

FastPak for Java

Using Update **3** Tools

FastPak for Java, Version 1.2.10.55

Legal Information

All Software and Computer Systems Company, LLC logos shown in this document are a trademark (TM) of Software and Computer Systems Company, LLC. *Java JWaveScope* and *FastPak for Java* are trademark (TM) of Software and Computer Systems Company, LLC. All software produced and licensed by Software and Computer Systems Company, LLC is copyright© Software and Computer Systems Company, LLC **2005 - 2007**. The contents of all pages and images contained in this web site are copyright© Software and Computer Systems Company, LLC, **2007**,

Sun, Sun Microsystems, Solaris and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and certain other countries. X Windows and the X Window System are trademarks of the X Consortium. UNIX is a registered trademark in the United States and other countries of the X/Open Company, Ltd. Apple Macintosh and OS X are trademarks of Apple Computer, Inc. Novell is a registered trademark of Novell, Inc., and SUSE is a trademark of SUSE LINUX Products GmbH, a Novell business. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Motif is a registered trademark of The Open Group.

Unless explicitly stated, original products and services offered, sold, or licensed by Software and Computer Systems, LLC to customers are the exclusive right of Software and Computer Systems Company, LLC. Clients, users, or interested parties should not assume an affiliation exists between Software and Computer Systems Company, LLC and any of the computer manufacturers, operating system distributors, or other vendors that may be used in the production or completion of a work produced by Software and Computer Systems Company, LLC for a customer or product.

Table of Contents

Introduction	4
Notation Conventions Used in This Document	5
The Signed Jar Loader	6
Native Application Launcher	15
The Daemon Mode Auto Terminator	18
The ConfigTool	20

Introduction

Welcome to the *FastPak for Java Using Update 3 Tools* manual.

This manual will briefly describe the use of the following loadable modules that are included in FastPak for Java, Release 1.2.10.55, update 3. This manual will focus on the following tools included in this and subsequent releases of the product:

- **The Signed Jar Loader:** This module allows users to load signed jar applications and verify their signature before launching them. This module can be run via scripts completely outside the FastPak application loader. This manual will describe the generation of keys, the signing of applications, and how to use the loader both inside and outside the FastPak environment. This binary is named "SignedJarLoader.jar" and is located in the "JarApps" directory where FastPak was installed.
- **The Native Application Launcher:** This module allows FastPak to launch native applications outside the FastPak environment. This binary is named "NativeAppLauncher.jar" and is located in the "JarApps" directory where FastPak was installed.
- **Daemon Mode Auto Terminator:** This module allows FastPak to automatically launch some predefined applications in daemon mode and then, after a specified period of time, put the FastPak kernel in an exit state that terminates the Controller Thread while allowing the applications to continue running under a single JVM. When all applications are done running, the FastPak kernel completely terminates. This module alleviates the need for users to manually terminate a FastPak instance running in daemon mode. This binary is named "AutoTerm.jar" and is located in the "JarApps" directory where FastPak was installed. It also requires that the FastPakAPI.jar file be installed in the FastPak class path or be copied to the JarApps directory as well.
- **ConfigTool:** Previously a user had to bring up FastPak to modify configuration parameters. This allows them to modify the parameters without starting FastPak. This tool works completely outside of FastPak (it's not a loadable module). This tool is located in FastPak's base installation directory and should appear as an icon with the words "Config Tool" going diagonally down the icon.

Of the tools just mentioned, the Signed Jar Loader can be used quite effectively outside the FastPak environment as a standalone application or under a FastPak environment as a loadable module. The other tools can also be used outside the FastPak environment, but it wouldn't make much sense to do so.

Notation Conventions Used in This Document

Items that will vary as input for any of the tools identified in this document are surrounded by angle brackets (i.e. <>). These variables are replaced, ***minus the angle brackets***, by their appropriate values. If a value, such as a directory name has spaces embedded in it, then it must be surrounded by double quotes. Here are some examples:

<number of days> could be replaced by the entry 365 to indicate a period of 365 days.

<full path name of keystore> could be replaced on an OS X system by something like:

/Users/smith/MyPrivateKeystore

for a user with a user ID of “smith” and a keystore name of “MyPrivateKeystore”.

On a Windows system this may be something like:

“C:\Documents and Settings\smith\MyPrivateKeystore”

Important note: If the name of a path or a file has embedded spaces in it, the full name must be surrounded by double quotes as shown directly above. This applies to all operating systems.

The Signed Jar Loader

The Signed Jar Loader verifies the signature of an application that has been signed by a private certificate (cert) (not distributed to users) against a public cert (distributed to users). This tool helps to ensure that the file that a user wishes to run has not been modified by an unauthorized third party. If users wish to generate certs and sign documents, they will need access to the tools “keytool” and “jarsigner”. The Signed Jar Loader works only for applications in a “jar” (Java archive) format and it can't be used with flat applications (a distributed set of un-jarred class files).

Use of this tool will be demonstrated by example. The example will consist of the following steps:

1. Generating the keystore for signing jar files
2. Generating the public certs to be distributed to end users.
3. Importing the public cert into an end users keystore.
4. Sign the jar file
5. Distribute the signed jar file
6. Use the Signed Jar Loader on a users machine outside the FastPak environment.
7. Use the Signed Jar Loader under a FastPak environment.

Individuals unfamiliar with jarsigner or keytool should familiarize themselves with these tools before proceeding.

Part I: Generating the keystore for signing jar files

Before any files can be signed and loaded, a keystore must be created which will be used to sign files and generate public certificates (or certs). This is typically done by system administrators or developers.

Generating the keystore for signing files is done as follows:

```
keytool -genkey -validity <number of days> -keystore <full name and path of keystore> -alias <alias name>
```

Note that the command above is entered on a single line.

During this process, the keytool will ask you to provide a password and information about your company, name and address, etc.

Part 2: Generating the public certificates to be distributed to end users.

This is done using keytool in the following manner:

```
keytool -export -keystore <full name and path of keystore> -alias <alias name>  
-file <name of cert to export>
```

This creates a cert with the name you provided in the <name of cert to export>. This cert will be used to be distributed to potential end users of the signed file and must be placed in their own keystore (this will be described later).

Part 3: Importing the public certificate into an end users keystore.

This is done on each users system as:

```
keytool -import -file <name of cert to import>
```

OR

```
keytool -import -keystore <full path and name of special keystore> -file <name of  
cert to import>
```

In both cases above, the <name of cert to import> will be the exportable public cert made in step 2 . This public cert is typically copied to end users machines and this step is then done in the end users directory.

In the first case above, the certificate is placed in the default keystore, which is typically a file named **“.keystore”** in the home directory of the user. In the second case, because the “- keystore” option is specifying a specific keystore path and name, the full path and name of a non- default keystore is given (for example, on Windows it could be “C:\Documents and Settings\Certs\OurCerts.store” or on OS X it could be /Users/Certs/OurCerts.store). If a special keystore is used, when the Signed Jar Loader is used the keystore will need to be provided in the command line. This will be addressed later in this section.

Part 4: Sign the the jar files.

The individual(s) responsible for generating the public and private certs are typically also responsible for signing the files (most often applications)that will be verified by end users with the Signed Jar Loader. The syntax for signing a file is as follows:

```
jarsigner - keystore <full name and path of keystore> <file to be signed> <alias  
name>
```

Note that this is using the keystore created in Part 1 above, and not the public cert imported to end users. Also not that the “- alias” does not precede the <alias name> entry.

Part 5: Distribute the signed jar file(s)

Often, because the signed files are applications, instead of actually manually distributing them to end users, the signed application is put on a web server. The Signed Jar Loader is capable of launching such files remotely by itself or under the FastPak environment. This will be covered later in this document in the following sections. If the signed files are to be copied to the end users machines, after signing, they are just copied like any other file.

Part 6: Use the Signed Jar Loader on a users machine outside the FastPak environment.

The Signed Jar Loader can be used as either a standalone application or under the FastPak environment. In this section, we'll use the Signed Jar Loader as a standalone application. The Signed Jar Loader was designed using the FastPak design guidelines and as such, it's truly capable of running inside or outside the FastPak environment, as are all applications that conform to FastPak design guidelines. Developers may wish to start the Signed Jar Loader via scripts, batch files, or by executing it via a Java, C, or C++ application.

When using the Signed Jar Loader as a command line application, the following format is used:

```
java -jar SignedJarLoader.jar <keystore=name of optional keystore> <protocol>
<hostname> <app path and name> <input params>
```

This command can be broken down as follows:

- **java -jar SignedJarLoader.jar** This is the command line sequence to start the Signed Jar Loader under Java.
- **<keystore=name of optional keystore>** This is *optional* and is used only in the event that a non- default keystore is to be used. Typically, if this value is left blank, then the keystore of the name **.keystore** in the users home directory is used. Otherwise, the name of the keystore must be the full path and name of that keystore. If this option is used, it's best to surround the entire string in double quotes (i.e. "keystore= C:\Our Certs\MyCert.store" for a Windows environment, or "keystore=/Users/smith/MyCerts/MyCert.store" for an OS X, Linux, or Unix type environment.
- **<protocol>** This will always be **file** if the application is on a users local disk, and **http** if the file is to be accessed via a web server.
- **<hostname>** This translates to the host hosting the application. If this is a web server, then it will be a web server name, such as **www.scsc- online.com**, but if it's a local file it will always be **localhost**.
- **<app name and path>** This translates to the full path and name of the file to be loaded by the Signed Jar Loader. If this is a file on a local machine, it must be the full name and path (for example, for Windows this could be something like "C:\Documents and Settings\smith\JarApps\MySignedTest.jar", or for OS X

and Unix/Linux variants it could be /Users/smith/JarApps/MySignedTest.jar). If the file is being hosted on a web server, then it will be the name and path of the file relative to the web server's root. For example, SCSC's main server hosts demo application in it's "test" directory, hence /test/AnalyzerS.jar would access the signed version of the demo program "Analyzer.jar" in the /test directory of our server at www.scsc-online.com.

- **<input params>** This will always be a list of the command line parameters (if any) the application expects to receive. This is *optional*. If there aren't any command line parameters, just leave this blank.

Examples:

SCSC has three signed versions of the demo programs *Analyzer.jar*, *SCSCLogo.jar*, and *FourierAnalysis.jar* called *AnalyzerS.jar*, *SCSCLogoS.jar*, and *FourierAnalysisS.jar*, respectively being hosted on our web server under the /test directory. In the development section of your FastPak distribution, the corresponding public cert for these files, named SCSCDemo.cert, may be found. If you would like to test the Signed Jar Loader for remote loading of these files, follow these steps:

1. Import the SCSCDemo.cert to your keystore, or import it to a special keystore. For this test case, we'll assume it's being loaded into the default keystore location. If you choose to store this cert in a special keystore, then the argument "keystore=<full path and name of keystore>" must precede the "http" entry in the examples below. For example, if you put your keystore in a file on a Windows system called "C:\TestKeyStore\test.store" the entry would be "keystore=C:\TestKeyStore\test.store".
2. Make sure you're connected to the web.
3. Launch the AnalyzerS.jar file with the Signed Jar Loader using the following command:

```
java -jar SignedJarLoader.jar http www.scsc-online.com /test/AnalyzerS.jar
```

You should see a message come up verifying the integrity of the file and then it proceeds to launch the AnalyzerS.jar file. If you wish to try this with something that takes command line parameters, the application SCSCLogoS.jar will accept input values of 480, 600, or 768, (it references the three most common screen sizes height and uses 600 as it's default if none is specified). Try the following:

```
java -jar SignedJarLoader.jar http www.scsc-online.com /test/SCSCLogoS.jar
```

and

```
java -jar SignedJarLoader.jar http www.scsc-online.com /test/SCSCLogoS.jar 480
```

The latter will launch a smaller version of the application as specified by the command line parameter "480". Try launching the unsigned jar files on the SCSC web server, such as Analyzer.jar or SCSCLogo.jar (note the absence of the capital "S" before the .jar extension). These files are unsigned and the Signed Jar Loader will

refuse to launch them. You should get messages like this:

Jar file contains unsigned entries - REJECTED

******* Failed to verify jar file *******

Exiting SignedJarLoader module without launching.

If you wish to try the Signed Jar Loader on a local file, we suggest doing the following:

1. Create a temporary working directory to experiment in.
2. Copy an unsigned Java application in jar format to the working directory.
3. Create a keystore in the directory as described previously.
4. Export a public cert.
5. Sign your application as previously described.
6. Import the exported public cert into either a new keystore or use the default keystore.
7. Run the application with the Signed Jar Loader.

You may wish to copy the jar application *Analyzer.jar* from the JarApps subdirectory from your FastPak installation. We suggest you copy it twice, once as *Analyzer.jar* and the other as *AnalyzerS.jar*. Using the steps described above, sign the *AnalyzerS.jar* file but leave the other one unsigned. After you've done all the setup steps described above, try running each one as follows (this will assume the default keystore isn't specified thus it's using the default keystore named **.keystore** in the users home directory):

```
java -jar SignedJarLoader.jar file localhost <path to jar app>AnalyzerS.jar
```

In the example above, if your *Analyzer.jar* and *AnalyzerS.jar* files were stored in a directory named *C:\SignedTest* the name would be:

```
java -jar SignedJarLoader.jar file localhost C:\SignedTest\AnalyzerS.jar
```

The differences above from the web based version are that “file” replaces “http”, the server name is replaced by “localhost”, and the path name and application is the full path name to the file on the system.

When this is executed, the application should test the integrity of the signed application and then launch it (assuming all has been done properly). Now repeat the same command, but replace the signed jar file *AnalyzerS.jar* with the unsigned file, *Analyzer.jar*. When the Signed Jar Loader tries to execute this, it will find the file

doesn't contain the proper signing entities and fail to launch the program with the following message:

Jar file contains unsigned entries - REJECTED

******* Failed to verify jar file *******

Exiting SignedJarLoader module without launching.

Part 7: Use the Signed Jar Loader under a FastPak environment.

Understanding this section requires a working knowledge of FastPak. If you are not familiar with FastPak, please refer to the users guide for details on configuring an application for use with FastPak.

FastPak is capable of launching multiple applications using a single JVM. When using the Signed Jar Loader under FastPak, the application that FastPak executes is *not* the actual application you wish to launch, but rather the Signed Jar Loader itself. Once the Signed Jar Loader verifies the integrity of the file targeted for launching, it will in turn pass that application to FastPak for actual execution. It will of course, fail if the signature of the targeted application doesn't match that of the public cert stored in the appropriate keystore.

The best way to demonstrate this is by using two examples as follows:

1. **Launch AnalyzerS.jar hosted on the SCSC web site.** For this example, we will use the OS X operating system, assume the public key for this application (found in the developers section under the lib directory of the FastPak distribution called SCSCDemo.cert) has been placed in a keystore called “/SecureKeys/SCSC.store”, and we will assign it a profile name of “RemoteSecureAnalyzer”.
2. **Launch a signed version of SCSCLogo.jar from a local file with an input parameter of “480” for the screen size.** This will require the user to generate the keystore, export the public cert, import the public cert into the default keystore, and sign the application SCSCLogo.jar. We will assign this profile the name “LocalSecureLogo”.

In both examples, we will assume the reader is now capable of generating public keys, signing applications, etc. as previously described.

For example 1, while using FastPak you must use the “Edit” pull down to select the “Configure New” option, and when the menu shows up, enter following information:

1. **FastPak Reference Name:** The name RemoteSecureAnalyzer is entered.
2. **Define Local Application:** Use the menu to navigate and select the SignedJarLauncher.jar application, and the directory entry and file name will be automatically filled in.

3. **Host Name:** This should default to “localhost”. **Leave this as is, since the Signed Jar Loader is located on your system, not remotely.**
4. **Input Arguments:** This is where the Secure Jar Loader obtains information about the application to be executed, whether the application is remote or local. For this example, the complete entry string (which is obscured in the screen capture below by the GUI) is as follows:

keystore=/SecureKeys/SCSC.store http www.scsc-online.com /test/AnalyzerS.jar

The screen capture below shows this being filled out:

Important Notes: When filling out the form, the *Input Arguments* for the Signed Jar Loader will always define where the application targeted to be executed is stored, its type, its path, etc. In the profile form, the “Application Type” must always be Jar since the Signed Jar Loader is a jar file. The “Application Location” option is *always* a local file, since the Signed Jar Loader is stored locally, not on a web server. The “Launch on FastPak Startup”, “Application Priority”, and “Version Identifier” text may be selected as the user sees fit.

When done filling out the form press the OK button and the profile will be added to FastPak's list of available configured applications and will be ready for launching.

For example 2, you must create a unique keystore, export the public cert, import the public cert into the default keystore, and sign the application SCSCLogo.jar. If you wish, you may do this on a copy of SCSCLogo.jar and put it in a completely different directory, just remember to reference the signed version of the application when configuring this profile.

After you're done signing the application and importing public cert, you need to create the FastPak profile for the application. To accomplish this you perform the same steps as in the previous example with some changed.

For this example, we will assume a user named "smith" has created a directory called "/Users/smith/SignedJarApps" and copied the signed version of SCSCLogo.jar into that directory. Assuming this to be true and the fact that you will be using the default keystore (hence no need for a "keystore=..." entry), the profile fields are filled in as follows:

1. **FastPak Reference Name:** The name LocalSecureLogo is entered.
2. **Define Local Application:** Use the menu to navigate and select the SignedJarLauncher.jar application, and the directory entry and file name will be automatically filled in.
3. **Host Name:** This should default to "localhost". **Leave this as is, since the Signed Jar Loader is located on your system, not remotely.**
4. **Input Arguments:** For this example, the complete entry string (which is obscured in the screen capture below by the GUI) is as follows:

file localhost /Users/smith/SignedJarApps/SCSCLogo.jar 480

In the text above, "file" is telling SecureJarLoader.jar to use the file protocol, "localhost" is telling the SecureJarLoader.jar the file is on the users system, "/Users/smith/SignedJarApps/SCSCLogo.jar" is the path and name of the signed jar file, and "480" is the command line parameter to SCSCLogo.jar telling it to use it's smallest size.

Important Notes: As in the preceding example, when filling out the form, the *Input Arguments* for the Signed Jar Loader will always define where the application targeted to be executed is stored, its type, its path, etc. In the profile form, the "Application Type" must always be Jar since the Signed Jar Loader is a jar file. The

“Application Location” option is *always* a local file, since the Signed Jar Loader is stored locally, not on a web server. The “Launch on FastPak Startup”, “Application Priority”, and “Version Identifier” text may be selected as the user sees fit.

A screen capture of this is configuration is shown below:

The screenshot shows a dialog box titled "Add or Edit a FastPak Application Configuration". The "FastPak Reference Name" is "LocalSecureLogo". The "Application Type" is "Jar File" and the "Application Location" is "Local File". The "Launch on FastPak Startup" is "No" and the "Application Priority" is "Medium". The "Define Local Application" section shows the "Name" as "SignedJarLoader" and the "Path" as "/Users/smith/FastPak/JarApps/". The "Define Remote Application" section shows "Remote Name" and "Remote File Path" as "----- Not Applicable -----". The "General Application Information" section shows "Host Name" as "localhost", an empty "Version Identifier" field, and "Input Arguments" as "ost /Users/smith/SignedJarApps/SCSCLogo.jar 480". The dialog has "Help", "Cancel", and "OK" buttons at the bottom.

Once this application profile is completed, it will be ready for use with FastPak with the profile name “LocalSecureLogo.”

The Native Application Launcher

The Native Application Launcher allows FastPak to launch native applications on its hosting system. This launcher conforms to the FastPak Design Guidelines, thus it's a standalone application capable of running completely outside the FastPak environment. Unlike the Signed Jar Loader addressed in the previous section, this document will not discuss using this application outside the FastPak environment. The actual name of the binary is named "NativeAppLauncher.jar" and it's located in the "JarApps" directory of the FastPak installation.

When the Native Application Launcher launches an application, with the exception of any I/O streams the application may use to display information (standard output and standard error), the application exists completely outside the FastPak environment. Unlike a Java application, which will immediately terminate on a FastPak shutdown, a native application will continue running.

The Native Application Launcher is configured like any other Java application under FastPak, with the input parameters to the profile specifying the application to launch and its associated command line parameters. The screen shot below illustrates the profile for an instance of this module being configured for use under FastPak.

Add or Edit a FastPak Application Configuration

Assign FastPak Reference Name
FastPak Reference Name:

Application Type
 Jar File
 Flat File

Application Location
 Local File
 Remote File

Launch on FastPak Startup
 Yes
 No

Application Priority
 Low
 Medium
 High

Define Local Application
Name:
Path:

Define Remote Application
Remote Name:
Remote File Path:

General Application Information
Host Name:
Version Identifier:
Input Arguments:

To clarify the work needed to be performed, we'll go through this profile configuration for OS X (shown in the diagram), Linux/Unix, and Windows.

In all cases, the name of the profile will be called "OpenEditor" and it will open a text editor in the users home directory. For this example, we'll assume the user- ID is "smith" and the file to be open is named "Things to Buy.txt" on all platforms, and it's stored in the home directory. Refer to the preceding diagram for reference purposes as needed. The following will hold true for the examples we wish to configure:

Windows:

- **Windows Home Directory:** This will be "C:\Documents and Settings\smith"
- **Windows File to Edit:** "C:\Documents and Settings\smith\Things to Buy.txt"
- **Windows Editor Command:** Notepad

OS X:

- **OS X Home Directory:** /Users/smith
- **OS X File to Edit:** "/Users/smith/Things to Buy.txt"
- **OS X Editor Command:** open /Applications/TextEdit.app

Linux/Unix:

- **Linux Home Directory:** /home/smith
- **Linux File to Edit:** "/home/smith/Things to Buy.txt"
- **Linux Editor Command:** /usr/bin/gedit

When creating the profile for launching this native application, the following will hold true for all operating systems:

- **FastPak Reference Name:** This will be "OpenEditor" for all platforms.
- **Application Type:** This will always be a Jar file.
- **Application Location:** This will always be local.
- **Launch on FastPak Startup:** Set as you desire.
- **Application Priority:** Set as you desire.
- **Define Local Application:** Use the GUI to navigate to the file "NativeAppLauncher.jar" in the JarApps directory of the FastPak base installation.
- **Define Remote Application:** All fields here are not available for editing since this is a local application.
- **General Application Information:** *Host Name* is always set at "localhost"

The only changes that will occur that are platform specific will occur in the *Input Parameters* option under the configuration heading “General Application Information”. The following list shows the values that go into this field based on the operating system being addressed:

- **Windows:** notepad “C:\Documents and Settings\smith\Things to Buy.txt”
- **OS X:** open /Applications/TextEdit.app “/Users/smith/Things to Buy.txt”
- **Linux/Unix:** /usr/bin/nedit “/home/smith/Things to Buy.txt”

Important notes:

1. The name of the file to be opened (Things to Buy.txt) has spaces embedded in it, thus it and any preceding path components must be surrounded by double quotes. If the name of this file had been ThingsToBuy.txt (no spaces in the name) then in the OS X and Linux examples, the double quotes can be eliminated. However, on Windows, the path name preceding the file itself (C:\Documents and Settings\smith\) has spaces embedded in it as well thus the entire entry would still need to be put in double quotes even if the file was named ThingsToBuy.txt (“C:\Documents and Settings\smith\ThingsToBuy.txt”)
2. On OS X, the actual command is “open” and everything following it, including the name of the application is actually an input parameter. This is because OS X is actually a Unix variant, and the “open” command expects to see an application bundle (with a “.app” extension) as the first argument, with all other following entities being seen as input parameters to the application bundle. If you are using one of the OS X Unix level tools (such as “ls”) then the “open” command wouldn't be needed and won't even work. The full path for open on OS X is /usr/bin/open, for those interested.
3. The Linux/Unix version assumes the text editor is the X-Windows based text editor “nedit” and it's been installed in the /usr/bin directory. This will likely vary from OS implementation to implementation.

Once the profile is configured, it can be launched under FastPak just like any other FastPak profile. As is the case with all FastPak applications, whether they be Java applications or native applications, different profiles can be used to launch numerous applications. Additionally, if you wish to run an old Java application that uses a JVM that predates Java 1.4, it may be launched as a native application in this manner.

The Daemon Mode Auto Terminator

The Daemon Mode Auto Terminator is an application that uses the FastPakAPI.jar library to communicate with the FastPak kernel to automatically launch identified applications on start up and then, after a user specified period of time, terminate the Controller Thread and put FastPak into an “exit” state. FastPak's “exit” state allows all applications started under FastPak to continue running until all of them have completed. Once all applications have finished, the FastPak kernel terminates completely.

Previous releases of FastPak required the end user to manually terminate a FastPak instance in daemon mode by either using one of the supplied controller tools in the developers library or by killing the JVM. This made terminating a FastPak daemon somewhat tedious, especially to users that simply wanted to auto-launch a set of applications under FastPak at start up, let them run, and then have FastPak end automatically when all their applications had finished. The Daemon Mode Auto Terminator performs this function.

NOTE: The Daemon Mode Auto Terminator may also be used in GUI or Console modes *provided* the Controller Thread is running, however once the time-to-exit period is exceeded, all interfaces to FastPak, whether they be remote via a Controller Thread, or through a GUI or command line interface will be terminated and FastPak will operate in kernel-only mode until all applications have completed. After that, the FastPak kernel itself will terminate.

Configuring the Daemon Mode Auto Terminator for use with FastPak is easy. The name of the Daemon Mode Auto Terminator is “AutoTerm.jar” and it's in the “JarApps” subdirectory of the FastPak installation. AutoTerm.jar is configured as a local application in jar format. The only thing the person configuring this profile needs to worry about is the input parameters that will be fed to the AutoTerm.jar module.

The input parameters to the AutoTerm.jar application are as follows:

<hostname> <portNum> <timeoutMillis> [profile1 profile2 profileN]

Each of these entries can be described as follows:

- **<hostname>:** This is usually localhost
- **<portNum>:** This is the port number of the Controller Thread
- **<timeoutMillis>:** This is the amount of time the Controller thread is allowed to remain active. After this time has expired, FastPak terminates the Controller Thread, shuts down all GUIs and interfaces, and enters kernel-only mode, where it will continue to execute the launched applications until all of them have terminated.
- **[profile1 profile2 profileN]:** This is a list of the FastPak *PROFILESNAMES* (not actual application names) to be executed.

Example: We wish to configure an AutoTerm.jar module so that when FastPak starts up in daemon mode, it will automatically launch the applications associated with the profiles Analyzer, FourierAnalysis, and ActiveLogo, wait 10 seconds (10,000 milliseconds), shut down the Controller Thread, and then enter kernel-only mode and continue executing the applications until all have finished. After all applications are done, the FastPak kernel will terminate itself automatically.

A screen shot of this configuration is shown below:

The screenshot shows a dialog box titled "Add or Edit a FastPak Application Configuration". It contains the following fields and options:

- Assign FastPak Reference Name:** FastPak Reference Name:
- Application Type:** Jar File, Flat File
- Application Location:** Local File, Remote File
- Launch on FastPak Startup:** Yes, No
- Application Priority:** Low, Medium, High
- Define Local Application:** Name: Path:
- Define Remote Application:** Remote Name: Remote File Path:
- General Application Information:** Host Name: Version Identifier: Input Arguments:

Buttons at the bottom:

The entry for "Input Arguments" tells the AutoTerm.jar module that the host is the same machine (localhost), the port is 20001, the timed delay is 10000 milliseconds (10 seconds), and the FastPak profiles to automatically launch are Analyzer, FourierAnalysis, and ActiveLogo. Notice that in this case, the "Launch on FastPak Startup" option is "Yes", meaning that when FastPak starts it will automatically run this profile.

The ConfigTool

The ConfigTool is found in the base FastPak installation directory. The ConfigTool should only be used when FastPak is not running. FastPak has its own set of configuration tools available when it's up and running, and this configuration tool was extracted from them.

When started up, the ConfigTool will present the user with the following display:



The tool is used by selecting the option you want to perform from the list and clicking on the "OK" button. When done, click on the "Cancel" button. This GUI is really just an interface to the GUIs used in FastPak for configuring, editing, and deleting applications as well as editing the FastPak configuration settings or modifying FastPak's class path settings. Because of the redundancy in function with FastPak, users are asked to review the FastPak users manuals for details on configuring applications and FastPak settings, as this tool is really being provided as a convenience for developers.