FastPak for Java

A Technical Overview

Legal Information

All Software and Computer Systems Company, LLC logos are a trademark (TM) of Software and Computer Systems Company, LLC. *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 document 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 to customers by Software and Computer Systems, LLC 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

Terminology 6

A Summary of FastPak for Java Features and Strengths 8

A Summary of FastPak for Java Limitations 12

What FastPak Users Should Be Aware Of 14

An Overview of FastPak Performance 15

FastPak Components 24

Introduction

FastPak for Java, or more briefly, ***FastPak***, is an application loader for Java applications. ***FastPak*** is a tool designed to be implemented for end users by developers and administrators that fully understand what ***FastPak*** is capable and not capable of doing. Hopefully, this document will provide sufficient detail to make this clear. Readers should be aware that ***FastPak*** differs considerably from other application launching tools in that under ***Fastpak***, all applications are run under a single virtual machine instead of having one virtual machine running for each application. As will be seen this can improve overall system performance while saving system resources.

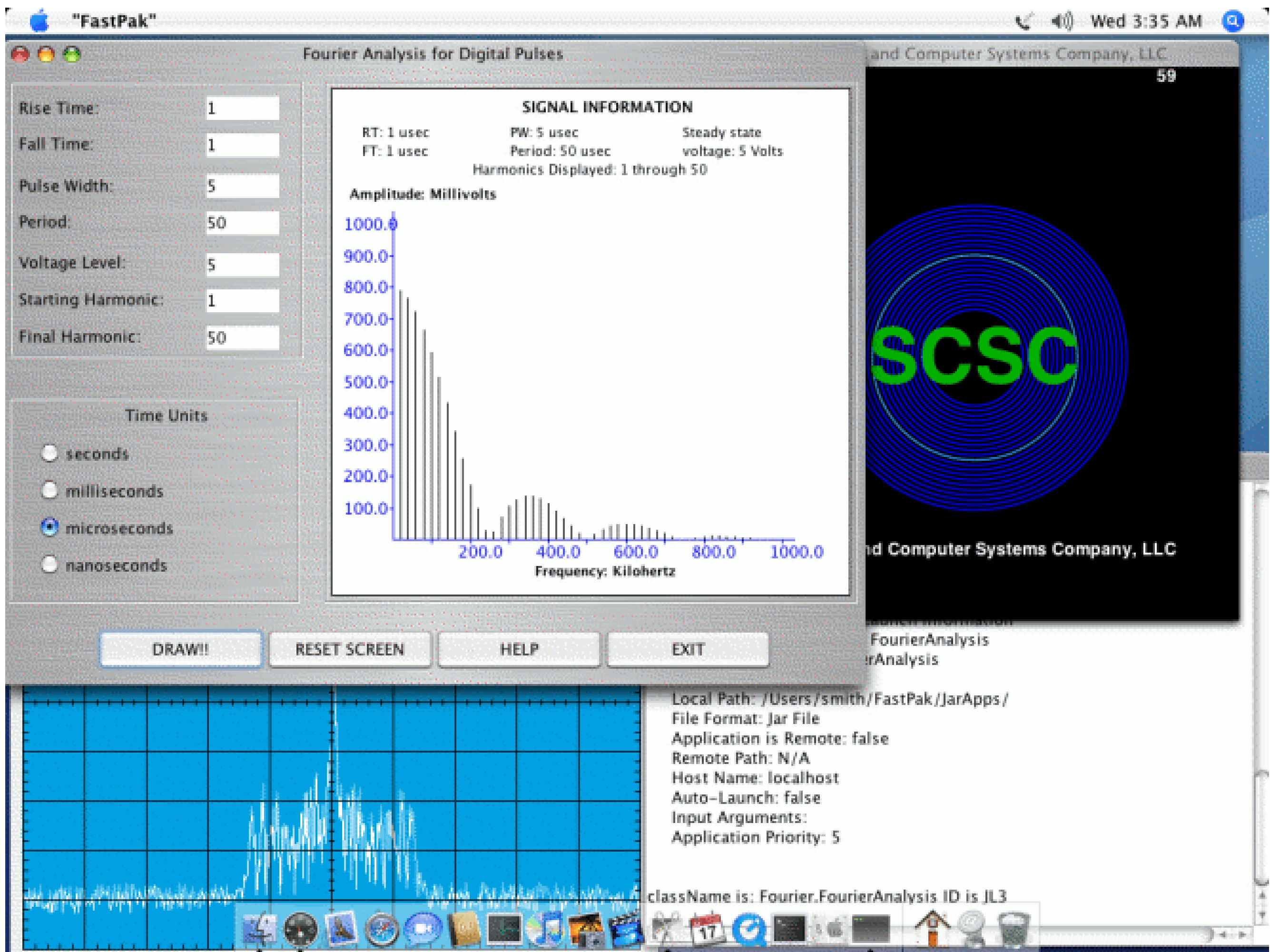

Figure 1. Three Java applications running under a singe JVM with ***FastPak***

The following sections of this document will provide a descriptive list of terms that may be useful to readers, identify ***FastPak*** features as well as its strengths and

limitations, point out characteristics of ***FastPak*** that users should be aware of, provide an overview of ***FastPak*** performance, and provide a list of ***FastPak*** components included in each ***FastPak*** distribution.

Terminology

When reading this document, the following list of terms and abbreviations will be used and defined as follows:

- **Application Profile:** An application profile is a file that is configured by a user, developer, or administrator to tell ***FastPak*** how to launch an application. This file allows the user to do such things as send the application command line parameters, over ride preassigned command line parameters, set the applications main thread priority with respect to other applications running under ***FastPak***, and automatically launch specified applications on ***FastPak*** start up, to name just a few. Details about the application profile are provided in the advanced users guide.
- **EDT**: This stands for the event dispatching thread. This is the thread in the JVM that handles all graphics updates and processing. If this thread is blocked waiting for something (a network connection is a good example) it can end up blocking all graphics processing.
- **Flat application:** A flat application is a group of class files that have been created using a Java compiler. These files are not archived, but rather exist in a directory structure, most often as a set of many individual class files. All references to the class files are absolute and require explicit references to the directories containing the class files if attempts are made to execute the flat application outside it's own directory structure. Most Java application launchers cannot launch flat applications, but ***FastPak*** can, provided they are configured under ***FastPak*** properly.
- **GUI**: This is an abbreviation for graphical user interface. This term may be used for operating system components as well as specific applications.
- **Jar application:** A “jar application” is a Java application that has been compiled into a series of class files and the class files are then archived into a single bundle with the Java archiving tool, “jar.” Users can often (but not always) launch such an application from an OS GUI or a command line without needing to supply any external links, references, library paths , etc. for the application to run. Jar applications under ***FastPak*** may also reference other jar archives as libraries provided their class paths are accessible to both the JVM and ***FastPak***. This document will not go into any of the specifics of creating a jar based application or how to use the “jar” tool.
- **Java application, or application:** This is an application that has been compiled using a Java compiler. In this context, the Java application, or just application for short, is one that has a main() method and is executed by loading it into a JVM which will then launch it. Unless otherwise stated, all references in this document to an application will be referring to a Java application. ***FastPak*** can run two types of Java applications, which will be referred to as a “jar application” and a “flat application.” Both of these are described in this section.

- **JDK:** This is an abbreviation for Java developers kit. This is a kit that includes the JRE and it's associated JVM (both previously described) as well as compilers, libraries and a host of tools used to build and assist in the build of a Java application. The release levels of a JDK and it's associated JRE are identical. **NOTE**: Applications built using a JDK can typically be run on a JVM of an equal or higher release level, but not the opposite. For example, if an application was compiled using JDK 1.2, then in most cases it will run problem free on a JVM associated with releases of 1.2 and above (which would presently be 1.3, 1.4, 1.5, and 1.6). However, an application compiled using a JDK of 1.5 would generally not work on earlier JVMs (in this case, 1.4 and earlier) unless the code was compiled to specifically generate code for older JVMs, which is typically not done.
- **JNI:** JNI stands for Java native interface. Applications using JNI are linking methods and/or libraries that are compiled using a native language such as C or C++ into a Java application. Use of JNI is platform specific and it's implementation varies considerably from one operating system/hardware configuration to another.
- **JRE:** This is an abbreviation for Java runtime environment. This is typically a package that is installed on a system that contains the corresponding JVM. Release levels of the two are similar if not identical. For example, a JRE release may be designated 1.4.2 and the corresponding JVM packaged in the release may report something like 1.4.2_09.
- **JVM**: This is an abbreviation for the Java Virtual Machine. Each JVM is seen by the hosting operating system as a resource intensive application
- **Launch time:** This term refers to the amount of time it takes for a given Java application to reach a fully operational state once started. For example, if a standalone jar application named “TestIt.jar” is launched from a terminal or console window with the command “java -jar TestIt.jar”, the launch time would be the amount of time it takes for the application to reach a fully operational state once the user hits the return/enter key after typing in the command. Launch time under ***FastPak*** is different because under ***FastPak*** the JVM is already in a fully operational state prior to application launch, and launch time will usually consist of the amount of time it takes for the user to select the desired application from a list using a mouse and then just launch the application. Launch times under ***FastPak*** are much faster than they are when launching applications with discrete JVMs.
- **SCSC**: This is an abbreviation for ***Software and Computer Systems Company, LLC,*** which is the company responsible for all aspects of ***FastPak for Java.***
- **SWT:** SWT stands for standard widget toolkit. This is a non-Java standard GUI implementation that ***FastPak*** currently does not support.

A Summary of *FastPak for Java* Features and Strengths

A high level summary of current ***FastPak*** features and strengths is described in the following list:

- **Extremely rapid application launching of applications –** With ***FastPak***, the JVM is already loaded so the delay associated with starting an application is minimal.
- ***FastPak* does not "take over" a system – *FastPak*** can run side by side with other Java applications.
- **Generally, any normal Java application can run under *FastPak* –** Assuming an application is properly installed for use under ***FastPak***, it should work without any problems. However, there are certain conditions where a developer may need to make modifications to code for it to co- exist with other applications running under ***FastPak***. These are detailed below under the section titled **A Summary of *FastPak for Java* Limitations** as well as in the developers manual.
- ***FastPak* can run applications hosted on a secure web server – *FastPak*** is capable of running applications stored on a web server in a jar format.
- **Saves memory, threads, and other system resources when multiple Java application are running simultaneously – *FastPak*** uses a single JVM to launch applications, thus the overhead typically associated with launching numerous applications, each with its own discrete JVM is greatly reduced.
- **Developers can create their own, custom GUI s for use with FastPak instead of its default GUI -** If ***FastPak*** is put into its daemon mode (described a few points below) custom GUI s can control ***FastPak*** by accessing the *Controller Thread* (described below) via a connection to localhost.
- ***FastPak* is easy to use –** When an application is configured to use ***FastPak***, it's simply a matter of selecting it from a list of available applications and running it. Additionally, if the application uses command line parameters, the user can opt to override them and enter new command line parameters on demand.
- ***FastPak* is very easy to configure –** All configuration is done via a GUI menu, and the configuration files themselves are simple properties files which can be edited in a text editor if needed.
- ***FastPak* provides a** *Controller Thread* **that allows remote control and management of *FastPak* – *FastPak*** makes use of a thread called the *Controller Thread* that allows access to a running ***FastPak*** instance on a local or remote machine (called a "targeted instance of ***FastPak***") . This thread, which uses sockets for communications, can be used to perform the following tasks:

    1. Run an application as configured on the targeted instance of ***FastPak***

2. Run an application with new or appended command line parameters on the targeted instance of ***FastPak***
3. Shows a list of the applications configured to run on the targeted instance of ***FastPak***
4. Dump the configuration for the targeted instance of ***FastPak***
5. Deactivate an application currently in the list of available applications on the targeted instance of ***FastPak***
6. Reactivate a previously deactivated application on the targeted instance of ***FastPak***
7. Dump a listing of high level threads running under the targeted instance of ***FastPak***
8. Stop the *Controller Thread* on the targeted instance of ***FastPak*** (WARNING: forces all networked I/O on the *Controller Thread* to stop, including the connection to the machine that issued the command!!)
9. Issue an "exit" command, that will cause the targeted instance of ***FastPak*** to terminate once all applications running under it are complete
10. Shutdown the targeted instance of ***FastPak*** and all applications running under it immediately
11. Turn disk logging on and off for the targeted instance of ***FastPak***
12. Provide a list of help commands (very basic commands that follow the *FastPak Protocol*) to the application accessing the targeted instance of *FastPak*
13. If *FastPak* is in daemon mode (see below), the *Controller Thread* can be used by developers to create a custom GUI for users to control *FastPak*

- ***FastPak* provides an auto launch capability –** ***FastPak*** can automatically launch applications specified to do so in their application profiles when ***FastPak*** is started.
- ***FastPak* can be run in a daemon mode –** ***FastPak*** provides a mode where it can be launched without any user interface at all. This is particularly useful for users that have a stable group of applications they wish to launch and have no real need for a user interface. It can also be used to interface with a custom GUI for ***FastPak*** via the *Controller Thread*.
- ***FastPak* can be run in a console mode –** ***FastPak*** provides a console mode that may be of value when evaluating programs that are problematic and are throwing exceptions that may be "swallowed up" using the GUI mode. It is also useful for accessing remote ***FastPak*** instances for management.
- **Applications can be bundled under different *FastPak* instances –** The

number of ***FastPak*** instances that can be run under a given system is limited only by its resource and CPU limitations. Since Java applications can run more efficiently under ***FastPak***, it's possible to run several instances of ***FastPak*** simultaneously, provided their configurations don't step on one another (port number conflicts for the *Controller Thread* and directory naming redundancy are the most common problems). With this in mind it's possible to dedicate several ***FastPak*** instances to specific tasks and modes of operation.

- ***FastPak* allows full network access for applications it launches –** ***FastPak*** does not interfere with any network connections an application attempts to make. An exception to this occurs if one application imposes a security manager that prevents such access.
- ***FastPak* can bring life back to older, slower hardware -** Not everyone has an expensive, high end machine with lots of memory, and these are typically the systems that find themselves limited to the number of Java applications they can run simultaneously if each application is run with its own dedicated JVM. Since each ***FastPak*** instance uses only a single JVM, such users may find they are able to run a number of Java applications simultaneously that had been impossible before.
- **Application priorities can be set with respect to one another –** Each application run under ***FastPak*** can be assigned a priority which can reduce or increase the amount of processing time dedicated to each application launched under ***FastPak***.
- **Supports multiple file formats –** ***FastPak*** can launch locally hosted flat applications and applications that have been archived into jar files as well as launching remote applications in a jar format hosted on a secure web server.
- **FastPak provides limited thread monitoring –** ***FastPak*** can report on high level threads running under the system as well as show garbage collection of completed applications (the latter is available only when in GUI mode)
- **Logging on and off on demand –** Logging of FastPak activity to a hard disk can be turned on and off on demand.
- **Can mark and save existing logging information on the GUI and save it to a file –** When using ***FastPak's*** GUI mode, logging text can be marked, edited, and saved to a disk file.
- **The** *Controller Thread* **can be brought up and down on demand in mode –** The *Controller Thread* can be brought up and down as needed and need not be running all the time. This applies only to the console and GUI modes. The daemon mode can only shut down the *Controller Thread,* and in doing so will terminate connections to its peer.
- **Multiple instances of the same applications may be run with identical, different, or no command line parameters –** ***FastPak*** isolates each application it launches into its own environment, thus multiple instances of an application can be launched without worrying about one application interfering

with another. There are exceptions to this rule as noted below under the section called **A Summary of *FastPak* Limitations**.

- ***FastPak* provides limited support for applications using JNI –** ***FastPak*** can run applications using JNI, however the support is limited to only one application running per ***FastPak*** instance, and even this can be highly system dependent. See **A Summary of *FastPak* Limitations** below.

A Summary of *FastPak for Java* Limitations

A high level summary of current ***FastPak*** limitations is described in the following list:

- **JVM must be 1.4 or greater-** ***FastPak*** will not operate properly under a JVM preceding release 1.4
- **It may be necessary to modify code to handle System.exit() (or equivalent) calls –** Calls, directly or indirectly, to System.exit() and equivalent calls must be replaced by code that terminates applications without terminating the JVM. This is not a requirement but a strong recommendation. Older GUI programs often used calls to System.exit() (or equivalent) to end an application. This was done because helper threads would not terminate properly on JVM releases prior to 1.4. In other words, to end an application the JVM itself was abruptly brought down by this call. This bug was fixed in all Java releases of 1.4 and above. The bug ID was **4030718.** Complete documentation for this bug fix may be found under the Java API documentation under the heading *AWT Threading Issues.* As of the writing of this document (April, 2007) this appears in the documentation for Java releases 1.4, 1.5, and 1.6 under the heading *Implementation-dependent behavior* in the release notes. Details on workarounds for this problem can be found in the developers manual
- ***FastPak* is not a secure product –** ***FastPak*** is intended to be a desktop application operating within a secure network. It is up to developers and administrators to implement any and all security features for any applications running under ***FastPak***. Please read the document titled **Security Warnings Regarding *FastPak for Java*** for details. A copy of this may be seen from the **SCSC** web site at http://www.scsc-online.com or it can be downloaded from the sites downloads section.
- **SWT is currently not supported –** Any applications developed using SWT to provide GUI support for an application to be run under ***FastPak*** are **not** ***FastPak*** compatible. It is possible to get some SWT applications to appear to work properly, but please trust us on this, they will not work properly!!
- **JNI Limitations –** Problems with applications using JNI can appear when one instance of a Java application using a JNI library is running and then another instance of the same (or different) Java application is brought up that attempts to load the same JNI library. This generally limits the use of such applications until one is complete and all references have been garbage collected. Additionally, JNI applications using screen I/O may be problematic. This is highly hardware and operating system dependent. **SCSC** recommends that JNI applications be avoided for use under ***FastPak*** unless very thoroughly tested.
- **FastPak can "live on" under certain OS X versions -** On OS X, *Panther* and *Jaguar* release, ***FastPak*** will "live on" and continue to run even after it's been terminated when launched in its default mode on this operating system, which makes use of OS X's menu bar. *This is not a* **FastPak** *bug, but a bug with those releases of the operating system.* There are two possible work arounds for this problem. The first is to go ahead and use the application as designed and

instruct users that when done using ***FastPak*** to click on the lingering icon in the task bar and quit the application. The second is to run ***FastPak*** as a console application via the console application launch script described in the advanced users manual. In the case of the latter, the OS X menu bar will not be a part of ***FastPak*** or any applications it launches, but it will terminate properly.

- **Avoid applications that deliberately kill threads –** This applies primarily to very old Java applications that kill threads and any applications that make deliberate use of ThreadDeath.

What *FastPak* Developers and Users Should Be Aware Of

As an application loader, ***FastPak*** has some characteristics that users, developers, and administrators may need to be aware of. These characteristics are summarized in the list below:

- **Respect the EDT-** Good Java GUI design respects the EDT. This is generally considered a good programming practice for standalone applications, but if a lot of programs using active graphics simultaneously are expected to be used under ***FastPak*** this can be critical. If you are running several GUI applications simultaneously under ***FastPak*** and one of the applications blocks the EDT, it will also block all graphics activity associated with all other GUI applications running under ***FastPak***. Further details regarding this may be found in the developers manual.
- **Not all exceptions thrown by a Java application are reported –** ***FastPak*** catches and handles all exceptions it can propagate. An application running under ***FastPak*** is seen really as an extension of the ***FastPak*** kernel. If an application throws an uncaught exception during its execution and it's in an execution path of the kernel at the time it's thrown, ***FastPak*** may catch the exception and can sometimes lock up. This can be avoided by catching all exceptions that may occur in your Java applications, and, better yet, fix all bugs and working environments so that exceptions are not thrown for applications in the first place. Most ***FastPak*** exception problems are relatively rare, and in most instances the exception will be reported to the GUI s logging area or as console/terminal messages in the console or daemon modes. This problem occurs more frequently using the GUI mode.
- **Applications that require very specific versions of Java should be avoided -** This occurs most frequently for users running legacy applications that require the use of methods (such as killing threads) that are no longer supported. Such applications are generally rare and should be run outside a ***FastPak*** environment.

An Overview of ***FastPak*** Performance

FastPak performance can be evaluated in two ways: resource use and application performance. For this discussion, resource testing will be limited to real memory consumption, CPU loading, and thread use, whereas application performance will focus on the amount of work an application can perform in a given amount of time.

Test Applications - The applications involved in the tests are bundled with ***FastPak*** as demonstration programs. The following text about each application describes what they do:

- **Analyzer.jar** *Analyzer.jar* is a Java application in “jar” format that is shown in both figures 2 and 4 (below) in the lower left corner of each figure. This program simulates a high speed sweep of a spectrum analyzer and is part of an old **SCSC** product called ***Java JWaveScope***. This particular version creates the data points for thousands of sweeps of an amplitude modulated signal as it would appear on a spectrum analyzer. Each screen display draws one of these sweeps, then overlays the corresponding grid and analyzer text data. This program operates at full speed. There are no fixed delays or sleeps put into the program. It is very graphics intensive. Every time the program has processed the entire set of arrays comprising the simulated signal data, a counter is incremented allowing the program to display how many times it's processed the data at any point in time during its run. This application is not interactive and does not accept command line input.
- **ActiveLogo** *ActiveLogo* is a flat application. It is shown in the upper right corner of figures 2 and 4. *ActiveLogo* is based on the actual **SCSC** Logo, but it's “active.” In this program, a sequence of blue circles are drawn, but one of them is turquoise, and then the green **SCSC** lettering as well as a counter and full company name in white lettering are placed on the screen. The turquoise circles propagate from the inner most circle out on each image update, similar to a radar display. Every time an iteration is complete, a counter in the upper right corner of the images is incremented. This application does have delays deliberately put in it, thus the program periodically yields control to other applications running under ***FastPak***. Because of the delays, this program should be considered to be moderately graphics intensive. This application is not interactive but it will accept input/command line data to modify the size of the logo image based on the screen display (a numeric entry of 640, 800, or 1024 as a reference to screen sizes of 640 by 480, 800 by 600, and 1024 by 768, respectively).
- **FourierAnalysis.jar** This application is an application that generates a diagram illustrating the amplitude of spectral components for a repetitive digital signal. The user enters the data and then presses the “DRAW!!” button. After pressing

this button, the program calculates the amplitude for each spectral component associated with the signal, and then draws these components in the frequency domain on the display. This program basically works in “burst mode,” meaning the only time it's really loading the CPU is when it's doing its calculations. Depending on the data input, this typically takes anywhere from a few microseconds to tens of seconds. Once done, the application basically sits idle and waits for more user input. This application is shown in both figures 2 and 4 in the upper left hand corner in its idle state. This application is interactive but it will not accept command line parameters. Although most people don't spend their time analyzing digital signals, this application is probably the most typical of a Java application because it is interactive and it has the potential to severely load the CPU

Test Platform - All tests were done using an Apple Mac Mini with 512Mb of memory and a Power PC G4 1.25 GHz single processor. The operating system was Mac OS X 10.4.8 and the version of Java was 1.5.0_06.

Resource Testing - To examine resource consumption, let's first take a look at some Java applications being executed using the “old way” of launching applications via a discrete JVM for each application. Focus will then turn to resource use on the same set of applications running under ***FastPak*** with a single JVM.

The image on the next page shows the three Java test applications described above running simultaneously. In this particular example, all three have been launched manually, and each has it's own JVM.

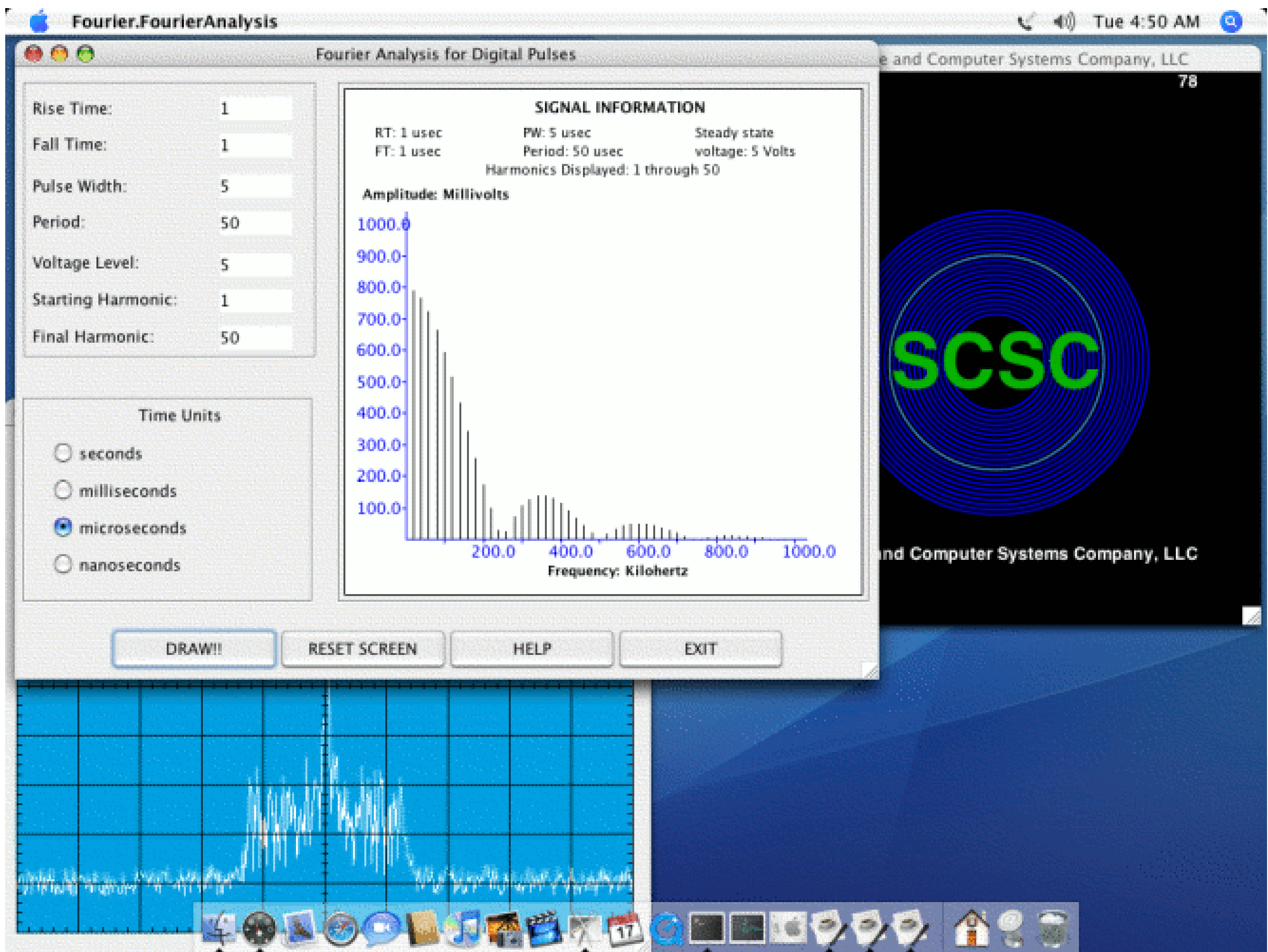

Figure 2. Three Java applications running under three discrete JVMs.

In the image above, you will notice three little coffee cup icons in the lower right portion of the image that indicate that three standalone JVMs are running – one for each of the three applications being executed (the “coffee cups” are an OS X feature and will not show up on other operating systems)

The image on the next page shows the resources that each instance shown above is using as viewed from OS X's **Activity Monitor** program:

Process ID	Process Name	% CPU	# Threads	Real	Virtual Memory	Kind
16847	JWaveScopeAnalyzer	54.10	14	27.57 MB	363.50 MB	PowerPC
16845	logo.ActiveLogo	20.30	14	26.77 MB	362.81 MB	PowerPC
16848	Fourier.FourierAnalysis	0.00	14	24.88 MB	361.52 MB	PowerPC
16851	Activity Monitor	2.50	2	10.93 MB	117.11 MB	PowerPC
16828	Finder	0.00	3	10.19 MB	117.95 MB	PowerPC
16854	Grab	0.10	3	6.96 MB	113.32 MB	PowerPC
16827	SystemUIServer	0.00	2	6.71 MB	110.50 MB	PowerPC
16837	Terminal	0.00	4	6.34 MB	110.93 MB	PowerPC
16809	loginwindow	0.00	3	3.84 MB	82.29 MB	PowerPC
16836	UniversalAccessApp	0.00	1	3.63 MB	84.84 MB	PowerPC
16825	Dock	0.00	2	3.05 MB	78.14 MB	PowerPC
16824	NetBlockadeHelper	0.00	1	2.95 MB	69.80 MB	PowerPC

Figure 3 Resource consumption for the applications show in Figure 2 above.

The diagram above has each application launched under its discrete JVM highlighted in blue. The data in the columns, "%CPU," "# Threads," and "Real" above will be used and compared to the same corresponding data for ***FastPak*** below.

The image on the next page shows the exact same set of applications running, but in this case, they have been launched under ***FastPak***, not manually.

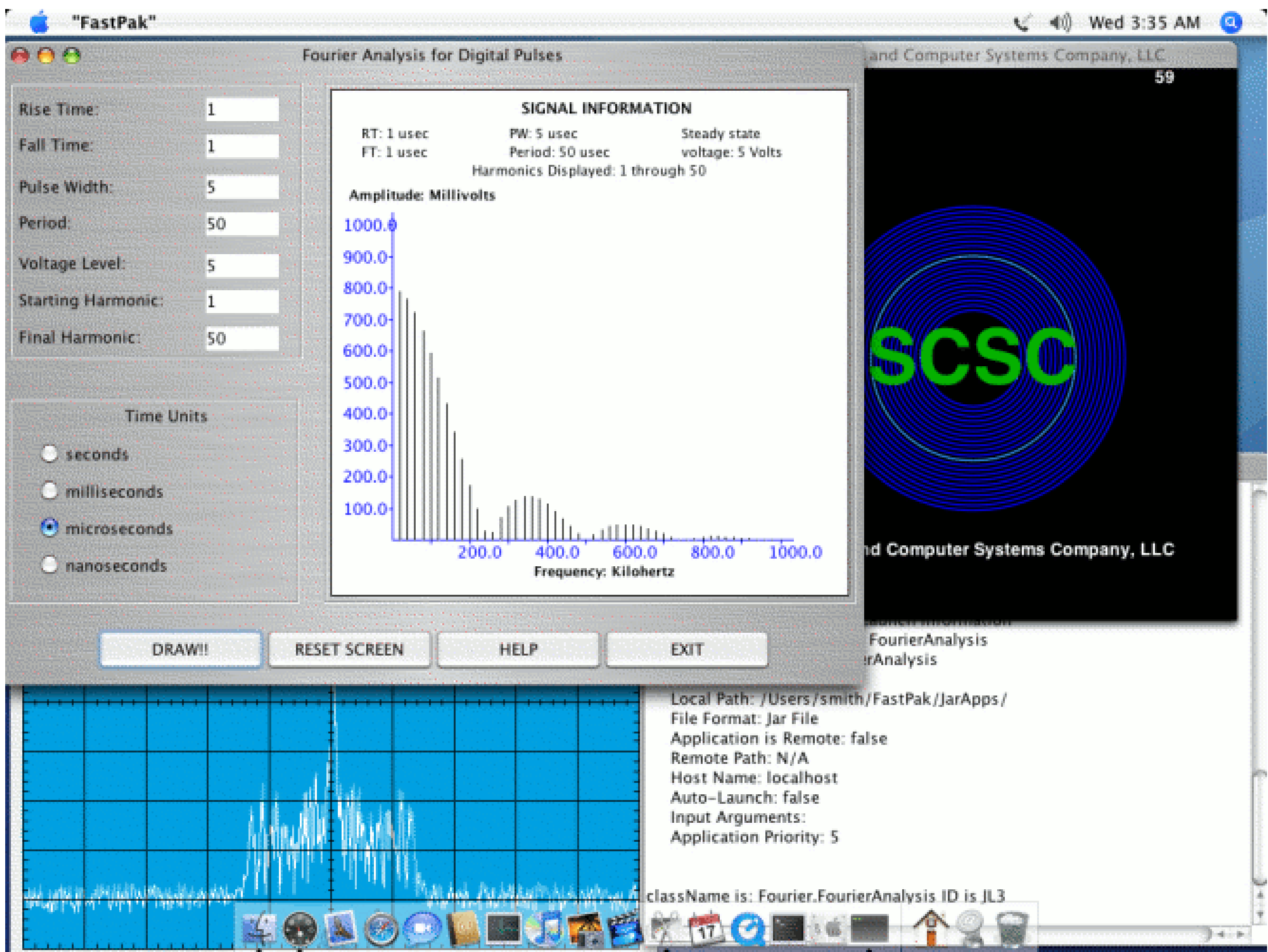

Figure 4. Three Java applications running under a single JVM using ***FastPak***.

The most noticeable differences between this image and the screen shot of the applications running in the previous example is that in this case you can see part of the ***FastPak*** GUI in the lower right corner and the three “coffee cup” icons are now replaced by a single icon showing a generic application, which is located just to the left of the icon for the home directory shown as a little house.

The screen shot on the next page shows resource usage of the exact same applications running under ***FastPak***:

Process ID	Process Name	% CPU	# Threads	Real	Virtual Memory	Kind
202	"FastPak"	74.30	18	40.59 MB	373.16 MB	PowerPC
191	Finder	0.00	3	10.61 MB	119.58 MB	PowerPC
204	Activity Monitor	2.00	2	9.21 MB	113.71 MB	PowerPC
190	SystemUIServer	0.00	2	6.76 MB	110.47 MB	PowerPC
206	Grab	0.10	3	6.68 MB	113.20 MB	PowerPC
76	loginwindow	0.00	3	3.86 MB	87.68 MB	PowerPC
197	UniversalAccessApp	0.00	1	3.61 MB	84.82 MB	PowerPC
188	Dock	0.00	2	3.02 MB	78.02 MB	PowerPC
74	ATSServer	0.00	2	2.99 MB	51.48 MB	PowerPC
187	NetBlockadeHelper	0.00	1	2.92 MB	69.80 MB	PowerPC
186	kansard	0.00	2	2.64 MB	70.01 MB	PowerPC
179	mdimport	0.00	3	2.29 MB	38.70 MB	PowerPC

Figure 5. Resource consumption for the applications show in Figure 4 above.

In this case, all three applications are running under a single JVM, which is associated with the instance of ***FastPak*** itself.

Using the data we've collected for both test conditions above, we will now compare the two different modes of operation just examined:

- **Using 3 discrete JVMs to launch the 3 applications.** This information is from figures 2 and 3 above. In this case, each instance is using 14 threads as shown under the "# Threads" column in figure 3, thus the total thread consumption is 3 X 14 = 42 threads. Under the "Real" heading for real memory consumption, we see figures of 27.57Mb, 26.77Mb, and 24.88Mb, for a total memory consumption of 79.22Mb. Finally, the percentage of CPU usage is obtained under the "%CPU" column. For the three standalone applications shown this is 54.1%, 20.3%, and 0.0% for a total CPU usage of 74.1%. (*FourierAnalysis.jar* shows 0.0% because it's an interactive application and is not being actively used.)
- **Using *FastPak* to launch the 3 applications.** This information is from figures 4 and 5 above. The row highlighted in figure 5 is all that needs to be read, since ***FastPak*** is executing the three applications under a single JVM. As shown, the total memory consumption is 40.59Mb, the number of threads used is 18, and the percentage of CPU time used is 74.3%.

Comparing the two test cases above, the following conclusions can be made:

1. **Running applications under *FastPak* uses considerably less memory.** The total memory used for ***FastPak*** and the three applications running under it is 40.59Mb as compared to 79.22Mb for the three applications launched "the old way." ***FastPak*** is using roughly half the memory to get the same thing done.

2. **Running applications under *FastPak* uses far fewer threads.** The total thread consumption of ***FastPak*** and the three applications running under it is only 18 threads, as compared to 42 threads for the applications launched “the old way.” ***FastPak*** is getting the same job done with nearly one- third the thread resources.
3. **Running applications under FastPak does not increase CPU consumption considerably.** The CPU consumption for ***FastPak*** and the three applications running under it is 74.3% as compared to 74.1% launching the Java applications “the old way.” If the reader wishes to take those figures literally, then ***FastPak*** requires less than 1% more CPU time, which is probably well within the accuracy error of the measuring tool. In other words, ***FastPak's*** CPU consumption is virtually identical to that of launching the applications “the old way.”

Something that cannot be shown in the information above is the launch times. All ***FastPak*** launches are less than 500 milliseconds on the test platform, and typically less than 50mSec. When launching the applications using discrete JVMs (aka “the old way”), by the time the third application was manually launched it took roughly 7 seconds to launch.

Application Performance - Since it should now be clear to the reader that ***FastPak*** unquestionably reduces resource load on a system using it as opposed to launching applications via corresponding discrete JVMs, we now move to the issue of addressing how much work may be accomplished for applications running under ***FastPak*** as opposed to running each application under discrete, dedicated JVMs. In other words, if a few applications are launched under ***FastPak***, over a fixed period of time (5 minutes, for example) will the applications running under ***FastPak*** get as much done as the exact same applications running with standalone, dedicated JVMs?

To evaluate this, a test similar to that described above was performed, but this time the increment counters in *Analyzer.jar* and *ActiveLogo* were monitored over a period of five minutes (*FourierAnalysis.jar* provides no such counters since it responds to user input and its appearance in the test is used to represent what amounts to a null load). The test was done using individual JVMs for each of the three applications which were launched via a shell script, run for five minutes, and at the end of five minutes, the counter values on *Analyzer.jar* and *ActiveLogo* were recorded. This test was then replicated using ***FastPak***'s auto launch feature (a feature that allows ***FastPak*** to automatically launch designated applications at start up), the applications were allowed to run for five minutes, and then the counter data on *Analyzer.jar* and *ActiveLogo* was recorded. A screen shot of the test done under FastPak is shown below at the point where the test just ended. Note the counter on *Analyzer.jar* indicates a value of 474, while the counter on *ActiveLogo* indicates a value of 109 (this screen capture is from the actual test).

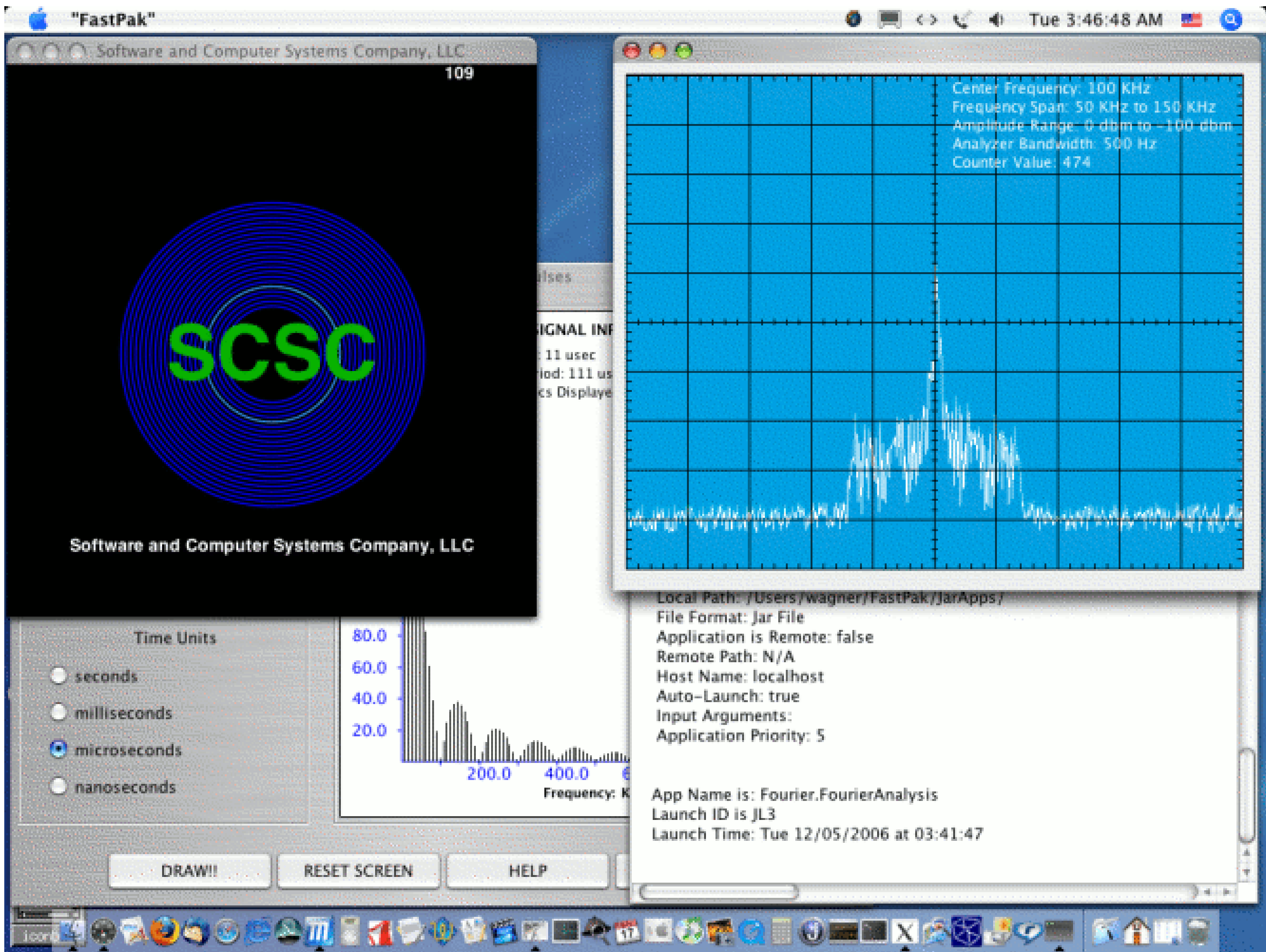

Figure 6. Application performance test using ***FastPak***.

Test results are summarized in the following table:

Application	**Repetition Count Using *FastPak***	**Repetition Count Using Discrete JVMs**
ActiveLogo	109	109
Analyzer.jar	474	475

Comparing the two test cases above the following conclusions can be made:

In this test case, the performance of applications under ***FastPak*** is practically identical to applications launched using discrete JVMs. The only noticeable difference exists in the number of repetitions for *Analyzer.jar,* which shows 475 for discrete JVM launches as opposed to 474 for ***FastPak***. ***FastPak*** is thus showing a decrease in

performance for *Analyzer.jar* of only 0.2%, which is virtually insignificant.

Up to the minute test data for various version of ***FastPak for Java*** on all supported operating systems may be found by visiting the **SCSC** web site at:

http://www.scsc-online.com

FastPak Components

All copies of ***FastPak for Java*** will include the following list of components:

- **The *FastPak for Java* application itself.** Generally there will be several of these compiled for each version of the JVM that is released . At present (December, 2006) JVMs for 1.4 and 1.5 are supported. This will change as more versions of Java become available.
- **The *Transfer Utility*.** The ***Transfer Utility*** is a tool to aid developers in distributing ***FastPak*** applications to end users. In many cases, particularly Linux and OS X, the base ***FastPak*** directory will have a user name embedded in it's home path. This utility will modify all the application profile properties files associated with those applications so they can be easily transferred to that users machine without a need for the end user to manually modify such files.
- **The Developers API.** This is a library in ".jar" format along with accompanying documentation in PDF, HTML, or text format and example programs.
- All Manuals. All manuals are included in PDF format. In some cases, HTML versions may be included as well.

Releases of ***FastPak for Java*** are operating system dependent. The list above is thus only partial, and most distributions will also include applications and scripts specific to that platform.