

# Develop apps with Web services and the eBay SDK, Part 1: Build an eBay search engine

Skill Level: Introductory

[Alex Garrett \(ljagged@thinkpig.org\)](mailto:ljagged@thinkpig.org)  
Senior Consultant  
Isthmus Group, Inc.

27 Jul 2005

The face of eBay® that most people are familiar with is the company's Web presence. Learn how to write a small application that allows users to execute ad hoc queries against eBay through eBay's SOAP API. The application uses the eBay Java™ SDK. The use case is targeted at a small subset of the API, but you can generally apply the principles.

## Section 1. About this tutorial

Forty percent of eBay's listings are driven by applications that use eBay's Web services. Half of these listings are from applications written by third-party developers. eBay handles over one billion requests every month from its Web services API. This means two things: a lot of people use eBay through vendor-supplied applications, and these applications could probably be doing more caching.

It's very important to eBay that people begin developing applications using their Web services API because, while there's money to be made by helping people have virtual rummage sales, there's *real* money to be made by helping retailers put their entire business online. That means there's money to be made by writing applications to help eBay help put businesses online. And, like a good partner, eBay is working hard to give you, the developer, everything you need to write applications that will make money for you and make money for eBay. If you don't like the mercenary aspect and you're a fan of Richard Dawkins, think of it this way: eBay is an evolutionary strategy for the junk in your garage to get from where it is (with you and

unappreciated) to where it wants to be (somewhere else and appreciated). The distribution of capital is the factor that helps motivate you to optimize the transfer.

---

## Section 2. Before you start

This tutorial assumes the following:

- You have joined the eBay Developers Program.
  - You have received your devId, appld, and certId.
  - You have downloaded and installed the eBay Java SDK.
  - You have created a user in the sandbox and have the user ID.
  - You have generated an authorization token.
  - You can successfully execute the ApiCallsDemo application in the eBay SDK samples folder.
- 

## Section 3. Introduction

### Overview of the SDK

eBay has created a Web services API and exposed it through SOAP so that you can write eBay applications using any programming language that supports SOAP. Working with SOAP directly can be difficult, so eBay has created a Java SDK that sits on top of the SOAP layer. The SDK abstracts away the details of dealing with SOAP directly. The SDK includes lots of classes, but they neatly fit into three main categories:

- **Kernel classes** are closest to the SOAP layer and, in some cases, serve as the building blocks of classes in the other categories.
- **Call classes** are part of the `com.ebay.sdk.call` package and represent invocations of a particular eBay service.
- **Type classes** are in the `com.ebay.soap.eBLBaseComponents`

package and represent datatypes that are used as parameters or return values for invocations.

## The kernel classes

The kernel classes are in the `com.ebay.sdk` package. This category includes 14 classes; 3 of them are exceptions, and this example uses only 2 directly. The two most important classes are `ApiContext` and `ApiCredential`. `ApiContext` defines the context under which any API calls will be made. The call classes all take an `ApiContext` as a parameter to their constructor. The `ApiContext` also holds a reference to the `ApiCredential`, which is used to authorize the call. If the credentials aren't set correctly, the call will fail.

The `com.ebay.sdk` package also holds classes that let you control logging for the application (by default, the application logs to the log for Axis), define a call-retry strategy for exceptions, and use other general application behaviors. `ApiCall` is also one of the kernel classes, but it's primarily used as a superclass for all the call classes described next. However, if you wish, you can use the `ApiCall` class directly by calling the `execute` or `executeByApiName` method.

## The call classes

These classes wrap SOAP API calls. The pattern of use is as follows:

1. Instantiate a call class, passing in the `ApiContext` in the constructor.
2. Instantiate and populate any parameters needed for the call.
3. Invoke the execution method and capture any return values.

The execution method eventually delegates to one of the `execute` methods on the `ApiCall` class. The actual name of the method differs, depending on the call class. For example, `AddItemCall` is used to list an item; the execution method takes an `ItemType` as a parameter and returns a `FeesType`. So, the pattern of use for listing an item is as follows:

```
ItemType item = new ItemType();
// populate item with all sorts of relevant information (omitted)
AddItemCall call = new AddItemCall(apiContext);
// assume we have an instance of the ApiContext
FeesType fees = call.addItem(item);
```

## The type classes

The type classes are Java representations of the datatypes described by the eBay Web services Web Services Description Language (WSDL) and contain no real behavior. `AbstractRequestType` and `AbstractResponseType` are the

superclasses of most of the classes in this package.

## Section 4. Conduct an eBay search

### How searching works at eBay

The easiest way to do an eBay search is to provide a query string. The format is familiar to anyone who has used a search engine. A query consists of one or more keywords separated by white space. Punctuation is used to provide meta-information for the query. The title and subtitle of all listings are evaluated for the criteria; the item description isn't used in the evaluation. You can find a full description of the query language in the eBay Help pages (see [Resources](#)).

eBay also has an advanced query system that lets you specify criteria other than a query string. For example, you can limit your query to certain categories, and you can filter results based on criteria such as price, listing type, and sellers. For more information about advanced searches, see [Resources](#).

### The use case

The case study you'll develop in this article is an application that takes a string of text, queries eBay, and returns the result. Here's an example:

```
fetch: id title description query: godot +blooper category: 377
```

This line includes three directives: `fetch`, `query`, and `category`. You indicate a directive by putting a colon after the text. Figure 1 lists the grammar.

**Figure 1. The query grammar**

QUERY	->	( FETCH_CLAUSE QUERY_CLAUSE CATEGORY_CLAUSE )+
DESCRIPTOR	->	[a-zA-Z]+:
FETCH_CLAUSE	->	'fetch:' COLUMN+   <nil>
QUERY_CLAUSE	->	'query:' QUERY_STRING
CATEGORY_CLAUSE	->	'category:'

		CATEGORY   <nil>
COLUMN	->	'id'   'title'   'subtitle'   'description'   'timeleft'   'currentprice'
QUERY_STRING	->	[^DESCRIPTOR]
CATEGORY	->	[0-9]+

This is a loose Extended Backus-Naur Form (EBNF) grammar. The words in all caps are non-terminals, and those in lowercase on the right are terminals.

Here are some examples of valid queries:

**fetch: id title currentprice query: godot category: 377**

This fetches the item ID, title, and high bid for all items in category 377 with the word *godot* in the title or subtitle.

**fetch: title category: 377**

This fetches the titles of all items in category 377.

**category: 377 query: godot fetch: id title currentprice**

This is the same as the first query -- order of clauses doesn't matter.

If you don't specify a `fetch` clause, the search defaults to the ID and title. If you don't specify a category, the search defaults to all categories. The only required clause is `query`. Therefore, the simplest query is `query: <keyword>`.

## Section 5. Write the code

### The driver

The application is driven from the console, but it's designed in such a way that the console is merely the presentation layer. The logic could be used just as easily in a Web application or a GUI application, or driven from a database. See [Listing 1](#).

The application is driven from the console, but it's designed in such a way that the console is merely the presentation layer. The logic could be used just as easily in a Web application or a GUI application, or driven from a database. See [Listing 1](#).

## Listing 1. Main.java

```
package org.thinkpig.ebay.search;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;

public class Main {

    public static void main(String[] args) throws Exception {
        System.out.println("eBayQueryTron v.001");
        System.out.println("-----");
        while(true) {
            String input = getInput();
            if (input.startsWith("quit")) { return; }
            AbstractQuery query = QueryParser.parse(input);
            System.out.println("Executing query --> " + query);
            QueryResult result = QueryExecutor.executeQuery(query);
            displayResult(result);
        }

        private static String getInput() throws IOException {
            System.out.print("\n> ");
            BufferedReader in =
                new BufferedReader(new InputStreamReader(System.in));
            return in.readLine();
        }

        private static void displayResult(QueryResult _result) {
            List items = _result.getResultItems();
            for(int i = 0; i < items.size(); ++i) {
                displayItem((QueryResultItem)items.get(i));
            }
        }

        private static void displayItem(QueryResultItem _item) {
            for(int i = 0; i < _item.getNumberOfColumns(); ++i) {
                System.out.println(_item.getValue(i));
            }
        }
    }
}
```

There's nothing remarkable about this code. Suffice to say that it's the entry point into the application and acts as a driver for the main bits. It glues together the logic in the application -- the `QueryParser` and the `QueryExecutor`. It also displays the results from the `QueryExecutor`.

### The QueryParser and the AbstractQuery

The `AbstractQuery` is a datatype that represents the BNF described in [Figure 1](#). The `AbstractQuery` class (see [Listing 2](#)) maps to the `QUERY` rule, and it has an attribute for the three clauses it supports. The descriptors are the attributes in the class; the rules that follow the descriptors map to Java datatypes. For example, the `QUERY_CLAUSE` is formed by the query descriptor (`query:`) followed by the `QUERY_STRING`. This is implemented in Java language as `private String`

queryString. The COLUMN\_CLAUSE is composed of one or more COLUMNS, so it's represented as a List.

## Listing 2. AbstractQuery.java

```
package org.thinkpig.ebay.search;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class AbstractQuery {
    public static final String NO_CATEGORY = "NO CATEGORY";

    private String category = NO_CATEGORY;
    private String eBayQuery = "";
    private List columns = new ArrayList();
    private boolean useDefaults = true;

    public AbstractQuery() {
        this.columns.add("id");
        this.columns.add("title");
    }

    public void setCategory(String _category) {
        if (_category == null) {
            throw new IllegalArgumentException("Category must
            not be null.");
        }
        this.category = _category;
    }

    public String getCategory() {
        return this.category;
    }

    public void addColumn(String _column) {
        if (this.useDefaults) {
            this.columns.clear();
            this.useDefaults = false;
        }
        this.columns.add(_column);
    }

    public List getColumns() {
        return Collections.unmodifiableList(this.columns);
    }

    public String toString() {
        return "fetch: " + this.columns + " query: " +
            this.eBayQuery + " category: " + this.category;
    }

    public void setEBayQuery(String _query) {
        this.eBayQuery = _query;
    }

    public String getEBayQuery() {
        return this.eBayQuery;
    }
}
```

The QueryParser is the object that turns the input string into an AbstractQuery (see [Listing 3](#)).

### Listing 3. QueryParser.java

```

package org.thinkpig.ebay.search;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class QueryParser {
    private static String currentToken = null;
    private static List tokens;
    private static String END = "!END"; // not a member of the // token alphabet

    public static AbstractQuery parse(String _queryString) {
        AbstractQuery query = new AbstractQuery();
        if (_queryString == null || _queryString.length() == 0) {
            throw new ParseException("Query string must be non-empty.");
        }
        // if we don't trim first, split will give us a "" token
        _queryString = _queryString.trim();
        tokens = new ArrayList(Arrays.asList(_queryString.split("\\s+")));
        nextToken(); // prime the pump
        parse(query);
        return query;
    }

    private static void parse(AbstractQuery _query) {
        while(!currentToken.equals(END)) {
            if(currentToken.equals("query:")) {
                nextToken();
                parseQueryClause(_query);
                continue;
            }
            if(currentToken.equals("category:")) {
                nextToken();
                parseCategoryClause(_query);
                continue;
            }
            if(currentToken.equals("fetch:")) {
                nextToken();
                parseFetchClause(_query);
                continue;
            }
            throw new ParseException("Unknown descriptor: " + currentToken);
        }
    }

    private static void parseQueryClause(AbstractQuery _query) {
        StringBuffer buf = new StringBuffer();
        while(!currentToken.equals(END)) {
            if (currentToken.endsWith(":")) { break; }
            buf.append(currentToken);
            buf.append(" ");
            nextToken();
        }
        _query.setEBayQuery(buf.toString().trim());
    }

    private static void parseFetchClause(AbstractQuery _query) {
        while(!currentToken.equals(END)) {
            if (currentToken.endsWith(":")) { return; }
            _query.addColumn(currentToken);
            nextToken();
        }
    }
}

```

```
private static void parseCategoryClause(AbstractQuery _query) {
    _query.setCategory(currentToken);
    nextToken();
}

private static void nextToken() throws ParseException {
    try {
        currentToken = (String)tokens.remove(0);
    } catch (IndexOutOfBoundsException _ioobe) {
        currentToken = END;
    }
}

private static String peek() {
    if(tokens.size() > 0) {
        return (String)tokens.get(0);
    }
    return END;
}
}
```

The `QueryParser` is an example of a simple recursive descent parser. It represents the grammar in a way that's similar to the `AbstractQuery`. You have a method for each transition rule in the grammar. The `parse(AbstractQuery _query)` method corresponds to the `QUERY -> CLAUSE+` rule. Informally, the method says, "As long as there are still tokens, parse as either a query clause, a category clause, or a fetch clause." The `QUERY_CLAUSE` rule is the query descriptor followed by a `QUERY_STRING`. A `QUERY_STRING` is anything that's not a descriptor. The `parseQueryClause` method appends all the tokens it finds to the `queryString` until it finds a descriptor token. The logic is similar for the `parseFetchClause` and `parseCategoryClause` methods.

---

## Section 6. Conduct the eBay query

### The QueryExecutor and the eBayApi

At this point, you have an `AbstractQuery` that you want to use as the basis for an eBay query. The `QueryExecutor` takes the `AbstractQuery`, translates it into the eBay format, and sends it to the Web service (see [Listing 4](#)).

#### Listing 4. QueryExecutor.java

```
package org.thinkpig.ebay.search;

import java.util.Iterator;

import com.ebay.sdk.ApiException;
import com.ebay.sdk.SdkException;
```

```

import com.ebay.sdk.call.GetSearchResultsCall;
import com.ebay.soap.eBLBaseComponents.GetSearchResultsRequestType;
import com.ebay.soap.eBLBaseComponents.ItemType;
import com.ebay.soap.eBLBaseComponents.SearchResultItemType;

public class QueryExecutor {
    private static final EbayApi api = new EbayApi();

    public static QueryResult executeQuery(AbstractQuery _query) throws
    Exception {
        GetSearchResultsCall call = convertQuery(_query);
        SearchResultItemType[] results = call.getSearchResults();
        QueryResult queryResult = convertResults(_query, results);
        return queryResult;
    }

    private static GetSearchResultsCall convertQuery(AbstractQuery _query) {
        GetSearchResultsCall call =
        new GetSearchResultsCall(api.getApiContext());
        call.setQuery(_query.getEBayQuery());
        if (!_query.getCategory().equals(AbstractQuery.NO_CATEGORY)){
            call.setCategoryID(_query.getCategory());
        }
        return call;
    }

    private static QueryResult convertResults(AbstractQuery _query,
    SearchResultItemType [] _results) throws SdkException {
        QueryResult result = new QueryResult();
        if (_results == null) {
            return result;
        }
        for (int i = 0; i < _results.length; i++) {
            SearchResultItemType type = _results[i];
            result.addItem(convertItem(_query, type));
        }
        return result;
    }

    private static QueryResultItem convertItem(AbstractQuery _query,
    SearchResultItemType _resultType) throws SdkException {
        QueryResultItem item = new QueryResultItem();
        Iterator i = _query.getColumns().iterator();
        while(i.hasNext()) {
            String column = (String)i.next();
            String value = findValue(column, _resultType);
            item.addColumn(column, value);
        }
        return item;
    }

    private static String findValue(String _column,
    SearchResultItemType _resultType) throws SdkException {
        ItemType item = _resultType.getItem();
        if (_column.equalsIgnoreCase("id")) { return
        item.getItemID().getValue().trim(); }
        if (_column.equalsIgnoreCase("title")) { return
        item.getTitle().trim(); }
        if (_column.equalsIgnoreCase("subtitle")) { return
        item.getSubTitle().trim(); }
        if (_column.equalsIgnoreCase("description")) {
            return item.getDescription().trim();
        }
        if (_column.equalsIgnoreCase("timeleft")) {
            return item.getTimeLeft() ==
            null ? "Unknown" : item.getTimeLeft().toString();
        }
        if (_column.equalsIgnoreCase("currentprice")) {

```

```
        return
            item.getSellingStatus().getCurrentPrice().toString().trim();
    }
    throw new SdkException("No SDK mapping for column: " + _column);
}
}
```

The `QueryExecutor` does three things: it creates the `GetSearchResultsCall` and populates it with the data from the `AbstractQuery`; it invokes the `getSearchResults()` method on the call object, which invokes the eBay Web service; and it converts the result to a `QueryResult` so that eBay-specific types don't leak out into the rest of the application. The `GetSearchResultsCall` is part of the eBay API, as is the `SearchResultItemType`. They're the only parts of the eBay SDK that the application depends on so far. You're strictly segregating the application from the eBay SDK because doing so helps protect your logic from changes eBay might make to its SDK. eBay is good about maintaining backward compatibility, but the API is evolving rapidly. When you change your code to take advantage of new functionality, you'll want to keep the changes to a minimum.

Notice at the bottom of the `findValue` method that it throws an `SdkException` if the item doesn't have a mapping for a column that you're interested in. The eBay Java API has two primary exceptions: `SdkException` and `ApiException`. The `SdkException` is used to indicate exceptional conditions within the SDK, whereas the `ApiException` indicates an issue with an invocation of the Web service. In this case, you throw an `SdkException` because you can't map the query column to a property of the item. If you tried querying for a category ID that didn't exist, you'd get an `ApiException`.

The `EbayApi` class encapsulates the process of getting an `ApiContext` and populating it with the appropriate credentials (see [Listing 5](#)). When you use this code, be sure to set the `USER_AUTH_TOKEN` to the token of your test user. Also, notice that the `serverURL` points to the sandbox. In a production application, the URL would point to the production server.

### Listing 5. EbayApi.java

```
package org.thinkpig.ebay.search;

import com.ebay.sdk.ApiContext;
import com.ebay.sdk.ApiCredential;

public class EbayApi {
    private static final String USER_AUTH_TOKEN =
        "<<Insert your token here.>>";
    private static final String serverURL =
        "https://api.sandbox.ebay.com/wsapi";

    private ApiContext apiContext;

    public EbayApi() {
        this.apiContext = new ApiContext();
    }
}
```

```
    ApiCredential cred = apiContext.getApiCredential();
    cred.seteBayToken(USER_AUTH_TOKEN);
    apiContext.setApiServerUrl(serverURL);
}

public ApiContext getApiContext() {
    return this.apiContext;
}
}
```

For this example, all you need to do is instantiate a new `ApiContext`, set the `eBayToken` attribute of its `ApiCredential` to your token, and set the `ApiServerUrl`. However, you could also make any other context-level configurations (such as logging) here. Note that the `ApiContext`'s attributes aren't immutable, so any code that needs a reference to the `ApiContext` (and that's a lot of code) can change its configuration. It's probably not something you need to worry about in practice, but it's worth being aware of.

---

## Section 7. Get the query results

### The `QueryResult` and the `QueryResultItem`

You've seen all the good stuff. The only things left are the `QueryResult` ([Listing 6](#)) and the `QueryResultItem` ([Listing 7](#)). The latter represents an individual search result, and the former bundles them all into one object that can be passed around. These classes abstract away the notion of search results into something that the presentation layer can destructure without having to know about the specifics of the eBay SDK.

#### Listing 6. `QueryResult.java`

```
package org.thinkpig.ebay.search;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class QueryResult {
    private final List results = new ArrayList();
    public void addItem(QueryResultItem _item) {
        results.add(_item);
    }

    public List getResultItems() {
        return Collections.unmodifiableList(results);
    }
}
```

Notice that you don't just return a reference to your results. You want to keep the result immutable, so the getter wraps it in an unmodifiable list wrapper.

### Listing 7. QueryResultItem.java

```
package org.thinkpig.ebay.search;

import java.util.ArrayList;
import java.util.List;

public class QueryResultItem {
    private final List columns = new ArrayList();
    private final List values = new ArrayList();

    public void addColumn(String _columnName, String _columnValue) {
        // we use this instead of a map to retain order
        columns.add(_columnName);
        values.add(_columnValue);
    }

    public int getNumberOfColumns() {
        return columns.size();
    }

    public String getValue(int _index) {
        return (String)values.get(_index);
    }
}
```

---

## Section 8. Wrap-up

### Conclusion

This tutorial demonstrated that the eBay SDK is fairly straightforward to use. It's mostly a matter of finding the correct call class, setting its attributes with the appropriate parameters, and invoking it. You don't need to mess with the SOAP pieces. However, it's a bit ingenuous to imply that building eBay applications with the SDK is easy. If it were, the SDK wouldn't ship with a 958-page PDF document; and the developer site wouldn't have forums, and FAQs, and a knowledgebase, and whitepapers, and sample code, and application development classes in streaming video. The eBay SDK is a bit like XML -- it's easy to learn, but it's hard to understand and use well. The good news is that eBay has made available all the resources just mentioned to help make your life as an eBay application developer easier. As I mentioned earlier, the company has a vested interest in getting developers to write applications that use its Web service API, and it will make the process as easy as possible for you. Take advantage of the situation and jump in with both feet.

## Resources

- [Build a marketplace with the eBay SDK and Web services, Part 1](#)
- [Finding your way through Web service standards, Part 1: Will my Web service work with your client?](#)
- Read about the eBay query language on this [Help page](#).
- To try different eBay searches, go to <http://search.ebay.com> and following the Advanced Search link at the bottom of the search form.
- If this article has piqued your interest, but you're new to Web services, look at the [developerWorks: New to SOA and Web services](#) section.
- For a perspective on how Web services and service-oriented architectures (SOAs) are changing the face of business, you may enjoy this developerWorks article: [The Tao of e-business services](#).
- The eBay [Java Developer Center](#) has lots of resources and code samples, as well as the SDK itself.
- eBay also sponsors a [weblog](#) with announcements and additional resources.
- If you appreciate the immersive experience of multimedia, you're sure to enjoy the presentations from the [Developer Education Course Catalog](#).
- Richard Dawkins introduced the science of memetics in this chapter from his book [The Selfish Gene](#).
- If you aren't familiar with Extended Backus-Naur Form, you may want to learn about [EBNF grammar](#).
- Finally, if you found the mini query parser to be worth the price of admission, you'd probably enjoy looking at Terrence Parr's [programming languages course notes](#). (Terrence Parr is the [ANTLR](#) guy.)

## About the author

Alex Garrett

After pursuing an undergraduate degree in classics, linguistics, computer science, psychology, and literature, Alex finally settled on a B.S. in philosophy from the University of Wisconsin. His professional career is as checkered as his academic one. He has been the e-commerce architect for a GE Capital company, a lecturer at a University of Wisconsin school, a systems administrator, an acquisitions editor for a small technical publishing company, and a code toad, among other things. Currently, he's a senior consultant with a Madison,

WI-based firm and a proud new father.