SmartFrog Ant Components

2 Feb 2011

SmartFrog Ant Components

2 Feb 2011

1 Introduction

The SmartFrog Ant Components allow Ant tasks to be run as part of a SmartFrog deployment. There are two components

- `Ant`: Provides access to any Ant task from within SmartFrog.
  This offers direct access to Ant tasks, and is a convenient way to access the many built in features of Ant from a deployment.
- `AntBuild`: Runs one or more existing Ant build files.
  This offers a way to run existing build files during a deployment. Many Java projects provide `build.xml` files to start them — this component can run these build.xml files and put them under control of SmartFrog, integrating logging and termination

Language aside,Ant and SmartFrog appear very similar: they are both languages and runtimes for getting work done under Java. However, Ant has always focussed strongly on *build time workflow*: there's little scope for failure handling and recovery outside testing; there is no easy way to halt a task midway through its work. We have tried to add these services on top, using techniques pioneered in Eclipse and NetBeans, but it is still somewhat limited.

2 Common Features

Here is the template in `/org/smartfrog/services/ant/components.sf` that defines a base component for the Ant components.

```
AntWorkflowComponent extends WorkflowPrim {

  antComponentSchema extends Schema {
    //list of [name,value] pairs defining ant properties
    properties extends Vector;

    //this is the name of a target to set the properties on
    propertyTarget extends OptionalCD;
    //log level, one of "debug", "info", "error", "warn"
    logLevel extends String;

    //this is an optional base directory. If it exists, then
    //it is used as the base directory for the run.
    //In AntBuild, it also sets the base dir for all (relative) directories
    //in the directory list
    basedir extends OptionalFilenameType;

  }


   LOG_INFO "info";
   LOG_DEBUG "debug";
   LOG_ERROR "error";
   LOG_WARN "warn"
   logLevel LOG_INFO;
}
```

1. All Ant components are workflow components; they can be set to terminate after their run.
2. The log level determines what information is logged to the SmartFrog console. Logging may be enhanced in future, so that we directly map the Ant debug/verbose/info/warning levels to those of SmartFrog.
3. The optional `basedir` attribute specifies a base directory. This is interpreted differently by the two components, but it effectively defines the directory in which builds run.

4. The `propertyTarget` attribute takes a LAZY reference to another component. After the Ant build is finished, all of the properties of that build are added as attributes to the target component. These attributes can then be referenced by other components.
5. The `properties` attribute is a list of name-value pairs, in their own lists:

```
properties [
      ["p1","overwrite"],
      ["new.prop","new.value"]
      ];
```

These are turned into Ant properties and passed down to the ant run(s). As with properties set on Ant's command line, these properties are immutable, and are passed down to any subsidiary builds that are invoked.

3 Ant Component

The Ant component executes all nested component descriptions as Ant tasks; mapping SmartFrog attributes to setters in the Java classes. attributes in the normal SmartFrog language.

This provides a simple way to use Ant tasks in a SmartFrog deployment.

```
Ant extends AntWorkflowComponent {


    antSchema extends Schema {
        //path to task declarations
        tasksResource extends String;

        //path to type declarations
        typesResource extends String;

        //this is just here to say leave the runtime alone.
        //it is for lazy references to ant properties
        runtime extends OptionalCD;

        asynch extends Boolean;
    }

   /**
    * Mandatory attribute: the class for component description
    */
   sfClass "org.smartfrog.services.ant.AntImpl";

   tasksResource "/org/apache/tools/ant/taskdefs/defaults.properties";
   typesResource "/org/apache/tools/ant/types/defaults.properties";
   asynch false;
   properties [];
}

/**
 In a workflow run, the operation is asynchronous, and terminated afterwards
 */
AntWorkflowRun extends Ant {
  sfShouldTerminate true;
  asynch true;
}
```

The component can be set to deploy synchronously during the startup phase of a component, or asynchronously. We recommend the asynchronous operation for any slow actions, as a slow synchronous operation can trigger timeouts in the overall deployment, as the parent container may assume that the Ant component has failed to start up. As the component is workflow-enabled, components may listen for it terminating; it can also be embedded inside a Sequence or other workflow container.

Here is an example workflow which sets properties and echoes some messages.

```
TestProperties extends AntWorkflowRun {
  echoTask extends echo {
    message "hello world!";
```

```
  }


  setProperty extends property {
    name "sfhome";
    value "SFHOME is <${env.SFHOME}>";
  }


  echoSFHOMETask extends echo {
    message "${sfhome}";
   }
}
```

This can be used in a sequence:

```
tests extends Sequence {
  antRun extends TestProperties {
    propertyTarget LAZY PARENT;
  }

  sfHome extends AssertPropertySet {
    propertyTarget LAZY PARENT;
    name "sfhome";
  }


  java.version extends sfHome {
    name "java.version";
  }

  antHome extends sfHome {
    name "ant.home";
  }
```

By setting the `propertyTarget` attribute, we can ensure that all the properties of the run are saved as attributes on a component. This must be a component that is set to outlast the (terminating) Ant run itself, if the values are to be accessed in successor components.

4 AntBuild Component

This component provides a way to run a complete Ant build file, or set of build files across multiple directories. the the properties generated after a build can be saved to a target component, and so outlast the life of the `AntBuild` component itself. This enables the output of an Ant run to be fed into a SmartFrog deployment. This component can do the following things:

1. Run an existing Ant build file using the directory it exists as the base directory.
2. Run an existing Ant build file against a list of directories.
3. Run a named Ant build file under a list of directories.

The first of these actions is equivalent to typing `ant -f build.xml` in a directory; The last two of these are more complex. Running an existing build file against a list of directories means taking an existing file, say "clean.xml" and applying it to every directory. Running a named build file under a list of directories means running a file called "build.xml" in every directory named.

No matter how you run the build, you have the following options

- Provide a list of Ant properties to set before the build(s) begin
- Provide a list of ant targets to run. You can also declare whether or not it is an error for a build file to be lacking any of these targets.
- Nominate a component to collect the results of the builds

- Declare whether or not a failure of the build file should terminate the component.

This component always runs its build asynchronously; you can declare it a workflow component, in which case it will terminate after the build.

Here is the definition of this component:

```
/**
 * This component runs Ant on the build files in the target directories.
 * This build does not run ant.bat, ant.sh or ant.pl; it runs Ant in the nominated
 * SmartFrog process. To avoid memory leaks from Ant tasks (e.g. Javac,Ivy), run
 * the build in a new process.
 * 1. Every build is sequential.
 * 2. a failure halts the run with the BuildException mapped to a SmartFrogException.
 * 3. you can specify zero or more targets to run
 * 4. logging is routed directly to the SmartFrog log.
 * 5. execution is always asynchronous after the component starts.
 * 6. As there is no separate JVM, all options such as memory, xml parser, etc, are inherited
 *    from SmartFrog.
 * 7. if antfile is set, it is a single build file to run in every directory in the
 *     directories list
 *  -or in basedir,if that is set, or in the direcotory of antfile itself, if basedir is
 *    unset.
 */
AntBuild extends AntWorkflowComponent {


    antSchema extends Schema {
        //list of targets to run. If empty, the default target is executed
        targets extends Vector;

        //the name of a generic ant file. Overrides the value of buildfile
        antfile extends OptionalFilenameType;

        //the name of an ant file to be resolved in every directory
        buildfile extends String;

        //a list of directories. Will be resolved relative to basedir, when
        //relative resolution is required.
        directories extends OptionalFilenameList;

        //should we keep running targets in a single build if the first one fails
        keepGoingInSingleBuild extends Boolean;
        //should we keep going if a build file fails. The first exception will be
        //thrown on termination, but the other builds will still be executed
        keepGoingAcrossFiles extends Boolean;

        //how long to wait patiently for a shut down.
        shutdownTimeout extends Integer;

        //should we skip missing targets
        skipUnimplementedTargets extends Boolean;

    }

    sfClass "org.smartfrog.services.ant.AntBuildImpl";

    //defaults to build.xml
    buildfile "build.xml";
    //we are workflow by default
    sfShouldTerminate true;
    keepGoingInSingleBuild false;
    keepGoingAcrossFiles false;
    shutdownTimeout 2000;
    skipUnimplementedTargets false;
    targets [];
    properties [];
}
```

An example of this component in use is a build that runs a build.xml file created inline, as discussed below. First, the build file and results are deployed:

```
  action extends Compound {
    build extends TestBuildFile;
    results extends Compound {
```

```
    }
}
```

These components will persist until the overall deployment is terminated. The `AntBuild` is then executed as a workflow:

```
tests extends Sequence {

  //run the build
  antbuild extends AntBuild {
    propertyTarget LAZY action:results;
    antfile LAZY action:build;
  }

}
```

This sequence runs the temporary build file created in the deployment, and saves the properties to the component `action:results`. This is the base template for the functional tests of the `AntBuild` component

5 Creating Build Files in SmartFrog

The `TempBuildFile` component creates a temporary file in a nominated directory (by default, the Java temp directory). Here is it in use in the test files

```
TestBuildFile extends TempBuildFile {

  text ##
   <project name="set-properties" default="default">

    <target name="default" depends="inner">
      <property name="p1" value="p1" />
      <property name="p2" value="p2" />
      <property name="pwd" location="." />
    </target>

    <target name="inner">
      <property name="inner" value="true" />
    </target>

    <target name="fail">
      <property name="failure.message" value="deliberate failure" />
      <property name="failure.status" value="-1" />
      <fail status="${failure.status}"
        message="${failure.message}"/>
    </target>

    <target name="sleep">
      <property name="sleep.milliseconds" value="100" />
      <sleep seconds="sleep.milliseconds" />
    </target>
   </project>
  #;

}
```

The XML components in the `sf-xml` JAR can be used to create well formed XML files, though it is easier to use the `Ant` component directly.

6 Implementation Details

The implementation details may be interesting to anyone who wishes to use the tasks, or integrate Ant with their own builds.

6.1 Execution

We load and execute Ant build files by creating new `Project` instances. We have to be thorough about forgetting about these projects and any classes they create, otherwise the Java process will leak memory copiously.

6.2 Memory Leakage

Ant tries hard not to leak memory; the IDE developer teams get in contact with the Ant developers whenever such an event occurs. However, some of the programs that Ant runs in-VM can leak memory, and tasks can leak memory if they try hard.

1. If classes loaded in a custom classloader are retained, the classloader itself is not garbage collected. The symptoms of this are an Out of PermGen Space Exception. Do not hang onto Ant-loaded classes past a build.
2. JUnit tests and other programs run in the `<java>` task, can, if `fork="false"`, leak.
3. Tasks that use Thread Local Storage (i.e. `java.lang.ThreadLocal` instances) can leak memory.
4. Classes/Libraries that create and never clean up large static datastructures can leak memory. The javac and javadoc programs have done this in the past.

Because of these risks, we advise starting a complex build in a new Java process. This is done simply by setting the `sfProcessName` attribute of the deployed `Ant` or `AntBuild` component or that of its parent to a new process name.

6.3 Logging

Logging is hooked in fairly simplistically right now; we grab the formatted strings from Ant's main logger and route them to wherever output goes. The next step would be to map Ant levels 1:1 to our log levels, and do all the formatting ourselves. SmartFrog logging is pretty tricky to set up right, but it can be changed on the fly, so during a long deployment you can ask for more/less information in places.

6.4 Exceptions

Exception Handling is processed by catching any Ant `BuildException`, and turning it into a `SmartFrogAntBuildException`, with location and (maybe) exit status pulled out. This can go over the wire to the caller over the RMI/SSL link. As with all chained RMI exceptions, there is still some brittleness related to exception unmarshalling nested exceptions; it is expected that the Ant component is on the classpath of the recipient.

The classname of the exception is defined in the `ant/components.sf` file:

```
AntBuildConstants {
  //a SmartFrog exception that indicates the build has halted
  AntBuildException "org.smartfrog.services.ant.SmartFrogAntBuildException";
  //this is a subclass of Ant's BuildException, one that is raised to interrupt a build.
  BuildInterruptedException "org.smartfrog.services.ant.BuildInterruptedException"
}
```

6.5 Interrupting/Terminating a build

Ant builds are not (currently) designed to be interruptible. Yet all IDEs manage to do this.

We have a special Ant Logger that forwards all output to whichever other logger it is configured to forward all output. Before it does so, it checks a property that tests a flag to see if it has been requested that the build is interrupted. If this flag is set, it throws a new `BuildInterruptedException`, an exception that extends Ant's own `BuildException`.

This is apparently how NetBeans and Eclipse halt builds; this is clearly a common enough use case that Ant itself should support it.

The key point to remember is:

> *SmartFrog can only cleanly interrupt a build when the task or program prints something; long-lasting programs that do not print, do not get interrupted.*

If a build is not interrupted within the time limit specified by `shutdownTimeout`, the thread running the build will itself be terminated. This is messy and can leak memory, possibly even spawned processes.

> *It is better to spawn Java programs from SmartFrog itself, rather than delegate to Ant*

6.6