CERN
Engineering Data
Management System
for
Detectors
and
Accelerato**R**

EDMS Document No.

CERN Div./Group or Supplier/Contractor Document No.

IT

Date: July 20, 2004

BASIC SMARTFROG COMPONENTS AUTOMATED SOFTWARE INSTALLATION

Abstract

Some basic SmartFrog components needed for automated software installation; download files, install rpm's, generation of parameterized configurations files and error handling in sequence.

Prepared by :	***Checked by :***	
Andreas Braathen IT Andreas.Braathen@cern.ch		

Table of Contents

1. INTRODUCTION

1.1 GERERAL ABOUT SMARTFROG

SmartFrog is a technology for describing distributed software systems as collections of cooperating components, and then activating and managing them. It was developed at HP Labs in Bristol, in the UK. The core SmartFrog framework is released under LGPL.

SmartFrog consists of a language for
describing component collections and component configuration parameters,
and a runtime environment which activates and manages the components to
deliver and maintain running systems.

More information about SmartFrog can be found on http://www.smartfrog.org.

1.2 OBJECT

The object of this project is to create basic SmartFrog components for automated software installation. These tools will help in the development, installing and testing of LCG software.

1.3 MILESTONES

Create 5 different tools in SmartFrog. These tools should work by it self and together with the other tools.

These tools should be designed and implemented.

1.Downloader	Download files through HTTP and the normal file system. The files that are downloaded are specified in a configuration file.
2. RPMInstaller	Uses the UNIX/Linux command rpm to install rpm in the system.
3. VarInit	Implement the ability to create an ASCII file with text that could be parameterized through SmartFrog
4. UndoInterface	Interface that should be implemented to undo
5. ManageSequence	An extension of the Sequence class. Uses the UndoInterface to rollback in case of an error.

All tools are described in more depth in the chapter **Tools** and examples of how a SmartFrog script would use these tool in chapter **Examples**.

1.4 TIMETABLE

Date	Tool
15.06 – 18.06	Getting to know SmartFrog
21.06 – 25.06	Downloader
28.06 – 02.07	RPMInstaller
05.07 – 09.07	VarInit
12.07 – 16.07	UndoInterface
19.07 – 23.07	ManageSequence
25.07 – 30.07	Clean up

The tools should be designed, implemented and tested in this time frame.

1.5 DOCUMENTATION

Documentation is done in the source code, JavaDoc created from the source code and this document.

Problems and important notes about how these tools work are covered in the chapter **Tools**.

1.6 REQUIREMENTS

All source code are written in Java and SmartFrog's script language. Since SmartFrog is written in Java this makes the code platform independent.

1.7 TESTS

All tests are done on CERN Linux. All tools, except RPMInstaller, should work on any operation system with at JVM (Java Virtual Machine).

2. WORK FLOW OF AUTOMATED INSTALLATION

2.1 INTRODUCTION

This is an example how you could use the different components in an automated installation.

The scenario is to install some RPM files on a local computer. The RPM files are located on a server.

2.2 DESCRIPTION OF THE INSTALLATION

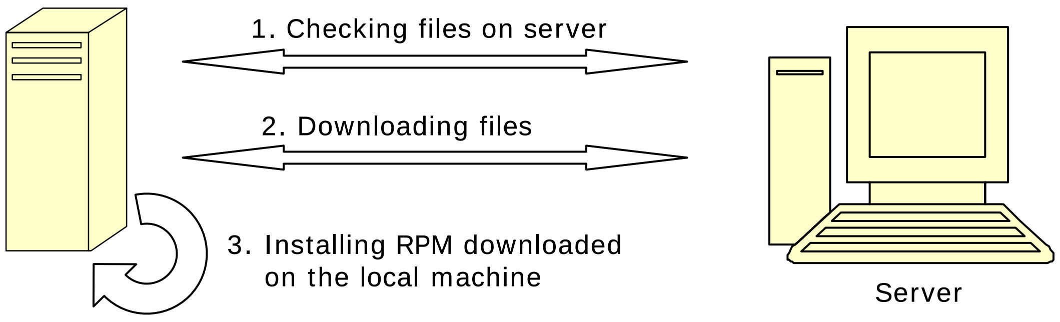
The installation is done through the ManageSequence components. If there is an error the component would undo what it tried to do.

Inside the ManageSequence component the other components would be run.

If there is an error in any of the components all the previous components would undo their action.

The Sequence would be:

1. Check if the files could be downloaded from the server
2. Download the files from the server.
3. Install the RPM files downloaded on the local machine.


1. Checking files on server
2. Downloading files
3. Installing RPM downloaded on the local machine
Server

How this scenario could be implemented using the SmartFrog script language is showed in the chapter **Examples**.

3. TOOLS

3.1 INTRODUCTION

All the tools described in this chapter are written in Java, and implement an interface used by SmartFrog. This interface implements Prim and Runnable. These interfaces can be found in the documentation about SmartFrog (http://www.smartfrog.org).

Examples of a SmartFrog file for all of these tools are described in chanter **Examples**.

3.2 DOWNLOADER

Short description:

Copy files over via either HTTP or normal file system to a defined location.

Depending libraries:

- Common IO module in the Jakarta Project (jakarta.apache.com)
- CERN`s Undo interface for SmartFrog tools (ch.cern.openlab.smartfrog.undo.Undo)

Package:

ch.cern.openlab.smartfrog.download:

- Download (Interface)
- DownloadImpl

Input variables:

• *protocolConfig*	which protocol used to read the configuration file, either http, file or none. None if you don't use a configuration file.
• *config*	location of the configuration file (path).
• *protocolDownload*	which protocol used to read the configuration file, either http, file or none. None if you don't use a configuration file.
• *host*	start location where the files should be downloaded from.
• *tmpDir*	location where the files are downloaded to.
• *stopOnError*	if true the download will stop on the first error, if false will just give an error en continue to download.
• *test*	mode not to download, but just to check if everything would have been downloaded with this configurations.
• *debug*	if true it will print out some extra debug information.
• *md5*	if true it will look for md5 checksum after the name of the file in the configuration file and this checksum would be compared with the checksum on the downloaded file.

At least the variables **host** and **tmpDir** have to be set in the script file used by SmartFrog to use this tool.

Programmers note:

The reason the tool uses the Common IO module from the Jakarta Project is that the standard Java implementation to read from an InputStream and sends the data to an OutputStream that write to a file is slow. The Jakarta implementation is more than twice as fast on a fast connection. This implementation is equally or faster than a wget command in Linux systems.

3.3 RPMINSTALLER

Short description:

Try to install RPM files on the system (must be a Linux system)

Depending libraries:

- CERN`s Undo interface for SmartFrog tools (ch.cern.openlab.smartfrog.undo.Undo)

Package:

ch.cern.openlab.smartfrog.rpmInstall

- Install (Interface)
- InstallImpl

Input variables:

• *RpmLocation*	location of the start folder where the RPM files are located.
• *rpmBin*	location rpm command used. Must be specified if the command rpm is not in PATH.
• *rpmOptions*	options used to the rpm command when installing.
• *force*	if true, the rpm command will use the force option. This means that the RPM will be installed by force.
• *rpmUndoOption*	options used to the rpm command when uninstalling. Only used when this tool is undone.
• *debug*	if true it will print out some extra debug information.

At least the variable **rpmLocation** must be set in the script file used by SmartFrog to run this tool.

Programmers note:

This tool only tries to run the command rpm with options to install. If this fails it will print the error to the user. Because of so many different versions of RPM there are no sentient to do this.

It is also possible to use the apt-get command to install rpm’s, but this tool is not standard installed with the CERN Linux.

3.4 VARINIT

Short description:

Implement the ability to create an ASCII file with text that could be parameterized through SmartFrog.

Depending libraries:

- None

Package name:

ch.cern.openlab.smartfrog.varInit

- VarInit (Interface)
- VarInitImpl

Input variables:

• *protocol*	which protocol used to read the input, either http, file or none. If the input is just is text set in the directly in the infile parameter the protocol should be none.
• *infile*	either the location of the file that contain the text being fixed or the text directly.
• *outfile*	location where the output should be stored

All the variables must be set in the script file used by SmartFrog to run this tool.

The variables in the SmartFrog file must be defined like this:

```
variables extends {
	path "/testing/something";
	other "something else";
}
```

You can change what the variables are called but it must be defined inside the variables content.

Programmers note:

This tool uses easy String substitution and therefore infinite loops can happen if a variable uses an other variable name inside its content and the other do the same. E.g.:

path "set to true";
set "some path is here";

path would look like "some some some some some……………..." and just generate this forever.

The tool does not use a parser to check if the syntax is legal.

3.5 UNDOINTERFACE

Short description:

Interface to describe what class must implement to be able to use a standard undo implementation

Depending libraries:

- None

Package name:

ch.cern.openlab.smartfrog.undo

- Undo (Interface)

Input variables:

- *none*

To implement the interface the class must extend the Undo interface.

The only methods the class must implement is the standard run()-method used by the Runnable interface and the Undo()-method called to undo the action done in run().

On error call the method onError(String errorMsg) with an error message of what happened to cause the error. This method is implemented in the interface Undo and will stop the run()-method.

If you have an error that the program in running twice change your run()-method to not start the thread, but just call the super.sfStart(). The reason for this error is that the Undo-interface implements run() and there it starts a thread that executes the run()-method. If you control your own thread and not uses the standard implementation of the Undo-interface the ManageSequence will have no way of controlling the thread.

Programmers note:

This interface does not use standard throw-catch on errors. The reason for this is that the interface should be very easy to implement on already implemented tools. Since the run()-method does what would be the Do()-method it can not throw any exception since that would violate the Runnable interface.

Therefore the onError(..)-method is used to throw an exception. The problem with this way of throwing an exception is that it tries to kill the thread created by run(). But if run() doesn’t find out that it should be terminated it would continue to until it is done.

The time it takes to do the termination of the thread that run() makes can also result in that the program is not finished running before the next one starts. With programs with much print out to screen you can see that some programs might continue even after the call on onError(..)-method. The reason for this is that the program does not handle error in the run()-method. The implementation of run() should stop in a safe mode if an error is catched.

3.6 MANAGESEQUENCE

Short description:

Work as the Sequence class in SmartFrog, but uses the Undo interface to undo the actions done on an error in the sequence.

Depending libraries:

- CERN`s Undo interface for SmartFrog tools (ch.cern.openlab.smartfrog.undo.Undo)

Package name:

ch.cern.openlab.smartfrog.undo

- Undo (Interface)
- ManageSequence
- UndoManager

Input variables:

• *undoAll*	if true an error undo all previous program that is in this sequence.
• *startState*	skip all program before this number. Eg. startState 2 will not execute the first program but jump to the second.
• *justUndo*	true will just run the Undo()-method in all the programs.

None of the variables must be set in the script file used by SmartFrog to run this tool.

If none are set it will start from the beginning of the sequence and only undo the program where an error occurred.

The program sequence in the SmartFrog file must be defined like this:

```
actions extends LAZY {
        test extends Prim {
          sfClass “ch.cern.smartfrog.download.DownloadImp”l;
          protocolConfig “http”;
         ............................
        }
        ...............
}
```

Programmers note:

Notice what programmers note on the Undo Interface says about error detection. If the a program in the sequence never finishes it is because it never terminates. This should be done in the end of the run()-method. There is also a possibility to use the method terminate() in the Undo-Interface.

4. EXAMPLES

4.1 INTRODUCTION

These examples show how a SmartFrog scripting file can use the different components described in this document.
The scenarios used by the example are described in the different examples.

All this examples can be executed by SmartFrog by running “sfRun <name of script file>”. This would only work on a system where SmartFrog is installed and the environment variables used by SmartFrog are set. For more information on how to set environment variables see the SmartFrog documentation. (www.smartfrog.org).

4.2 DOWNLOADER EXAMPLE

Scenario 1:
Check if the files EX_CERN.rpm and web_download.rpm could be downloaded from the server www.cern.ch in the directory examples/downloader. Connect to the server using HTTP.

download_sen1.sf:

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/download/download.sf"

test extends Downloader {
        protocolConfig “none”;
        protocolDownload “http”;
        host “www.cern.ch/examples/downloader”;
        tmpDir “/tmp/download”;
        files [“EX_CERN.rpm”,”web_download.rpm”];
        test true;
}
```

Notes:

Even if the directory where files should be downloaded to is specified it is not created or changed when the test variable is true.

Scenario 2:
Download the files listed in the file download_list.txt. This file is located on the server test.cern.ch in the directory configfiles/ and the files that should be download is located on the server www.cern.ch in the directory testRPMS/download. Both the download_list.txt and the files downloaded should be accessed using HTTP.
The files downloaded should be placed in the directory /tmp/test/

download_sen2.sf

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/download/download.sf"

test extends Downloader {
        protocolConfig “http”;
        protocolDownload “http”;
        config “test.cern.ch/configfiles/download_list.txt”;
        host “www.cern.ch/testRPMS/download/”;
        tmpDir “/tmp/test”;
}
```

The download_list.txt looks like this:

```
WP/config_httpd.rpm
CON/compt.rpm
vpn.rpm
…..
```

Notes:

The directory the files in the download_list.txt is located is created in the local directory the files are downloaded to. So in this example the directory WP and CON are created and the files in this directory are downloaded in these folders.

4.3 RPMINSTALLER EXAMPLE

Scenario 1:
Install all the files in the folder /tmp/download/test1/. Use the rpm command located at /local/sbin/rpm with the options "-–ignoreos --noorder". On error use the option "—notriggers" with undo. To lose the trouble of dealing with dependencies use the force option on the rpm command.

rpm_sen1.sf:

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/rpmInstall/rpmInstall.sf"

test extends Install {
        rpmLocation "/tmp/download/test1/";
        rpmBin "/local/sbin/rpm";
        rpmOptions "--ignoreos --noorder ";
        rpmUndoOptions "—notriggers";
        force true;
}
```

Notes:
The reason all this options can and should be set by the users is that the rpm command has very many different options and how these options are set is different on different version and distributions of rpm.
The RPMInstaller will give the user the output of trying to run the rpm command.
The problem can often be solved by looking at the manual for the rpm command.

4.4 VARINIT EXAMPLE

Scenario 1:
Read a local file in /tmp/test/input.txt and change path with /tmp/download/ and uploadFolder with /tmp/upload/. The result should be saved in the file /tmp/output.txt

varInit_sen1.sf:

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/varInit/varInit.sf"

test extends VarInit {
        protocol "file";
        infile "/tmp/test/input.txt";
        outfile "/tmp/output.txt";
        variables extends {
          path "/tmp/download/";
          uploadFolder “/tmp/upload/”;
        }
};
```

Scenario 2:
Read a file over HTTP located at www.cern.ch/examples/varinit/test.txt and change userDir with /home/test/ and rpmBin with /sbin/rpm. The result should be saved in the file /home/test/output.txt

varInit_sen2.sf:

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/varInit/varInit.sf"

test extends VarInit {
        protocol "http";
        infile "www.cern.ch/examples/varinit/test.txt";
        outfile "/home/test/output.txt";
        variables extends {
          userDir "/home/test/";
          rpmBin “/sbin/rpm”;
        }
};
```

Scenario 3:
Read from SmartFrog script file and change user with braatha and tmpDir with /tmp/. The result should be saved in the file /home/out/output.txt

varInit_sen3.sf:

```
#include "org/smartfrog/components.sf"
#include "ch/cern/openlab/smartfrog/varInit/varInit.sf"

test extends VarInit {
        protocol "http";
        infile ##
          creating user
         with tmp set to tmpDir
        #;
        outfile "/home/out/output.txt";
        variables extends {
          user "braatha";
          tmpDir “/tmp/”;
        }
};
```

4.5 MANAGESEQUENCE EXAMPLE

All the components used by ManageSequence must implement the interface Undo.

Scenario 1:
First check if all the files are on the server, if one is not there stop. If all are there download them. The download should be done through HTTP. After download is done install all the files using RPMInstaller.
The different configurations are taken from some of the previous examples.

manseq_sen1.sf:

```
#include "org/smartfrog/components.sf"
#include "org/smartfrog/sfcore/workflow/components.sf"
#include "ch/cern/openlab/smartfrog/download/download.sf"
#include "ch/cern/openlab/smartfrog/rpmInstall/rpmInstall.sf"

sfConfig extends Sequence {
    sfClass "ch.cern.openlab.smartfrog.undo.UndoSequence";
    undoAll true;
    actions extends LAZY {
      test extends Downloader {
           protocolConfig “http”;
           protocolDownload “http”;
           config “test.cern.ch/configfiles/download_list.txt”;
           host “www.cern.ch/testRPMS/download/”;
           tmpDir “/tmp/test”;
           test true;
        }
      download extends Downloader {
           protocolConfig “http”;
           protocolDownload “http”;
           config “test.cern.ch/configfiles/download_list.txt”;
           host “www.cern.ch/testRPMS/download/”;
           tmpDir “/tmp/test”;
        }
       test extends Install {
           rpmLocation "/tmp/test/";
           rpmBin "”;
           rpmOptions "-- ignoreos --noorder ";
           rpmUndoOptions “—notriggers”;
          force true;
       }
     }
};
```

Notes:

Some of the options used to rpm might not work on your computer; the reason for this is the options this example uses might not be supported by your version of rpm.

Scenario 2:
You want to do the same thing as scenario 1, but now you want to start to download right away and not test, and on an error just undo the component that failed.

manseq_sen1.sf:

```
#include "org/smartfrog/components.sf"
#include "org/smartfrog/sfcore/workflow/components.sf"
#include "ch/cern/openlab/smartfrog/download/download.sf"
#include "ch/cern/openlab/smartfrog/rpmInstall/rpmInstall.sf"

sfConfig extends Sequence {
     sfClass "ch.cern.openlab.smartfrog.undo.UndoSequence";
     startState 2;
     actions extends LAZY {
        test extends Downloader {
              protocolConfig “http”;
              protocolDownload “http”;
              config “test.cern.ch/configfiles/download_list.txt”;
              host “www.cern.ch/testRPMS/download/”;
              tmpDir “/tmp/test”;
              test true;
          }
        download extends Downloader {
              protocolConfig “http”;
              protocolDownload “http”;
              config “test.cern.ch/configfiles/download_list.txt”;
              host “www.cern.ch/testRPMS/download/”;
              tmpDir “/tmp/test”;
          }
         test extends Install {
              rpmLocation "/tmp/test/";
              rpmBin "”;
              rpmOptions "-- ignoreos --noorder ";
              rpmUndoOptions “—notriggers”;
             force true;
         }
        }
};
```

Notes:

To change the start state makes it much easier to debug a sequence of components, since you can start from any state and just undo the component that failed.