Quick and Easy Solutions With Free Java Libraries

By Shaun Haney

The BBj® API is a toolkit full of solutions for everyday business needs. Businesses of varying sizes may have the need to integrate charts into applications, map data from a database to human-readable information, or even transfer files between client and server. These very viable requirements are now extremely easy to deliver.

Part I

While BASIS continues to add to the breadth of the API, it is possible that the ready-made solutions developers are looking for do not yet exist within the BASIS API. This does not mean they must develop the entire solution themselves. BBj allows developers to integrate Java objects into their application, and with BBj Client Objects, they can perform the work on the client's side.

Say, for example, you are a business with several sales offices that needs an intranet application to convert a customer query from your BBj databases instantly to an Excel spreadsheet. There are, of course, several ways of achieving this task. One solution is to use the BBj ODBC Driver with Microsoft Excel, however it requires that the users have Microsoft Excel and ODBC configured on their machine.

Perhaps a better solution would be to use a Java library that reads and writes Excel files (see **Figure 1**), and then creates the Excel files programmatically. Your system administrator could use JNLP (Java Network Launch Protocol) to deploy the BBj program that performs this task, thus making the program easily accessible via a URL on the business' internal Web site. Users could create the Excel files on their own machines (instead of the server) using BBj's new ClientObjects. While the client systems must have Java installed, they do not require Microsoft Excel or ODBC to create the Excel document on their machine.

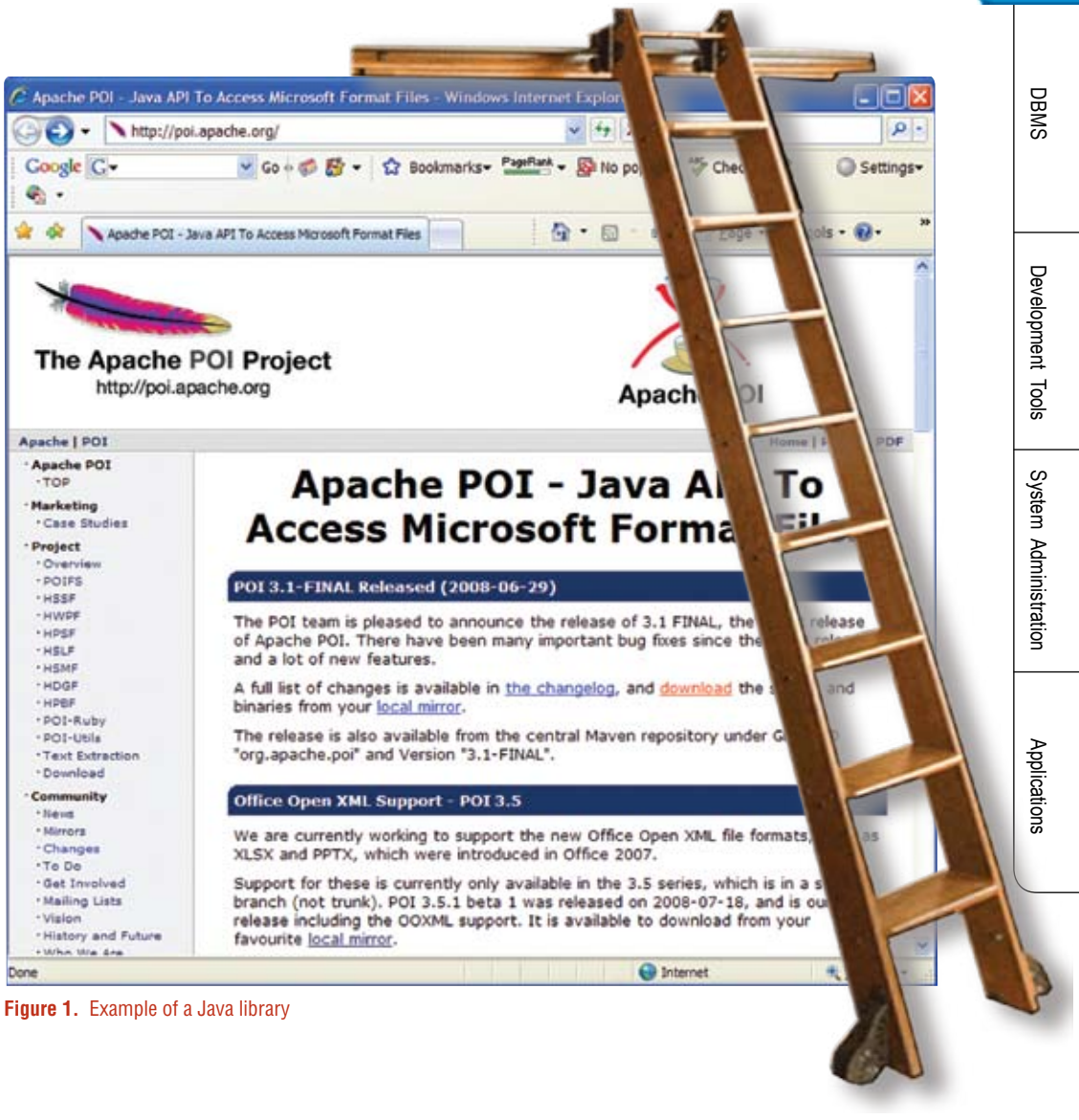


Figure 1. Example of a Java library

Part II

Finding and Downloading a Java Library

The Web offers a plethora of libraries for free download. Simply perform a search to find just the right one to assist with your development requirements. This tutorial focuses on the need for a Java Library that reads and writes Excel files. A quick Google search using "java excel" in the search field locates the Apache POI (Poor Obfuscation Implementation) library. This suite of libraries allows reading and writing of various Microsoft Office document formats programmatically, making it possible to write BBj applications to create output files compatible with Excel.

continued...

Shaun Haney
Quality Assurance Engineer

Follow along with this tutorial as an example of how to download and utilize the Apache POI library.

Download the Java library

1. Go to http://poi.apache.org and locate the subhead (see **Figure 2**) that displays the latest release, which as of this publication date is `POI 3.1-FINAL Released (2008-06-29)`.

2. Click on "download" (or "local mirror").

3. Click on the suggested mirror site to download the library.

4. Choose the `bin` directory.

In this example, you work for a business with several sales offices and need an intranet application to convert a customer query from your BBj databases instantly to an Excel spreadsheet. You would like to use a Java library that reads and writes Excel files, and then creates the Excel files programmatically. Your system administrator will use JNLP (Java Network Launch Protocol) to deploy the BBj program that performs this task, thus making the program easily accessible via the intranet. Users could create the Excel files on their own machines (instead of the server) using BBj's new ClientObjects.

Figure 2. Current release of Apache POI

5. On the page for the `bin` directory, you will see both `tar.gz` and `.zip` format files listed.
   a. For Windows, download the `zip` file.
   b. For non-Windows operating systems, download the `tar.gz` file.

6. Choose a directory into which you want to extract the downloaded archive file and proceed with the extract process. (*I used WinZip to extract the files into my* `H:\Work\Advantage\POI` *directory*.)

7. Locate the JAR files.
   (*I will be using* `poi-bin-3.1-FINAL-20080629\poi-3.1-FINAL\poi-3.1-FINAL-20080629.jar`.)

   For future library downloads, confirm the actual JAR files you need for a particular library by doing the following:
   • Read the available installation and API documentation that comes with the Java library.
   • Browse the JAR file in the IDE as discussed in the "Exploring and Using JARs in the IDE" section.

Adding the JARs to the BBj Classpath

Before you can use these Java classes in BBj, you must add the classes to BBj's classpath.

1. Launch Enterprise Manager and log into `localhost`.

   If BBjServices is running on a different computer, copy the JAR to that machine before adding the JAR to its classpath. Make sure that "Server Information" appears on the left side, then select the **Classpath** tab and press the top green [+] button to launch a file open dialog. Navigate to the `poi-3.1-FINAL-20080629.jar`.

2. Double-click on `poi-3.1-FINAL-20080629.jar` to add it to the list of JARs and directories in BBj's classpath.

continued...

3. To save the JAR into the classpath, click the button that appears as a little disk icon on the bottom right corner of the window.

4. Restart BBjServices.

Exploring and Using JARs in the IDE

The IDE makes the BBj developer's life easier by providing features such as code completion and ready access to compilation and execution at the touch of a button. In addition to these features, the IDE also allows developers to explore methods and fields of classes in Java JARs. This is a good way to become familiar with JARs that contain classes with an unfamiliar API. By allowing the developer to examine the structure of the JAR, the IDE provides the developer with a map of the API.

To mount the `poi-3.1-FINAL-20080629.jar` in the IDE,

1. Launch the IDE and choose the **Filesystems** tab in the IDE.

2. Right click on the root node of the tree labeled **Filesystems** and choose **Mount** from the pop-up menu.

3. Choose **Archive Files** from the submenu. A **File Open** dialog appears.

4. Navigate to the location of `poi-3.1-FINAL-20080629.jar` and select the JAR.

5. Click ["Finish"] to mount the JAR in the **Filesystems** tree.

In addition to being able to explore the JAR, you now have code completion available in your BBj program on all of the classes in the JAR (shown in **Figure 3**).

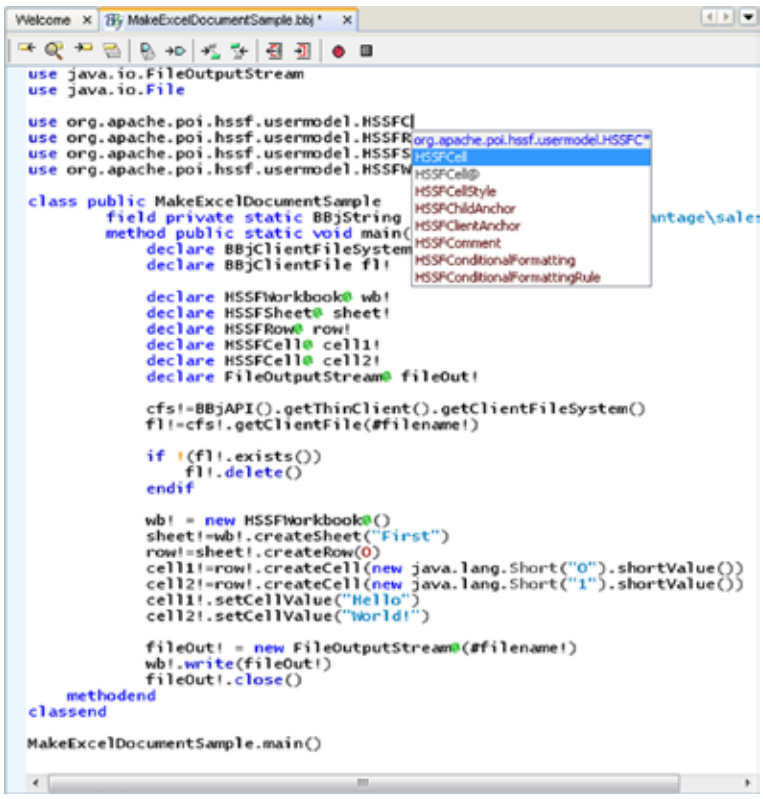


Figure 3. Code completion from mounted JARs

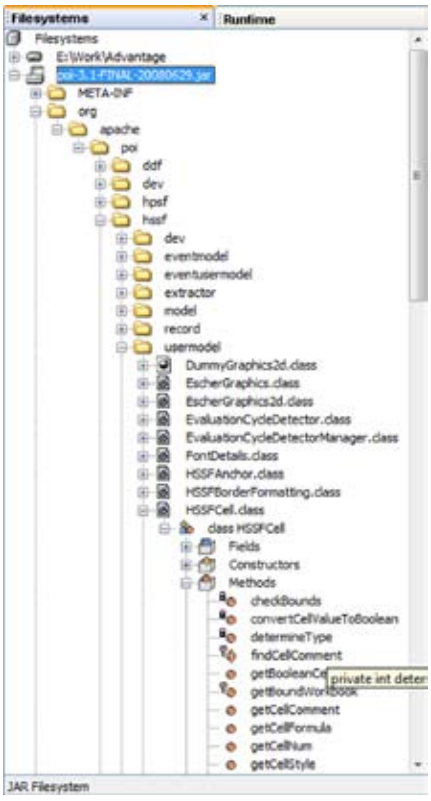


Figure 4. Exploring a mounted a JAR in the IDE

After mounting the JAR in the IDE, expand the JAR and notice the Java classes, their constructors, methods, and fields. Roll the mouse over the classes to see tooltips listing the parameters they take. By exploring a JAR this way, developers can get hints of what functionality the Java classes provide, and what they might look for in the API's documentation as seen in **Figure 4**.

continued...

Next, write a BBj program (**Figure 5**) and then embellish the code to include the BBjClientFileSystem for accessing files on the client instead of the server.

```
use java.io.FileOutputStream
use java.io.File

use org.apache.poi.hssf.usermodel.HSSFCell
use org.apache.poi.hssf.usermodel.HSSFRow
use org.apache.poi.hssf.usermodel.HSSFSheet
use org.apache.poi.hssf.usermodel.HSSFWorkbook

class public MakeExcelDocumentSample
        field private static BBjString filename! = "H:\Work\Advantage\sales.xls"
        method public static void main()
            declare BBjClientFileSystem cfs!
            declare BBjClientFile fl!

            declare HSSFWorkbook wb!
            declare HSSFSheet sheet!
            declare HSSFRow row!
            declare HSSFCell cell1!
            declare HSSFCell cell2!
            declare FileOutputStream fileOut!

            cfs!=BBjAPI().getThinClient().getClientFileSystem()
            fl!=cfs!.getClientFile(#filename!)

            if !(fl!.exists())
                fl!.delete()
            endif

            wb! = new HSSFWorkbook()
            sheet!=wb!.createSheet("First")
            row!=sheet!.createRow(0)
            cell1!=row!.createCell(new java.lang.Short("0").shortValue())
            cell2!=row!.createCell(new java.lang.Short("1").shortValue())
            cell1!.setCellValue("Hello")
            cell2!.setCellValue("World!")

            fileOut! = new FileOutputStream(#filename!)
            wb!.write(fileOut!)
            fileOut!.close()
    methodend
classend

MakeExcelDocumentSample.main()
```

Figure 5. BBj program that programmatically creates an Excel worksheet

Confirm that the BBj Sample works on the same machine without the use of Client Objects, then convert the program to use ClientObjects by adding an @ where appropriate. **Figure 6** shows a comparison of the program before and after the changes, taking advantage of the BASIS IDE's built-in graphical diff capabilities.

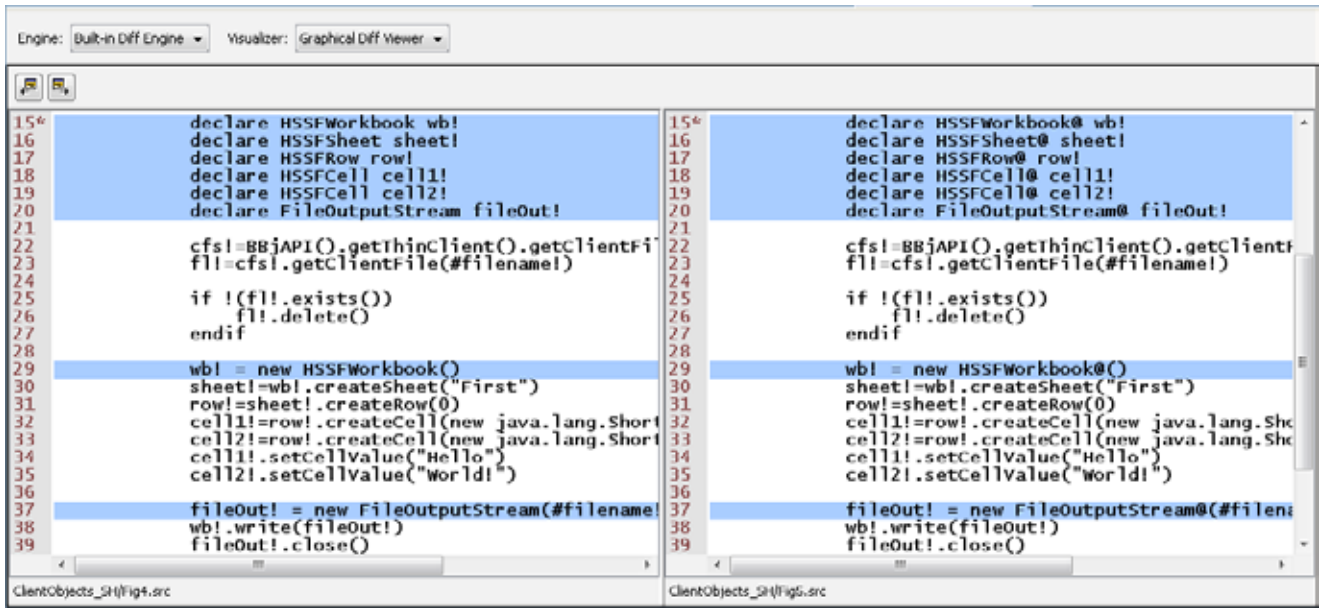


Figure 6. Graphical diff of the Client Object code

continued...

JAR Registration

For development purposes, obtain a DVK license from BASIS. It allows you to integrate Java classes as client objects into BBj applications during development. Remember to register your JAR files with BASIS before final distribution of your application so that clients can run BBj programs that utilize Client Objects without nag messages. Registering a JAR is easy to do, either in BBj or from the command line. To register the JAR, enter the following statement:

In BBj

```
source$ = "c:\temp\poi-3.1-FINAL-20080629.jar"
target$ = "c:\temp\dist\poi-3.1-FINAL-20080629.jar"
com.basis.jarreg.client.JarRegistrar.registerJar(source$, target$)
```

From the command line (with an appropriately set Java classpath)

```
java com.basis.jarreg.client.JarRegistrar c:\temp\poi-3.1-FINAL-20080629.jar c:\temp\dist\poi-3.1-FINAL-20080629.jar
```

Note: This example assumes that the original unregistered JAR is in `c:\temp` and that `c:\temp\dist` already exists but does not contain the new JAR before you run the command. Registering a JAR allows programs to use the JAR for the current version of BBj and any future releases. For example, if you were to register the JAR using BBj 10.0, the JAR would be usable in versions 10.x, 11.x, and 12.x, but not in 9.x.

Deploying Applications Using the JNLP

To save your system administrators and internal users the time and hassle of installations and updates, make your applications available on your company intranet via JNLP. The only real deployment required is configuring the Web server and uploading your applications to it. Users can access their application and updates the moment they are available; they do not even have to be aware that updates are available because the updates occur automatically the first time the user clicks on the icon after the JNLP files are updated.

Configuring JNLP is different depending on the type of Web server used. Since Apache is quite common, use this as the example for setting up the JNLP.

Readying Apache for JNLP

1. In the Apache installation, locate the `<Apache install>\conf\mime.types` file and verify that it has the following entry:

   ```
   application/x-java-jnlp-file              jnlp
   ```

2. Add the line if it is not present, and restart the Apache server.

Create a Subdirectory Under the Root Web Directory for the Application

Find out which directory the Apache Web Server keeps its documents by looking for the DocumentRoot directory specification in the `<Apache install>\conf\httpd.conf` file. There will be an entry like the following:

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs"
```

To change the directory, search the file for any occurrences of "DocumentRoot" and change any directories that need to be the same as DocumentRoot accordingly.

Under the DocumentRoot directory, create a directory called excel_demo. Into this directory, copy the BBj program that tests the POI API, and the necessary BBj JARs.

In order to load and use the JARs in the JNLP, you must sign them; if not, you will get an error trying to run the JNLP that says it cannot verify the JARs. In order to sign the JARs, generate a key (as shown in **Figure 7**) and then use the key to sign the JARs (as shown in **Figure 8**). We generate the key with the "keytool" executable that comes with the Java JDK. Launch a terminal or a command prompt and perform the steps shown in **Figures 7** and **8**. (*In the portion that says* "Enter key password for <excel>," *press [Enter].*)

continued...

```
c:\shaun_exp\htdocs\excel_demo>keytool -genkey -keyalg rsa -keystore excelkeystore -alias excel
Enter keystore password:  excel6
What is your first and last name?
  [Unknown]:  Shaun Haney
What is the name of your organizational unit?
  [Unknown]:  Quality Assurance
What is the name of your organization?
  [Unknown]:  BASIS International Ltd.
What is the name of your City or Locality?
  [Unknown]:  Albuquerque
What is the name of your State or Province?
  [Unknown]:  New Mexico
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Shaun Haney, OU=Quality Assurance, O="Basis International Ltd. ", L=Albuquerque, ST=New
Mexico, C=US correct?
  [no]:  yes

Enter key password for <excel>
        (RETURN if same as keystore password):

c:\shaun_exp\htdocs\excel_demo>
```

Figure 7. Generate a keystore

Once you have made the keystore, use it to sign the JARs by typing the text in **Figure 7** at the command line:

```
c:\shaun_exp\htdocs\excel_demo>jarsigner -keystore excelkeystore BBjThinClient.jar excel
Enter Passphrase for keystore: excel6

Warning: The signer certificate will expire within six months.

c:\shaun_exp\htdocs\excel_demo>jarsigner -keystore excelkeystore BBjUtil.jar excel
Enter Passphrase for keystore: excel6

Warning: The signer certificate will expire within six months.

c:\shaun_exp\htdocs\excel_demo>jarsigner -keystore excelkeystore    poi-3.1-FINAL-20080629.jar
excel
Enter Passphrase for keystore: excel6

Warning: The signer certificate will expire within six months.

c:\shaun_exp\htdocs\excel_demo>jarsigner -keystore excelkeystore webstart2166.jar excel
Enter Passphrase for keystore: excel6

Warning: The signer certificate will expire within six months.
```

Figure 8. Use the keystore to sign the JAR Files

The warning "The signer certificate will expire within six months" occurs when there is no certificate purchased from a provider like VeriSign. For a small "proof of concept" project such as this one, a six-month certificate provides more time than needed. For a larger project, you will need to purchase a certificate. Certificate providers such as VeriSign provide instructions for using the certificate in a signed JAR.

continued...

Create the JNLP File

To get a JNLP file that can run BBj, refer to the BASIS documentation Running BBj Thin Client with Java Web Start. Modify the example in the documentation to become the JNLP File in **Figure 9**.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for b-commerce Application -->
<jnlp spec="1.0+" codebase="http://stewey/excel_demo" href="excel_demo.jnlp">
<information>
<title>Test for using POI HSSF in BBj</title>
<vendor>BASIS International Ltd.</vendor>
<description>This application creates an Excel spreadsheet on the client's machine.</description>
<description kind="short">Create an Excel spreadsheet</description>
</information>
<security>
<all-permissions/>
</security>
<resources>
<j2se version="1.5+" initial-heap-size="24m" max-heap-size="24m"/>
<jar href="BBjThinClient.jar"/>
<jar href="BBjUtil.jar"/>
<jar href="poi-3.1-FINAL-20080629.jar"/>
</resources>
  <resources os="Windows 95">
   <nativelib href="webstart2166.jar"/>
  </resources>
  <resources os="Windows 98">
   <nativelib href="webstart2166.jar"/>
  </resources>
  <resources os="Windows Me">
   <nativelib href="webstart2166.jar"/>
  </resources>
  <resources os="Windows NT">
   <nativelib href="webstart2166.jar"/>
  </resources>
  <resources os="Windows 2000">
   <nativelib href="webstart2166.jar"/>
  </resources>
  <resources os="Mac OS X">
   <nativelib href="webstart2120.jar"/>
</resources>
<application-desc main-class="com.basis.bbj.client.comm.WebStartLauncher">
     <argument>-q</argument>
     <argument>-RHstewey</argument>
<argument>-cc:/progra~1/basis/cfg/config.bbx</argument>
<argument>c:/shaun_exp/htdocs/excel_demo/MakeExcelDocumentSample.bbj</argument>
</application-desc>
</jnlp>
```

Figure 9. JNLP File for the Example POI application.

Connect to the JNLP Application and Run it

Now launch the application from the Web browser and see it run. (*Since I am working at a machine named "stewey," enter* http://stewey/excel_demo/excel_demo.jnlp *in the address bar of my browser,*)

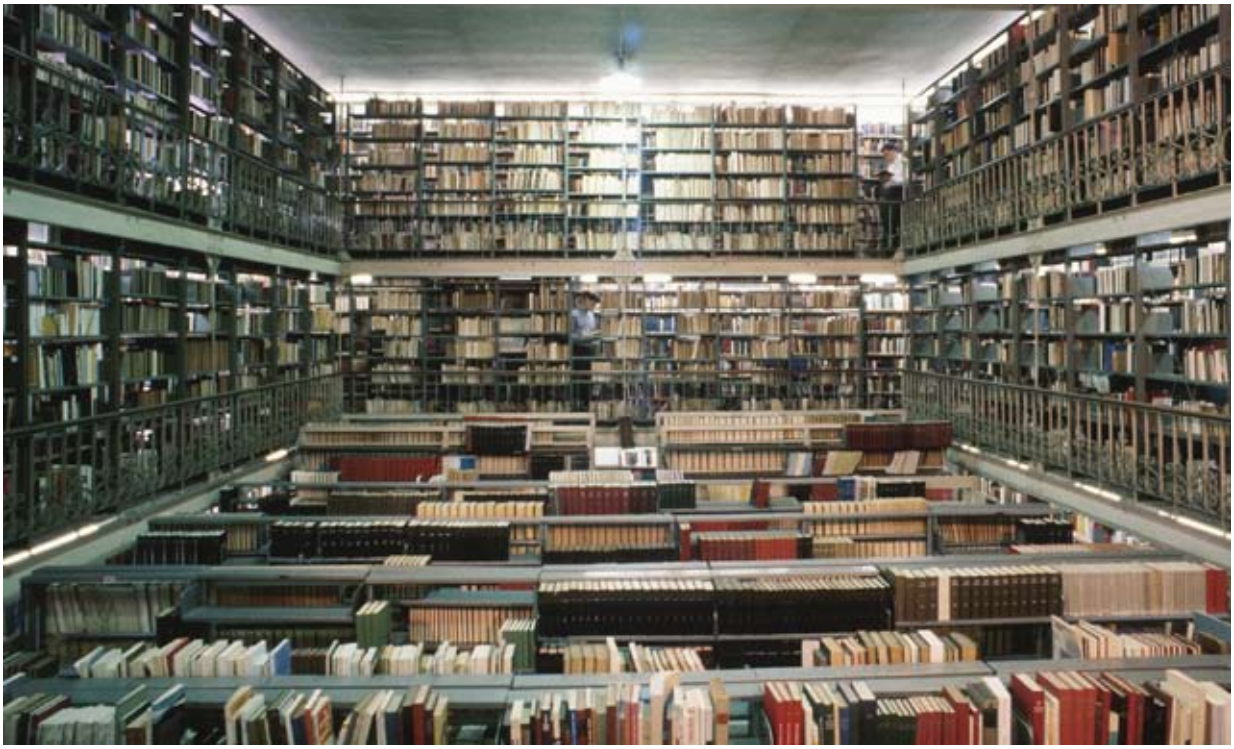
1. Enter the name of your JNLP file into your browser.

   First, you will see a "Java Loading" splash screen, then a security warning. The security warning occurs because there is no purchased certificate from a certificate provider.

2. Click the [Run] button.

   A familiar BBj console pops up and falls to a prompt. The program creates a new Excel file that populates its first two cells with "Hello" and "World!" on a sheet labeled "First."

continued...



This simple example has a few obvious flaws. The absolute path most likely does not exist on the client's machine, which will cause the program to fail. A more in-depth example would have the user browse for his file location or allow the user to configure where on his machine to save the file via a properties file. In order to do a meaningful task, we would need to incorporate a database query or other means of getting information that would be useful to the client. The demo does, however, show the powerful functionality ClientObjects and the JNLP can add to a BBj program.

Summary

In this tutorial, you integrated a free open source Java library that creates Excel files into a BBj program and made this functionality available to clients in a distributed environment. The ability to integrate Java Objects into BBj makes BBj a very extensible tool. A developer can add almost any feature the BBj API does not already contain by creating or searching for a Java Object that does the job and adding it to BBj's classpath. ClientObjects make it possible to distribute the Java Objects to clients in a client-server environment. The JNLP makes this distribution nearly seamless, eliminating the need to install the application on each user's machine thereby saving an enormous amount of time. BBj is a truly valuable and cost efficient tool for a company's intranet application suite. With BBj, the Java library world is your oyster, go and seek your pearls!