Amazon site so that only one request is made per second, but this is easily done by keeping track of the system time.

Running the application is just a matter of typing:

```
java -jar TopRank.jar "Stephen King"
```

The following command lists the top 10 books by Stephen King:

```
java -jar TopRank.jar --help
```

Use the following to get the full set of program options. For example, this is how to get the list of the top 50 books by Stephen King on the Amazon.co.uk site:

```
java -jar TopRank.jar --locale=UK --total=50 "Stephen King"
```

Section 8. Summary

Summary

This tutorial introduced you to AWS in general and to the Amazon ECS in particular. You've learned how to request information from Amazon using ECS and how to parse the information for use in your own Java applications.

The next step is to take what you've learned and build your own Amazon-powered applications and Web pages. Further tutorials will explore other aspects of AWS, such as the Amazon Simple Queuing System.

Downloads

Description	Name	Size	Download method
Resources for this tutorial	ws-aws1code.zip	509 KB	HTTP

Information about download methods

Resources

Learn

- The Amazon Web Services is your starting point for information and code samples related to AWS.

- The JDOM project provides an easy way to manipulate XML documents in a Java-friendly way.

- The Log4J project is a convenient logging framework for Java applications.

- The book Mining Amazon Web Services by John Mueller describes older versions of the Amazon Web Services.

- The book Google, Amazon, and Beyond by Alexander Nahkimovsky and Tom Myers covers Web services from different vendors.

- The book Professional Web APIs by Denise M. Gosnell includes more up-to-date coverage of the different Web service APIs.

Get products and technologies

- The Ant tool is a standard way to build projects in Java.

- The Java Simple Argument Parser (JSAP) makes command-line argument parsing simple.

About the author

Eric Giguerre
Eric Giguere is a software developer and author who has written numerous articles and several books about all kinds of programming topics. Having just finished his fourth book, "Make Easy Money with Google," he's hard at work on a follow-up title.