SmartFrog Installer Component

Introduction

SFInstaller component is designed to help the user in installing SmartFrog itself on the node before using SmartFrog for any other application delpoyments. This component is similar to any other component written in SmartFrog. It is 100% java component so can run on all java supported platfroms.

The components that are used in SFInstaller are:
1. SmartFrog FTP/SCP component : For transferring the SmartFrog release to remote nodes
2. SmartFrog Telnet/SSH component : For installing the SmartFrog release and starting the SmartFrog Daemon on remote nodes
3. SmartFrog Emailer component: For sending notification to the driver after the installation and starting the daemon service
4. Velocity Generator tool : For generating the sfinstaller description file for multiple hosts with various configurations (without security, with security, with dynamic class loading).

Only one node (driver node) should have SmartFrog installed for where this component will be deployed and hence, will trigger SmartFrog installation on other nodes.

Pre-requisites

1. SmartFrog framework is already installed in root or driver node.
2. SmartFrog daemon is running in driver node.
3. All other nodes should be running Telnet/SSH and FTP/SCP daemons.
4. Other nodes should be accessible from the root [driver] node.
5. Java is installed in remote nodes.

Before moving on to usage of the SFInstaller , let us go through the components that are used in this.

FTP Component

SmartFrog FTP component is used for transferring files to/from one node to other. This can be used as a independent utility component or part of standard SmartFrog workflows like Compound, Parallel, Sequence etc. It uses the Apache Commons net 1.2.0 library which provides APIs for getting remote session and transferring files. The FTP component is build using this library and placed in the classpath.

The following table shows the attributes that need to be configured for using FTP component.

Attributes

Attribute Name	*Description*	*Optional/Mandatory*
ftpHost	FTP host name	Mandatory

username	FTP User Name	Mandatory
transferType	Type of file transfer (possible values "get"/"put")	Optional (Default is "get")
transferMode	Mode of file transfer	Optional (Default is "ascii")
localFiles	List of local files in the form of a vector. Fully qualified, comma separated file names.	Mandatory
remoteFiles	List of remote files corresponding to local files in the form of a vector.	Mandatory
passwordProvider	Reference to password provider component. By default FilePasswordProvider is used.	Mandatory
shouldTerminate	Boolean attribute enables self termination of the FTP component after it upload/download files.	Optional (Default is true)

Please refer to ftpExample.sf for a sample usage of the FTP component under net component.

Telnet Component

SmartFrog Telnet Component is used for executing shell/DOS commands on the remote machine. This can be used as a independent utility component or part of standard SmartFrog workflows like Compound, Parallel, Sequence etc. It uses the Apache Commons net 1.2.0 library which provides APIs for getting remote session and executing commands. The Telnet component is build using this library and placed in the classpath.

The following table shows the attributes that need to be configured for using Telnet component.

Attributes

Attribute Name	*Description*	*Optional/Mandatory*
host	host name of the remote machine	Mandatory
username	Telnet User Name	Mandatory
ostype	OS on remote host (possible values "windows"/"linux")	Mandatory
commands	List of commands to be executed in telnet session in the form of a vector.	Mandatory

passwordProvider	Reference to password provider component. By default FilePasswordProvider is used but user can provide their own Password providers such as Database and DirectoryServer password providers by implementing PasswordProvider interface method getPassword().	Mandatory
timeout	Time in milli seconds. Beyond this time telnet component stops waiting on telnet input stream for some response.	Optional (default value is 30000 ms)
port	Telnet Port	Optional (default value is 23)
shellPrompt	Shell Prompt expected after login in telnet session	Optional (default value is "$")
logFile	Log file used to log all activities in a telnet session. This could be useful for debugging.	Optional
cmdsFailureMsgs	List of commands failure messages for corresponding commands in the form of a vector.	Optional
shouldTerminate	Boolean attribute enables self termination of the Telnet component after executing telnet commands.	Optional (Default is true)

Please refer to telnetExample.sf for a sample usage of Telnet component under net component.

Security

Remember that neither telnet or FTP are secure protocols; both send passwords in clear text when a session is made, and neither perform any rigorous verification that the remote server is the one that is normally used.

The FilePasswordProvider uses passwords stored in a text file. This text file should have restricted permissions: only the user that the SmartFrog daemon runs as needs read access. Because the passwords are kept out of the .sf files, these file do not need to be kept secret, and can be stored in public locations.

SCP Component

SmartFrog SCP component is used for transferring files to/from one node to other over SSH. This can be used as a independent utility component or part of standard SmartFrog workflows like Compound, Parallel, Sequence etc. It uses the Java Secure Channel (Jsch)library. JSch ia a pure Java Implementation of SSH2. It provides APIs for getting remote session and transferring files to/from securely. The component provides support

for both username/password and public/private-key authentication mechanisms. The SCP component is build using this library and placed in the classpath.

The following table shows the attributes that need to be configured for using SCP component.

Attributes

Attribute Name	*Description*	*Optional/Mandatory*
host	Machine running ssh daemon	Mandatory
userId	User login Id	Mandatory
passwordFile	File containing password	Mandatory
keyfile	File containing the RSA key for authentication when using public-private key authentication	Mandatory
localFiles	Comma separated path to local files, in the form of a vector	Mandatory
remoteFiles	Comma separated path to remote files, in the form of a vector	Mandatory
transferType	Type of file transfer (Upload or download)	Mandatory (Possible values are "get" and "put")
port	Port where the ssh daemon is running	Optional (default value is 22)
shouldTerminate	Should terminate this component after processsing or not	Optional (Default is true)

Please refer to scpAuthPassExample.sf and scpAuthPubKeyExample.sf for sample usage of SCP component under ssh component.

SSH component

SmartFrog SSH component is used to remotely execute shell/DOS commands securely over SSH. This can be used as a independent utility component or part of standard SmartFrog workflows like Compound, Parallel, Sequence etc. It uses the Java Secure Channel (Jsch)library. JSch ia a pure Java Implementation of SSH2. It provides APIs for getting remote session and executing a series of commands securely. The component provides support for both username/password and public/private-key authentication mechanisms. The SSH component is build using this library and placed in the classpath.

The following table shows the attributes that need to be configured for using SSH component.

Attributes

Attribute Name	*Description*	*Optional/Mandatory*
host	Host running ssh daemon	Mandatory

usetId	Login ID	Mandatory
commands	Comma separated list of command to be executed in SSH session.	Mandatory
passwordFile	File containing password of user.	Mandatory
keyfile	File containing the RSA key for authentication when using public-private key authentication	Mandatory
timeout	Timout for commands execution	Optional (default value is 0)
port	Port	Optional (default value is 22)
failOnError	Terminate if any command fails.	Optional (Default is true)
shouldTerminate	Boolean attribute enables self termination of the SSHExec component after executing SSHExec commands.	Optional (Default is true)

Please refer to sshExecAuthPassExample.sf and sshExecAuthPubKeyExamplesf for sample usage of SSH component under ssh component.

Emailer Component

SmartFrog Emailer component is used for seding emaile messages. This can be used as a independent utility component or part of standard SmartFrog workflows like Compound, Parallel, Sequence etc. It uses Java Mail 1.3.1 and Java Activation framework version 1.0.2. It provides APIs to send single and multi part email messages over SMTP protocol.The Emailer component is build using these libraries and placed in the classpath.

The following table shows the attributes that need to be configured for using SSH component.

Attributes

Attribute Name	*Description*	*Optional/Mandatory*
to	Comma separated email addresses for to	Mandatory
cc	Comma separated email addresses for cc	Optional
from	Email address from which mail is being send	Mandatory
smtpHost	SMTP Server used to send emails over SMTP protocol	Mandatory
subject	Subject of the email	Optional

message	List of remote files corresponding to local files in the form of a vector.	Optional
runAsWorkFlowComponent	Boolean attribute to enable emailer component run as a standard workflow component. In this mode Emailer is terminated after sending the email. By default it is true.	Mandatory
attachments	Vector of attachments each element denotes one file in file system	Optional
sendOnStartup	Flag to indicate that a message should be sent on startup. Implicitly true when runAsWorkflowComponent is set	Optional
sendOnShutdown	Flag to indicate a message should be sent on shutdown	Optional

Please go through example.sf and exampleUsageAsWFComp.sf for sample usage of Emailer component under emailer component.

Working of SFInstaller Component

SFInstaller component uses these components in a workflow fashion to achieve the goal of installing SmartFrog framework on multiple nodes parallely. For a single node, a Sequence workflow of ftp/scp, telnet/ssh and emailer components is used. To extend this functionality for large number of nodes, a Parallel component having these Sequence components as children is created. This requires to write a complex SmartFrog description when using this for large number of nodes. We use the velocity generator to ease the writing of such a description file. The velocity generator uses a template for sfinstaller component which has complex workflow consisting of various Sequence components in Parallel. The user can use this existing template just by providing configuration attributes for ftp/scp/telnet/ssh/emailer components which are explained earlier. There are two ways in which these atributes can be provided.

Using SFInstaller component

There are two steps in using SFInstaller component.

1. Generating the sfinstaller description file for multiple hosts

The description file for installing SmartFrog on multiple hosts is generated using the velocity template written for this component. There are two ways in which the velocity generator can be used.

Using the API

A Java API is available for this. The API signature is:

static public void createTemplate(Map map, String templateFile, String outputFile, boolean securityStatus, boolean dynamicLoadingStatus, String [] jars)

a) Place the velocity jar and sfinstaller.jar files from the sfinstaller dist\lib directory in the classpath.

b) Create a map of Daemon objects.

The Daemon object is used to store information about the each node. The Daemon object is created as:

public Daemon(String name, String os, String host, String transfertype, String logintype, String user, String password, String localfile1, String localfile2, String localfile3, String keyfile, String secproperties, String smartfrogjar, String servicesjar, String examplesjar, String releasename, String javahome, String installdir, String emailto, String emailfrom, String emailserver)

where:

name: logical name for the daemon

os: OS (Windows or Linux) for the daemon

host: hostname or IP address of the node

transfertype: Transfertype (ftp/scp)

logintype: LoginType(telnet/ssh)

user: username for logging in

password: password file

localfile1: SmartFrog release file

localfile2: JavaService executable (for Windows platform)

localfile3: Javaservice wrapper for starting SmartFrog daemon (for Windows platform)

keyfile: Security key file (only for starting Daemon in secure mode)

secproperties: Security properties file (only for starting Daemon in secure mode)

smartfrogjar: Signed smartfrog jar file (only for starting Daemon in secure mode)

servicesjar: Signed smartfrog services jar file (only for starting Daemon in secure mode)

examplesjar: Signed smartfrog examples jar file (only for starting Daemon in secure mode)

releasename: Release name

javahome: Java home on remote node

installdir: installation directory on remote node

emailto: Comma separated email addresses for to

emailfrom: email address from which mail is being send

emailserver: SMTP Server used to send emails over SMTP protocol

Example of creating Daemon object:

Daemon host1 = new Daemon("daemon1", "linux", "host1.india.hp.com", "ftp", "telnet", "root", "D:\\\\passwd.txt", "D:\\\\SmartFrog.3.04.014.beta.20050103_ALL.tar.gz", null, null, null, null, null, null, null, "SmartFrog.3.04.014.beta", /usr/local/java, "/tmp", "abc@hp.com", "smartfrog@hp.com", "smtphost.hp.com")

c) Use the java service executable and java service wrapper from the bin directory.

d) Use the "sfInstaller.vm" template file.

e) Call the API : TemplateGen.createTemplate(map, templateFile, outputFile, securityOn, dynamicLoadingOn, jars)

where:

map: Map with Daemon objects for the description

templateFile: Velocity template file

outputFile: Output description file

securityStatus: A flag to keep security on of off

dynamicLoadingStatus: A flag to keep dynamic classloading on or off

jars: List of jar files for dynamic classloading

Please refer to Example.java for a sample usage of API.

Using the Script

The doAll.bat and doAll scripts are available for this.

a) Place the velocity jar and sfinstaller.jar files from the sfinstaller dist\lib directory in the classpath.

b) Modify the hosts.all file with the following configuration data for each host:

name: logical name for the daemon

os: OS (Windows or Linux) for the daemon

host: hostname or IP address of the node

transfertype: Transfertype (ftp/scp)

logintype: LoginType(telnet/ssh)

user: username for logging in

password: password file

localfile1: SmartFrog release file

localfile2: JavaService executable (for Windows platform)

localfile3: Javaservice wrapper for starting SmartFrog daemon (for Windows platform)

keyfile: Security key file (only for starting Daemon in secure mode)

secproperties: Security properties file (only for starting Daemon in secure mode)

smartfrogjar: Signed smartfrog jar file (only for starting Daemon in secure mode)

servicesjar: Signed smartfrog services jar file (only for starting Daemon in secure mode)

examplesjar: Signed smartfrog examples jar file (only for starting Daemon in secure mode)

releasename: Release name

javahome: Java home on remote node

installdir: installation directory on remote node

emailto: Comma separated email addresses for to

emailfrom: email address from which mail is being send

emailserver: SMTP Server used to send emails over SMTP protocol

c) Modify the template file "sfinstaller.vm" under sfinstaller/src/org/smartfrog/services/installer for any configuration changes like jar files for classloading etc.

d) Run the scipt/batch file doAll/doAll.bat under sfinstaller/src/org/smartfrog/services/installer.

2. Deploying the generated description file

a) Place the jar files for the dependent SmartFrog components like ftp/telnet/scp/ssh/emailer in the classpath of the smartfrog daemon, running in the driver node. These jar files are packaged with specific components like Apache Commons net libraries and sf-net.jar are available in net component, JSch libraries and sf-ssh.jar are available in ssh component, Java Mail 1.3.1 and Java Activation framework version 1.0.2 libraries and sf-emailer.jar are available in emailer component.

b) The generated description file is deployed using sfStart on the driver node.