SmartFrog User Manual

<u>Table Of Contents</u>

1   Introduction

This manual is aimed at those wanting to run the SmartFrog framework. It assumes that user is working with the binary distribution of the SmartFrog framework.

It describes:

- Directory structure for the binary distribution

- Starting and managing the daemons

- Setting up and managing the security infrastructure

- Running the first example

- Configuring dynamic code downloading

- The manual does not cover details of how to write SmartFrog applications i.e. how to use the SmartFrog notation, the component model, etc. The reader is directed to the SmartFrog Reference Manual and its accompanying Javadoc for this information.

2   SmartFrog Framework

2.1   Directories and Files in the binary distribution
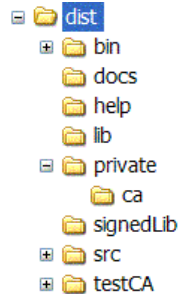
SFHOME = INSTALLATION_DIRECTORY/dist

The distribution expanded into SFHOME contains a number of directories and files. These are

- docs – a directory which includes a number of documents describing the framework and examples. It also includes the Javadoc for the framework and the accompanying packages.

- bin – contains scripts for starting and stopping the daemons and applications; these may be used as is, or modified as required to suit local conditions. It may be desirable to include SFHOME/bin in any path environmental variable used by the shells.

- lib – contains all the jar files provided in the distribution.

- signedLib – initially empty, it is used for placing jar files signed during the security initialization process. Not all jar files in lib will be signed as not all are compatible with the security infrastructure.

- testCA – the files needed to provide a dummy certification authority for testing security for those who do not have access to OpenSSL.

- private – a directory which will contain files for the security infrastructure. Note that before any re-installation of the infrastructure, this directory may need to be saved to preserve security information. For more details see section 4.12.

- src – the source code for some examples showing a few of the capabilities of the system. These examples have been precompiled and are provided in sfExamples.jar in the lib directory.

- build.xml[1] – a file used by ant to re/compile the examples, to generate java docs for the examples and to control the security build process.

- rmitargets – a file that contains the classes that need to be compiled with "rmic" compiler.

---

[1] ANT 1.6.1 (http://ant.apache.org/) and Java JDK 1.4.2 have to be installed and properly configured to be able to re-compile the examples, generate javadocs for the examples and run the security build process.

2.2   Classpath

It is recommended to start SmartFrog using packaged scripts, in this case there will be no need to set the class path; it is set within the scripts. However, if the user does not use the scripts, he/she has to set the classpath. The classpath settings are determined by the use of security and/or remote class loading.

- No security, local classes – set the class path to include every jar file in the lib directory [For reference see packaged script setClassPath]

- Security, local classes – set the class path to include every jar file in the signedLib directory; these are only present after the security infrastructure has been built and not all of the packages are available in this case.

- With remote class loading – set the class path to minimally include one of the *lib/smartfrog.jar* or *signedLib/smarfrog.jar* files, depending on whether security is enabled or not. This must be local since otherwise SmartFrog cannot be started and the class loading initiated. In addition, the property *org.smartfrog.codebase* must be set to contain URLs to the remaining jar files. This should be done even with the process running the server component.

More details of how to do this are in the section on remote class loading.

3   Using The Framework

SmartFrog has many different ways of being started; all provided by the Java class SFSystem that is fully documented in the SmartFrog reference manual. Users may use either this class or the scripts provided in the SFHOME/bin directory. This section only covers the use of these scripts.

The scripts provide some of the more common use models for the SmartFrog framework. They are provided primarily to give users a simple way of experimenting with the framework and are useful for running the examples provided with the distribution.

All the scripts assume the following:

- security features are not required (example scripts for enabling security are provided in the *security* sub-directory of bin).

- the SFHOME variable has been set to refer to the installation directory.

It is, of course, possible to use remote class loading and security together. However, it should be noted that it is left to users to create appropriate scripts for their specific environment, those provided are merely examples demonstrating what must be done. The full set of environmental variations is too great for a generalized script set to be provided.

There are two concepts to understand as part of the underlying control of the SmartFrog system. The first of these is the SmartFrog Resource Reference (SFREF). This is a URL to a description file to deploy or otherwise use. The second is an SmartFrog Action Descriptor (SFACT), which is used to indicate to SmartFrog an action to take. These are now described in more detail.

3.1   SmartFrog Resource References

Throughout the SmartFrog system, including on the command line, references to SmartFrog resources (i.e. files) may be given in a number of ways:

- as a URL to the file;

- as a relative or absolute path name to a file;

- as a path to a resource in a jar file on the classpath or code base.

In this last case, the reference should be given as a path relative to the root of the package structure within the jar file, i.e. without the leading /. In most cases this leading / is removed by the code, but there may be some instances where this is not so.

In the following descriptions of the scripts, a reference to such a resource is referred to as an SFREF.

3.2   SmartFrog Action Descriptor

An action descriptor is is used on the command line to describe a certain type of action that will be carried on by the daemon. An action has a number of ":" separated fields, most of which may be left blank in many cases.

The format is:

```
NAME:ACTION:SFREF:SUBREF:HOST:PROCESS
```

The semantics of the fields are defined as follows:

3.2.1   NAME

The name is a single word, or a SmartFrog reference in which case it must be surrounded by quotes. The name has one of two interpretations depending on the action to be taken (see next field).

In TERMINATE, DETACH, DETaTERM, the name is a reference to the component on which to apply the action.

In DEPLOY, the name is treated like a placement and the name is split into two: all but the last part is a reference to another component and the last (or only) part is the name which will be given to the deployed component within that referenced component. If the component is not a ProcessCompound, the component is also made the parent of the deployment.

In all cases, the NAME is resolved relative to the process compound of the HOST and PROCESS specified by the appropriate fields.

When a name is not provided, it indicates the process compound of the host and process defined in the HOST and PROCESS fields.

(Note, in this release, the DEPLOY action is restricted to a simple word and may not be not a placement reference. This will be fixed in a subsequent release.)

Examples:

```
foo

"HOST localhost:foo"
```

3.2.2   ACTION

This field defines the action to be taken on the named component

- DEPLOY a component or application.

- TERMINATE a component or application.

- DETACH a component from its parent.

- DETaTERM detach and terminate a component from its parent.

3.2.3   SFREF

The SmartFrog description (if needed) to be used by ACTION. It is a SmartFrog Resource Reference see 3.1. It needs to use quotes when the reference is using ":". Currently this is only required for a DEPLOY action and is ignored otherwise.

Examples:

```
/home/sf/foo.sf

"c:\sf\foo.sf"
```

3.2.4   SUBREF

When the SFREF is parsed and resolved, the result is a component description containing a number of attributes. In the "sf" language, this is the contents of the sfConfig definition. Under normal circumstances, it is this whole definition that is used for the deployment, but occasionally, for testing purposes perhaps, it is useful to specify some single subcomponent. Under these circumstances, the name of this attribute, or a reference to a deeply nested application, may be provided. This is the SUBREF.

Examples:

```
foo

"fist:foo"
```

3.2.5   HOST

host name or IP from where to resolve the name. If HOST is not present, the process name is ignored and the process executing is used. If you want to refer to another process, other than the executing one, on the local host, "localhost" should be used and the appropriate PROCESS name used.

Examples:

```
foo.hpl.hp.com

127.0.0.1
```

3.2.6   PROCESS

process name from where to resolve the name. When empty it defaults to "rootProcess".

3.2.7   Examples

These examples show the use of the action descriptors for different purposes.

Example 1: Deploy a description in local daemon

```
counterEx:DEPLOY:org/smartfrog/examples/counter/example.sf::localhost:
```

Example 2. Terminate local sfDaemon

```
rootProcess:TERMINATE:::localhost:
```

or

```
:TERMINATE::::localhost:
```

Example 3: Deploy "counterToSucceed" from counter/example2.sf

```
counterEx3:DEPLOY:org/smartfrog/examples/counter/example2.sf:"testLevel1:coun
terToSucceed":localhost:
```

4   The Command-Line Scripts

4.1   sfDaemon

sfDaemon is a script that creates a root process compound (see reference manual). As a root process, it starts a registry to enable it to be located by other SmartFrog systems, by default this registry is on port 3800.

At start up, the daemon (and every sub-process created by it) read the following files from the directory ${SFHOME}/bin/ .

- default.ini : to define JVM system properties; this may be edited to alter those it uses. It may not be used for properties that affect security or code loading – it is read after these are already initialized.

- default.sf : containing some standard components that should be deployed in all daemons and sub-processes; this may be edited as required. By default, it contains a component to start a display window for the process (shows stdout, stderror and provides additional information). This description registers itself using the name 'sfDefault'.

To stop the daemon, see sfStopDaemon.

4.2   smartfrog

smarfrog wraps SFSystem. SFSystem that is fully documented in the SmartFrog reference manual. Smartfrog behaves like sfDaemon except that the daemon created is not a root process, just a simple daemon which can be used to run SmartFrog applications which are run locally or do not require other SmartFrog systems to locate them.

The command line parameters are as for sfDaemon, none, one or more than one of each is permitted:

- -a SFACT: SmartFrog Action Descriptors (SFACT) that the daemon has to execute.

- -f SFREF: file that contains a set of SmartFrog Action Descriptors (SFACT).

- -e: the daemon will terminate after finishing the deployment. Similar functionality can be achieved with:

```
-a :TERMINATE::::
```

- -?: usage and help information

Only the default.ini file is read by default: editing the script will be required to change this.

smartfrog script is the most flexible command and can be used to replace all other scripts except sfDaemon.

4.3   sfRun

SfRun is similar to sfDaemon except that the daemon created is not a root process, just a simple daemon which can be used to run SmartFrog applications which are run locally or do not require other SmartFrog systems to locate them using the registry (HOST links).

The command line parameters are:

- SFREF : the SFREF where the text is to be found to be parsed and deployed.

- -e: to terminate the daemon after if finishes the deployment. Optional.

Only the default.ini file is read by default: editing the script will be required to change this.

This command is equivalent to:

```
smartfrog -a :DEPLOY:SFREF::: -e
```

4.4   sfStart

sfStart is intended as a simple way of getting applications launched into a network of SmartFrog daemons, even if one is not running on the local host. The command line is

```
sfStart hostname appname SFREF
```

with the following interpretation:

- *hostname* : is the name of a host on which the root daemon is running through which deployment should be done; the name *localhost* may be used to indicate that the daemon is running locally.

- *appname* : the name by which the application should be known in that daemon – this is important for sfTerminate.

- SFREF : the SFREF where the text is to be found to be parsed and deployed.

This command is equivalent to:

```
smartfrog -a appname:DEPLOY:SFREF::hostname: -e
```

4.5   sfStop

sfStop is now obsolete and replaced by sfTerminate. Please see sfTerminate.

4.6   sfStopDaemon

sfStopDaemon is the counterpart to the sfDaemon command, in that it can terminate every application that was deployed in a daemon and then the daemon itself.

The command line is:

```
sfStopDaemon hostname
```

with the following interpretation:

- *hostname* : is the name of a host on which the root daemon is running through which deployment should be done; the name *localhost* may be used to indicate that the daemon is running locally.

This command is equivalent to:

```
sfTerminate hostname rootProcess
```

    or

```
smartfrog -a rootProcess:TERMINATE:::hostname: -e
```

    or

```
smartfrog -a :TERMINATE:::hostname: -e
```

4.7   sfDetachAndTerminate

sfDetachAndTerminate is a script that detaches the named component from the SmartFrog system and then terminates it. This detachment and termination does not affect the other components.

The command line is:

```
sfDetachAndTerminate hostname appname
```

with the following interpretation:

- *hostname* : is the name of a host on which the root daemon is running through which deployment should be done; the name *localhost* may be used to indicate that the daemon is running locally.

- *appname* : the complete name by which the component should be known in that daemon.

This command is equivalent to:

```
smartfrog -a appname:DETaTERM:::hostname: -e
```

4.8 sfTerminate

sfTerminate is a script that terminates the named component from the SmartFrog system. The termination does not affect the other components.

The command line is:

```
sfTerminate hostname appname
```

with the following interpretation:

- *hostname* : is the name of a host on which the root daemon is running through which deployment should be done; the name *localhost* may be used to indicate that the daemon is running locally.

- *appname*: the complete name by which the component should be known in that daemon.

This command is equivalent to:

```
smartfrog -a appname:TERMINATE:::hostname: -e
```

4.9 sfParse

sfParse is different from the other scripts in that it does not invoke the SFSystem class to deploy the application. It is merely a tool to help in learning the notation or to debug an application description when unexpected effects are observed.

The sfParse script parses and fully resolves the description and then prints the resultant expanded and processed description to the console.

If the verbose flag is set, the sfParse script carries out the same process, but at each step, the abstract syntax tree on which the various tools operate is printed to the console. Consequently, the complete description (with include files expanded) is printed, it is printed again after type resolution, again after place resolution and so on, for all the required phases.

The command is

```
sfParse [-v] [-q] [-r] [-R] {[-f] filename | SFREF}
```

where the optional flag "–v" sets the verbose mode and the "SFREF" is a SmartFrog file.

All the command line parameters are:

- -v: verbose, prints every parser phase.

- -q: quiet, no phase printed. Overwrites –v. Only errors are reported.

- -r: show status parsing report.

- -R: create status parsing report in html file named: <filename>_report.html.

- -f filename: file with a list of SmartFrog descriptions (SFREFs) to parse.

- SFREF: reference for a SmartFrog file or description.

- -?: shows command line usage info.

Example:

```
sfParse -r org/smartfrog/examples/counter/example.sf
```

4.10   sfManagementConsole

sfManagementConsole starts a console to show information and manage the applications deployed in a particular daemon in real time. No information is cached. Each application is shown in a different tab panel.

The command line parameters are:

- -h HOSTNAME : host name where the daemon is running.

- -p PORT_NUMBER : port that the daemon is using.

- -w WINDOW_POSITION: where to show the console window. Valid values are: N (north), NE, NW, S (south, SE, SW, C (centre), E (east), W (west).

- -r: shows an extra panel that connects to the daemon directly. Note that rootProcess component has a link to itself. Experts only.

- -?: shows command line usage info.

Some important features:

Button bar:

Reload Panels: The system is rescanned for an updated view.

Using the right button of the mouse there are some management actions available:

1. Right click in left panel:

Terminate Component: Terminates the component selected. The termination can be NORMAL or ABNORMAL.

Detach and Terminate Component: Detaches the selected component from the SmartFrog system and then terminates it. It uses a NORMAL termination.

Detach Component: Detaches the selected component from the SmartFrog system.

2.   Right click in right panel over the attributes table:

Remove Attribute: Remove the selected attribute from the selected component.
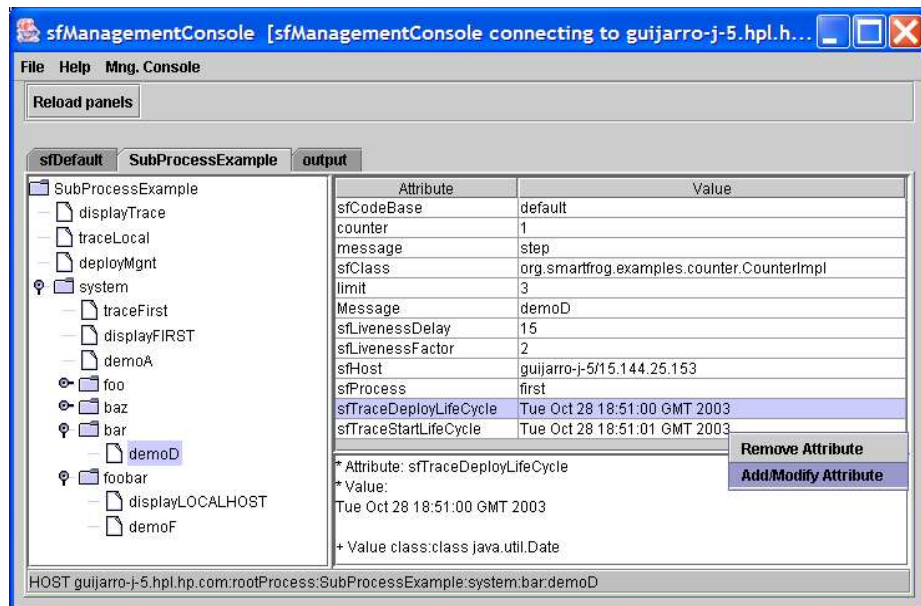
Add/Modify Attribute: Add a new attribute and its value or to modify an existing one. Next is shown a picture of the dialog used to add\modify attributes with the attribute types that it supports.
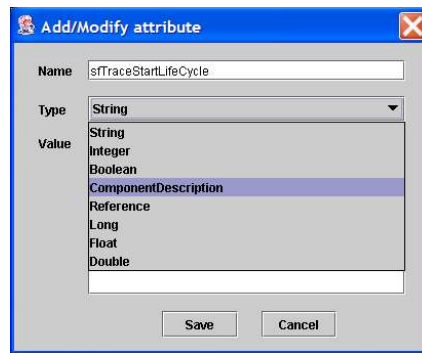
At the bottom of the management console, a status bar shows the canonical name for the selected component. This information can be copied using "crtl+C" after selecting it.

This console can also be started from every daemon display using the option: Help -> SF Mng Console. After selecting that option, a dialog is displayed to choose a host to connect.
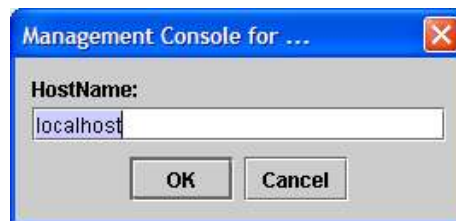
4.11   sfVersion

sfVersion prints out the current SmartFrog version number to the console. The command line syntax for the script is

```
sfVersion
```

4.12   Typical Usage Models

The two models for deploying applications supported by the scripts are follows:

- Locally, in a single process for testing or simply because no distribution is necessary. This is achieved by using the sfRun script.

- In a network of machines, each running a root daemon and each started by the sfDaemon script. The applications are launched using sfStart and perhaps terminated using the sfTerminate script.

5   Re- Compiling the examples

5.1   Running the build scripts

The examples are built by ant build scripts provided as part of the release. These scripts are contained in the file SFHOME/build.xml. Once ant is installed, these may be run from the SFHOME directory. Running the command *ant* on its own generates a short help message that describes all the scripts available. These scripts are now described in more detail.

5.2   Re-Compiling examples

Compiling or recompiling the examples is as simple as running

```
ant build
```

in the SFHOME directory.

This builds all the code contained in SFHOME\src\org\smartfrog\examples and creates a jar file called sfExamples.jar in SFHOME\lib.

To compile code existing in different location a modification of SFHOME\build.xml is required.

5.3   Creating Java Docs for the examples code

Creating the java docs for the examples code is as simple as running

```
ant jdocs
```

in the SFHOME directory.

This creates a new directory called SFHOME\jdocs where the java docs are placed. Once ant finishes, check SFHOME\jdocs\index.html to browse the result.

6   Enabling Security

The security infrastructure is designed to provide an environment within which a SmartFrog system may be securely deployed and applications executed. A complete description of the principles behind the security system is given in the reference manual.

6.1   Creating The Security Infrastructure

The principle behind the security build environment is as follows:

- create a certification authority for code signing on a central host, thereby defining the community;

- sign all jar files compatible with the security infrastructure (the main restrictions are to do with class loading) on that host;

- create host identities on the central host;

- securely distribute identities and signed files to each host;

- ensure operating system protection of security credentials;

- start SmartFrog with security enabled.

6.1.1   Running the security build scripts

The security is built by ant build scripts provided as part of the release. These scripts are contained in the file SFHOME/build.xml. Once ant is installed, these may be run from the SFHOME directory. Running the command *ant* on its own generates a short help message that describes all the scripts available. These scripts are now described in more detail.

6.1.2   Creating a certification authority

This must be done once only for each security community required. It is usually done on a secure central host and the information distributed using secure means to the execution hosts in the system. Note that each security domain will require its own copy of the SmartFrog directories so as not to confuse the identities, keys, etc.

To create a new certification authority requires OpenSSL to be installed. Once this is done, creating a new CA is as simple as running

```
ant initCA
```

in the SFHOME directory.

This creates the authority for signing code that will be checked for by each host as code is loaded. If a new certification authority is created, no existing signed code will be loaded by new daemons, and existing running daemons will no longer be able to interact with them.

6.1.3   The Test Certification Authority

If OpenSSL is not present, yet there is a need to test code using a CA, the SmartFrog distribution comes with a dummy CA, contained in the directory testCA. Using this CA provides a fully capable security infrastructure, but it should not be considered secure for real deployment.

The command to create the dummy CA is

```
ant dummyInitCA
```

run from the SFHOME directory. This should not be done if the initCA script has been run.

6.1.4   Resetting the authority

At times, it might be useful to reset all the information regarding a certification authority. This may be done by running the script

ant cleanCA

run from the SFHOME directory. Users must now run one of the above scripts.

6.1.5   Signing the jar files

Once the certification authority has been created, all the jar files may be signed and placed into the signedLib directory by running the command

```
ant signJars
```

from the SFHOME directory. Currently only a few of the jar files are signed. If a user jar file is required to be signed, in addition to those distributed with SmartFrog, the signJars script in the build.xml file may be modified. This is very straightforward.

6.1.6   Creating new daemon identities

Each root daemon in the security domain must have a separate identity, this identity being cryptographically strong. The command

```
ant newDaemon
```

creates a new identity. These identities are created in the SFHOME/private directory, each in a separate subdirectory named hostxxx, where xxx is a unique number generated from the current time and date. As many of these as required may be created. The hostxxx directory conatins two files:

- mykeys.st – the keystore for the host that contains certificates, trust assumptions and keys.

- SFSecurity.properties – the default security properties for the host, this may be edited if required, and contains a random password that unlocks the keystore.

The testCA contains some pre-created identies that are copied into the private directory during the script dummyInitCA.

6.2   Preparing Hosts

Each host must be prepared to run in a secure mode. This means that several simple steps must be taken:

- All the signed jar files must be downloaded and placed in the appropriate location, such as the *signedJar* directory; minimally this must include the smartfrog.jar file.

- The host identity must be securely transferred to the host (by floppy, over a secure network or using some secured protocol). The two files from the selected hostxxx directory, the security properties and the key store, must be placed the *private* directory on the specific host and this directory MUST be made read-only and accessible only to the user account running the daemon.

- The start-up scripts must be protected to ensure that as SmartFrog is started the security is indeed enabled.

- If an .ini file is used during startup, this is read before the security system is initialized and consequently it is read without checking for signatures. It should therefore be read only from a secured source, such as the private directory. Note that security properties may not be set within the .ini file.

- All SmartFrog files and class files MUST be read from signed and sealed jar files otherwise exceptions will occur. This includes the iniSFFile. All application code must therefore be packaged, sealed, and signed.

${SFHOME}bin/security directory contains secure version of sfDaemon, sfStart and sfTerminate. If security is enabled only these set of scripts should be used.

7   Remote Class Loading

7.1   The Model

SmartFrog provides an integrated remote class loading system that is compatible with the security system, though it also works without security. It is slightly more restrictive than it might be because of the need to be secure, in that all the remote resources must be in Jar files.

A web server can be used to serve jar files to requesting daemons. All other daemons are then configured to use the server to access all jar files except the core smartfrog.jar that must be local to bootstrap the SmartFrog system.

Currently, it is only possible for a jar file to be loaded from a single place, and this location must be hard coded in the start-up scripts.

7.2   Enabling Class Loading Without Security

All daemons that wish to make use of the class loading must be initialized at start-up to access the server. This is done by setting the *org.smartfrog.codebase* system property to be a list of space-separated URLs to jar files.

This is done in the normal way (either using –D), and is set as in the following example:

```
-D org.smartfrog.codebase=http\://aHost..com\:8080/sfExamples.jar\ http://…
```

Use [Sample scripts for dynamic class loading](#) if you want to use this feature. All the batch files and shell scripts can be found in SFHOME/bin/classloader.

7.3   Enabling Class Loading With Security

[Sample scripts for Dynamic Class loading with security on](#). All the batch files and shell scripts can be found in SFHOME/bin/security/classloader.

8   Running First Example

8.1   Local, single process deployment

The simplest model for a SmartFrog description to be launched is to start a SmartFrog process directly running the application, without any intention of providing interaction with other SmartFrog systems around. To do this, use the *sfRun* script. To try this out, run the command line:

```
sfRun org/smartfrog/examples/arithnet/example1.sf
```

This should generate the following text on the screen.

```
SmartFrog 3.00.000_alpha
(C) Copyright 1998-2003 Hewlett-Packard Development Company, LP
Warning: SmartFrog security is NOT active
```

To terminate the process, do so using the ctrl-C key.

A full explanation of this example is given in the reference manual, but roughly speaking, two component are created, a printer and a generator, the generator sends a message to the printer every few seconds and printer prints that.
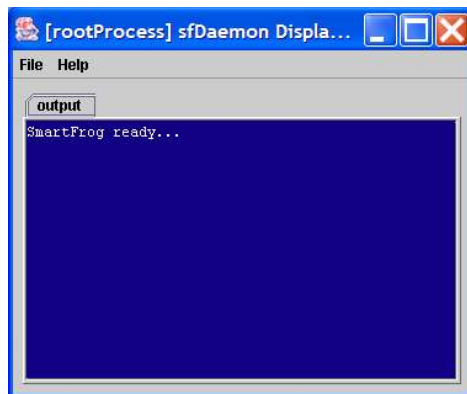
8.2   Into a single daemon

Now a slightly more complex model of deployment will be used. Firstly, a daemon will be started on a host – say the localhost – and one or more applications will be launched into this daemon.

First, start the daemon by running the command line

```
sfDaemon
```

This should start the daemon which should open a window into which it will write all output.

Next, in a different shell, launch the application into the daemon with the following command.

```
sfStart localhost example1 org/smartfrog/examples/arithnet/example1.sf
```

The example should now be running in the daemon, writing its information on that screen and the *sfStart* command should have terminated. The first two command line parameters are the host containing the daemon and a name by which this application instance should be known on that host – the use for this will be shown later.

Now we can start many applications at once, into the same daemon. Type the following

```
sfStart localhost example2 org/smartfrog/examples/arithnet/example2.sf
sfStart localhost example2 org/smartfrog/examples/arithnet/example3.sf
```

As each is launched, it should start printing on the screen of the daemon. Each is distinguishable by the differing printer names and the different messages output.

Once they are finished, the same lines can be typed again to launch the applications again. The only thing to note is that each application currently running must have a unique name (i.e. example1, example2, etc). This name may be any string and may be reused after the application using that name has terminated.

While an application is running, run the following command

```
sfTerminate localhost example1
```

replacing the example1 name with whatever application should be stopped.

8.3   Remote host

Install the SmartFrog system onto a second host - say "foo.com". Start the daemon in the same manner as above, using the sfDaemon command.

Repeat the sfStart commands, and sfTerminate if desired, with the hostname replacing "localhost", for example

```
sfStart foo.com example1 org/smartfrog/examples/arithnet/example1.sf
sfTerminate foo..com  example1
```

9   Appendix

9.1   Sample scripts for class loading without security

9.1.1   sfDaemon.bat

```
@echo off
setlocal
if defined SFHOME goto continue1
  if exist "%cd%\sfDaemon.bat" cd ..
  set SFHOME=%cd%
  cd bin
:continue1
if NOT (%1)==() GOTO usage
set CLASSPATH=%SFHOME%\lib\smartfrog.jar;%SFHOME%\lib\sfServices.jar;%SFHOME%
\lib\sfTestCases.jar;%CLASSPATH%
rem set CLASSPATH=%SFHOME%\lib\smartfrog.jar;%CLASSPATH%
set SERVER=localhost:8080
rem Please edit codebase if you have any other jar file in webserver
set CODEBASE="http://%SERVER%/sfExamples.jar"
rem set CODEBASE="http://%SERVER%/sfExamples.jar http://%SERVER%/
sfServices.jar"
java -Dorg.smartfrog.sfcore.processcompound.sfProcessName=rootProcess -Dorg.
smartfrog.codebase=%CODEBASE% -Dorg.smartfrog.iniFile=%SFHOME%
\bin\default.ini -Dorg.smartfrog.iniSFFile=%SFHOME%\bin\default.sf
org.smartfrog.SFSystem %1 %2 %3 %4 %5 %6 %7 %8 %9
GOTO end
:usage
echo Insufficient argument(s) to use sfDaemon
echo Usage: sfDaemon
:end
endlocal
```

9.1.2   sfStart.bat

```
@echo off
setlocal
if defined SFHOME goto continue1
  if exist "%cd%\sfStart.bat" cd ..
  set SFHOME=%cd%
  cd bin
:continue1
if (%1) == () goto usage
if (%2) == () goto usage
if (%3) == () goto usage
set CLASSPATH=%SFHOME%\lib\smartfrog.jar;%SFHOME%\lib\sfServices.jar;%SFHOME%
\lib\sfTestCases.jar;%CLASSPATH%
set SERVER=localhost:8080
rem Please edit codebase if you have any other jar file in webserver
set CODEBASE="http://%SERVER%/sfExamples.jar"
rem if exist "%SFHOME%\jre\bin\java.exe" set path=%SFHOME%\jre\bin
rem call %SFHOME%\bin\setClassPath
java -Dorg.smartfrog.iniFile=%SFHOME%\bin\default.ini -Dorg.
smartfrog.codebase=%CODEBASE% org.smartfrog.SFSystem -h %1 -n %2 %3 -e
GOTO end
:usage
echo Insufficient arguments to use sfStart
echo Usage: sfStart HostName ApplicationName URL
:end
endlocal
```

9.2   Sample scripts for class loading with security

9.2.1   SfDaemon.bat

```
@echo off
setlocal
if defined SFHOME goto continue1
  if exist "%cd%\sfDaemon.bat" cd ..
  set SFHOME=%cd%
  cd bin
:continue1
if defined SFPRIVATE goto continue2
  set SFPRIVATE=%SFHOME%\private
:continue2
if NOT (%1)==() GOTO usage
set CLASSPATH=%SFHOME%\signedLib\smartfrog.jar;%SFHOME%
\signedLib\sfServices.jar;%SFHOME%\signedLib\sfTestCases.jar;%CLASSPATH%
set SERVER=localhost:8080
rem Please edit codebase if you have any other jar file in webserver
set CODEBASE="http://%SERVER%/sfExamples.jar"
java -Djava.security.policy==%SFHOME%\private\sf.policy -Dorg.
smartfrog.sfcore.security.keyStoreName=%SFPRIVATE%\mykeys.st -Dorg.
smartfrog.sfcore.security.propFile=%SFPRIVATE%\SFSecurity.properties -Dorg.
smartfrog.codebase=%CODEBASE% -Dorg.
smartfrog.sfcore.processcompound.sfProcessName=rootProcess
org.smartfrog.SFSystem %1 %2 %3 %4 %5 %6 %7 %8 %9
GOTO end
:usage
echo Insufficient argument(s) to use sfDaemon
echo Usage: sfDaemon
:end
endlocal
```

9.2.2   sfStart.bat

```
@echo off
setlocal
if defined SFHOME goto continue1
  if exist "%cd%\sfStart.bat" cd ..
  set SFHOME=%cd%
  cd bin
:continue1
if defined SFPRIVATE goto continue2
  set SFPRIVATE=%SFHOME%\private
:continue2
if (%1) == () goto usage
if (%2) == () goto usage
if (%3) == () goto usage
set CLASSPATH=%SFHOME%\signedLib\smartfrog.jar;%SFHOME%
\signedLib\sfServices.jar;%SFHOME%\signedLib\sfTestCases.jar;%CLASSPATH%
set SERVER=localhost:8080
rem Please edit codebase if you have any other jar file in webserver
set CODEBASE="http://%SERVER%/sfExamples.jar"
java -Djava.security.policy==%SFHOME%\private\sf.policy -Dorg.
smartfrog.sfcore.security.keyStoreName=%SFPRIVATE%\mykeys.st -Dorg.
smartfrog.codebase=%CODEBASE% -Dorg.smartfrog.sfcore.security.propFile=%
SFPRIVATE%\SFSecurity.properties org.smartfrog.SFSystem -h %1 -n %2 %3 -e
GOTO end
:usage
echo Insufficient arguments to use sfStart
echo Usage: sfStart HostName ApplicationName URL
:end
endlocal
```