SSH Components

2 Feb 2011

SSH Components

2 Feb 2011

1 Introduction

The SSH component package, `sf-ssh`, provides SmartFrog components to connect to remote hosts, either to issue remote commands, or to copy files to and from the local host. This enables the components to be used to install SmartFrog itself on remote systems, or to manage a remote host that does not have SmartFrog itself installed. This can be useful when managing embedded systems, such as routers.

These components have been updated for SmartFrog 3.12.008; the changes are not yet complete. The recent and planned changes are both listed in this document.

2 Conceptual Model

SSH components are all derived from the `SSHComponent`. This is a workflow-enabled component: it can be configured to terminate after deployment, and so used in a workflow sequence.

The base component template defines common attributes for all components: the target host and port, a list of known hosts to trust, whether to trust all certificates,

```
SSHComponentSchema extends Schema {
    //authentication policy, one of "password" or "key"
    authentication extends String;

    //host to connect to
    host extends String;

    //should failures be reported
    failOnError extends OptionalBoolean;

    //for key authentication, the file containing the private key
    keyFile extends OptionalFilenameType;

    //hosts to trust
    knownHostsFile extends OptionalFilenameType;

    //a reference to a password provider
    passwordProvider extends Compulsory;

    //port to connect to
    port extends Integer;

    //timeout in milliseconds
    timeout extends Integer;

    //should all certificates be trusted
    trustAllCertificates extends Boolean;

    //the user name
    username extends String;
}


SSHComponent extends WorkflowPrim {
    sshSchema extends SSHComponentSchema ;
    authentication authenticate_password;
    failOnError true;
    port 22;
    sfShouldTerminate true;
    timeout 600000;
    trustAllCertificates true;
    //authentication policies
    authenticate_password "password";
    authenticate_key "key";
}
```

The default options of the `SSHComponent` trigger connections to port 22, the standard port for SSH, with trust for all remote certificates. The connection times out after ten minutes.

The authentication policy is either password or public key; public/private key connections are considered to be much more secure, though that does require a secure distribution and storage of the private keys.

Both policies also need a `passwordProvider`. This must be a LAZY reference to a component that provides passwords.

```
#include "/org/smartfrog/services/passwords/components.sf"

PasswordProvider extends InlinePassword {
    password "not-very-secret";
}
```

3 SSHExec Component

The `SSHExec` component can issue a series of commands to a remote server; the commands are represented as a list

```
SSHExec extends SSHComponent {
   sfClass "org.smartfrog.services.ssh.SSHExecImpl";
   sshSchema extends SSHComponentSchema  {
       commands extends OptionalVector;
       logFile extends OptionalFilenameType;
       //min-max values for the response
       exitCodeMin extends Integer;
       exitCodeMax extends Integer;
   }
   authentication SSHComponent:authenticate_password;
   exitCodeMax 0;
   exitCodeMin 0;
   //the component terminates at the end of deployment, by default
   sfShouldTerminate true;
}

SSHExecWithPrivateKey extends SSHExec {
   authentication SSHComponent:authenticate_key;
}
```

The commands are sent as a sequence down a single opened shell, so operations such as `cd ..` or `export DISPLAY=:0` will set the context for the following operations.

To close the connection, end the list of commands with the string "exit"; this will close the shell.

The `exitCodeMin` and `exitCodeMax` commands check the return code from the shell; they specify the range within which the exit code must fall.

Changes to SSHExec component in SmartFrog 3.12.008

- Choosing between authentication policies is a matter of switching authentication attribute, and providing the appropriate other attributes that specific authentication schemes may require
- the `passwordFile` attribute can be a reference to a `FileComponent`, as well as a path to a file.
- The attribute `sfShouldTerminate` is used to control whether the component should terminate or not.
- All validation of configuration values takes place before the connection is made.

Changes to SSHExec component in SmartFrog 3.12.018

- Commands are executed asynchronously.

Changes to SSHExec component in SmartFrog 3.12.026

- The `knownHostsFile` attribute takes the name of a file (or a LAZY reference to a File component) containing a list of trusted hosts. The format of the file is that of the command line ssh tools.

Planned Changes

Here are some items on the issue list for these components.

1. A planned change is to add an interface to allow other components to queue new commands.

4 ScpOperation

An `ScpOperation` is a component that can get or put multiple files in a single connection, over an SSH connection.

```
ScpSchema extends SSHComponentSchema  {
    localFiles extends Vector;
    remoteFiles extends Vector;
    transferType extends String;
}

/**
 * This is the basic Scp Operation
 */
ScpOperation extends SSHComponent {
    sfClass "org.smartfrog.services.ssh.ScpComponentImpl";
    sshSchema extends ScpSchema;

    //get files from the remote server
    get_files "GET";
    //put files to the remote server
    put_files "PUT";

    //default transfer is a get
    transferType get_files;
}

/**
 * Switch to the public/private keypair
 */
ScpOperationWithPrivateKey extends ScpOperation {
    authentication SSHComponent:authenticate_key;
}
```

The list of local files and that of the remote files must match. For bulk uploads where the number and name of files is unknown, the ScpBulkUpload component is often preferable.

Changes to SCP component in SmartFrog 3.12.008

- Choosing between authentication policies is a matter of switching authentication attribute, and providing the appropriate other attributes that specific authentication schemes may require
- the `passwordFile` attribute can be a reference to a `FileComponent`, as well as a path to a file.
- The attribute `sfShouldTerminate` is used to control whether the component should terminate or not.
- All validation of scp operation parameters takes place before the connection is made
- Copies are made asynchronously. This helps for long-haul deployment, but means that you cannot rely on the copy to have completed before the next component in a simple compound is started. You must use workflow containers or otherwise register an interest in the event raised when the scp component is terminated.
- Liveness tests of the component now relay the state of the worker thread. If it fails, the liveness fails.

5 SCPBulkUpload

This component does a bulk upload of a set of files, using the `java.util.regex` expression syntax to specify a set of files to upload. All files are placed in the single remote directory.

```
ScpBulkUpload extends SSHComponent {
    sfClass "org.smartfrog.services.ssh.ScpBulkUploadImpl";
    scpSchema  extends FilesSchema  {
      remoteDir extends String;
      //this value gets added as we go along
      transferCount extends OptionalInteger;
    }

  pattern Files:pattern;
  includeHiddenFiles Files:includeHiddenFiles;
  caseSensitive Files:caseSensitive;
  fileCount -1;
  minFileCount -1;
  maxfileCount -1;
}
```

As an example, here is a bulk upload of RPM files, to a remote directory `/home/example/rpm`.

```
  upload extends ScpBulkUpload {

    remoteDir "/home/example/rpm";
    dir LAZY PARENT:PARENT:action:tempdir;
    passwordProvider LAZY PARENT:PARENT:action:passwordProvider;
    pattern "\\w+.rpm";
    host "server1";
    username "test";
    sfShouldTerminate true;

  }
```

The component will terminate when the files are uploaded.

Some attributes read/write the number of files to work on. These are for validating and testing operations.

`fileCount`: This specifies the expected number of files to upload. When absent or less than zero, it is ignored. when zero or greater, it declares that the number of files matching the upload pattern must be exactly the number specified in the `fileCount` attribute. This is useful for verifying that regular expressions are working.

`minFileCount`: the minimum number of files expected to match the regular expression. Setting this to 1 declares that at least one file must be uploaded.

`maxFileCount`: the maximum number of files expected to match the regular expression. This can be a sanity check on accidentally uploading large numbers of files.

`transferCount`: this keeps an ongoing record of the number of files transferred. It is only upset after successful transfers.