Java Properties in CAPS 5.1
Using Properties for Runtime Control, Note 3

Custom properties in a properties file

Introduction

Java CAPS 5.1 developer may need to determine a runtime value of one or more system properties to, perhaps, use its vale for runtime flow control. This is the next in a series of notes on the use of Java Properties for controlling runtime behaviour of Java CAPS solutions.

This note introduces the use of the Java Properties file, some locations where such a file can be put, a method to use a properties file residing in an arbitrary location without hardcoding the location, and the most basic method of access to the properties in the properties file.

Examples shown in this note apply to the Sun SeeBeyond Integration Server, provided as part of the Java CAPS 5.1 distribution.

Java Properties file

A Java Properties file may contain a series of *key=element* pairs, where *key* is the name of the property and *element* is the value of the property. See description of the java.util.Properties class **load** method, http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html, for a detailed discussion of what the properties file can contain and how it can be formatted.

Let's assume with have a properties file containing the following:

```
# begin properties
My.property.1     :theFirstProperty
MyProperty2       =theSecondProperty
! end properties
```

in a properties file named note3.properties.

Loading Java Properties from a file

To get the properties into the JCD we need to use the java.util.Properties *load* method. The load method expects a java.io.InputStream argument. This stream would, ultimately, be associated with a properties file somewhere in the file system.

One could construct the InputStream through a FileInputStream using a reference to a file in the file system. This reference would be an absolute reference or a reference relative to the default application working directory.

Here are sample Java statements that would get the Properties object populated from a properties file whose absolute path is hardcoded.

```
java.util.Properties p = new java.util.Properties();
;
java.io.File fin
       = new java.io.File("c:/temp/properties/note3-1.properties" );
java.io.FileInputStream fins = new java.io.FileInputStream( fin );
p.load( fins );
fins.close();
;
```

If the properties file is not present an exception will be thrown.

Hardcoding absolute file references in JCDs may be considered non-portable.

This method contravenes the dictates of the EJB specification as the JCD is explicitly accessing a file in the file system. Since the file is accessed for reading and there is no lock this would be considered a minor matter.

This method will load the properties file each time the JCD is invoked.

To avoid hardcoding the absolute path to the property file one could consider placing the file in the application's default working directory. For a Java CAPS Java Collaboration Definition the default application working directory will be `<JCAPSInstallRoot>/logicalhost/is/domains/<domain-name>/config`. Assuming we have a properties file called note3-1.properties in the application's default working directory we could do something like this:

```
java.util.Properties p = new java.util.Properties();
;
java.io.File fin = new java.io.File( "note3-2.properties" );
java.io.FileInputStream fins = new java.io.FileInputStream( fin );
p.load( fins );
fins.close();
;
W_toFileClient.setText( propertiesListAsString( p ) );
W_toFileClient.write();
;
```

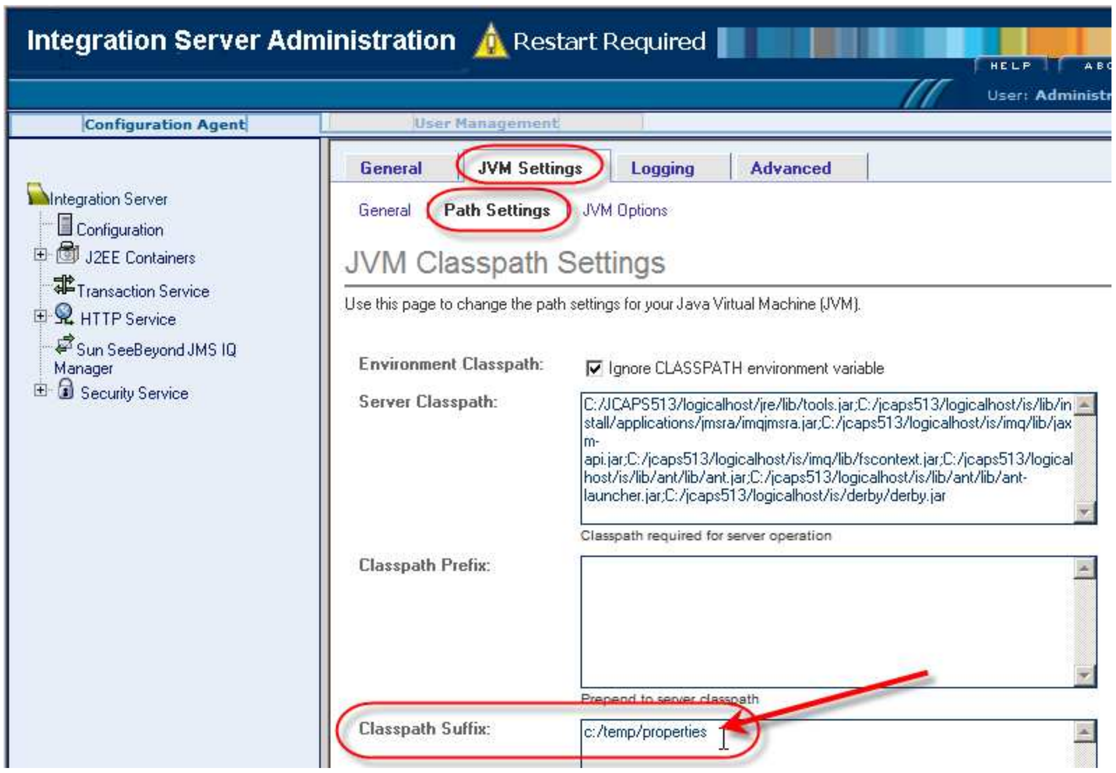
If the properties file is not present an exception will be thrown.

Dumping property files in server's *config* directory might be considered somewhat questionable.

This method contravenes the dictates of the EJB specification as the JCD is explicitly accessing a file in the file system. Since the file is accessed for reading and there is no lock this would be considered a minor matter.

This method will load the properties file each time the JCD is invoked.

One could avoid using the FileInpoutStream, whether with an absolute or a relative file path reference, by exploiting the Java Class Loader's capability to locate a resource at runtime through the classpath and provide it as a stream to the collaboration. Unfortunately, Java CAPS classpath consists of a long series of absolute paths to Java Archives but there are no directories. One must explicitly add a directory, which will contain the properties file, to the classpath. The Enterprise Manager of the Integration Server Administration Console can be used to do this. Let's add a directory `c:/temp/properties` to the classpath and place a file `note3-3.properties` there. Note that modification of the classpath requires restart of the Integration Server.



JVM Classpath Settings

Use this page to change the path settings for your Java Virtual Machine (JVM).

Environment Classpath: ☑ Ignore CLASSPATH environment variable

Server Classpath: C:/JCAPS513/logicalhost/jre/lib/tools.jar;C:/jcaps513/logicalhost/is/lib/install/applications/jmsra/imqjmsra.jar;C:/jcaps513/logicalhost/is/imq/lib/jaxm-api.jar;C:/jcaps513/logicalhost/is/imq/lib/fscontext.jar;C:/jcaps513/logicalhost/is/lib/ant/lib/ant.jar;C:/jcaps513/logicalhost/is/lib/ant/lib/ant-launcher.jar;C:/jcaps513/logicalhost/is/derby/derby.jar

Classpath required for server operation

Classpath Prefix:

Prepend to server classpath

Classpath Suffix: c:/temp/properties

The following will load the properties file using the classloader:

```
java.util.Properties p = new java.util.Properties();
```

```
;
p.load
      ( this.getClass().getClassLoader().getResourceAsStream
           ( "note3-3.properties" ) );
```

If the properties file is missing a runtime exception will be thrown.

This method will load the properties file each time the JCD is invoked.

Access properties

Use code similar to that in the project developed in Note 1 to list all properties, or to access specific properties by name, for example:

```
String sAllProperties = propertiesListAsString( p );
;
String MyProperty2 = p.getProperty( " MyProperty2" );
;
```

Characteristics of this method

Properties are local to the Java Collaboration.
Property values can be changed at any time.
Property file is read each time through the collaboration.

Closing

One can add custom Java properties to be used for dynamic runtime control of Java CAPS solutions by reading a Java Properties file from within a Java Collaboration.

The JCD receive method, shown below, consolidates all of the code fragments discussed in this note.

```
public void receive
    ( com.stc.connectors.jms.Message input
    , com.stc.connector.appconn.file.FileApplication W_toFileClient )
        throws Throwable
{
    java.util.Properties p = new java.util.Properties();
    ;
    {
        java.io.File fin
            = new java.io.File
                ( "c:/temp/properties/note3-1.properties" );
        java.io.FileInputStream fins
            = new java.io.FileInputStream( fin );
        p.load( fins );
        fins.close();
        ;
        W_toFileClient.setText( propertiesListAsString( p ) );
        W_toFileClient.write();
    }
    ;
    {
        java.io.File fin = new java.io.File( "note3-2.properties" );
        java.io.FileInputStream fins
             = new java.io.FileInputStream( fin );
        p.load( fins );
        fins.close();
        ;
        W_toFileClient.setText( propertiesListAsString( p ) );
        W_toFileClient.write();
        ;
    }
    ;
    p.load
       ( this.getClass().getClassLoader().getResourceAsStream
           ( "note3-3.properties" ) );
    W_toFileClient.setText( propertiesListAsString( p ) );
    W_toFileClient.write();
    ;
    ;
}
```

The helper method, `propertiesListAsString`, used in the code above, is shown below.

```
String propertiesListAsString( java.util.Properties props )
    throws java.io.IOException
{
    // get system properties list as string
    java.io.ByteArrayOutputStream baos
                           = new java.io.ByteArrayOutputStream();
    java.io.PrintStream ps = new java.io.PrintStream( baos );
    props.list( ps );
    ps.flush();
    baos.flush();
    String sProperties = baos.toString();
    ps.close();
    baos.close();
    return sProperties;
}
```