IBM Application Runtime Expert for i

Replacement Variables – What They Are and How to Use Them

Background

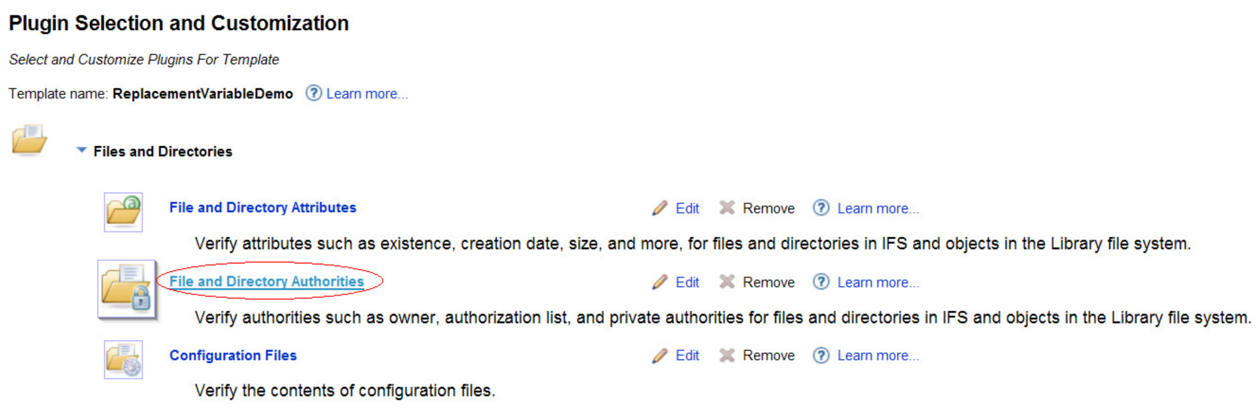
The IBM Application Runtime Expert for i, or ARE, provides robust support for replacement variables. Replacement variables – often call path variables – allow static values, such as directory paths, to become dynamic, meaning that their exact values can be calculated (and substituted) at runtime. This is extremely useful for building templates where, for example, the structure of a directory tree is static, but the root of the directory tree may vary.

The ARE Web user interface provides a way to define replacement variables when dealing with IFS paths in plugins such as File and Directory Attributes and File and Directory Authority. A much more comprehensive set of replacement variable options are also available in all plugins by editing the XML (which can also be done directly from the Web user interface) for the plugin and adding some simple XML tags. An example of this second (editing the XML) feature is covered in a separate document.

IMPORTANT: It is important to understand that a replacement variable is not scoped to a specific plugin. All replacement variables are global in scope, which means the variable name and value are available for use by any plugin.

Replacement Variables in the Deployment Template Editor

Let's walk through an example of using replacement variables during the process of creating a template. In this example, we use the deployment template editor to create a replacement variable for an IFS path. We'll then look at how to specify what value to substitute for that replacement variable when the template is used to verify a system. First, we'll create a new template called 'ReplacementVariableDemo', and select the Edit option for the File and Directory Authorities plugin.


Plugin Selection and Customization

Select and Customize Plugins For Template

Template name: **ReplacementVariableDemo** Learn more...

Files and Directories

File and Directory Attributes Edit Remove Learn more...

Verify attributes such as existence, creation date, size, and more, for files and directories in IFS and objects in the Library file system.

File and Directory Authorities Edit Remove Learn more...

Verify authorities such as owner, authorization list, and private authorities for files and directories in IFS and objects in the Library file system.

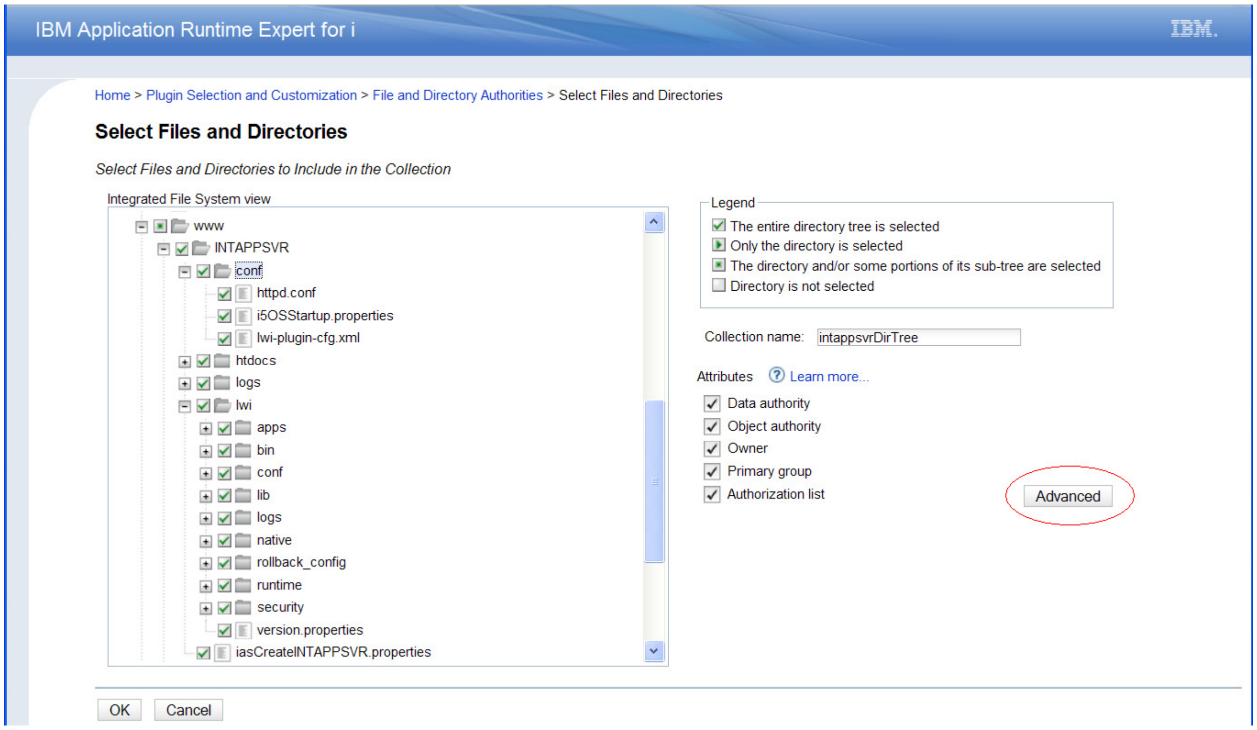
Configuration Files Edit Remove Learn more...

Verify the contents of configuration files.

Figure 1 – Select the file and directory attributes plugin

Next, create a new collection of files and directories by clicking on the ‘Add file/directories’ button.

Select the directory tree you would like to verify. For this example, we selected the directory tree /www/INTAPPSVR, which is the directory tree for an integrated Web application server instance named INTAPPSVR.


IBM Application Runtime Expert for i

Home > Plugin Selection and Customization > File and Directory Authorities > Select Files and Directories

Select Files and Directories

Select Files and Directories to Include in the Collection

Integrated File System view

```
www
  INTAPPSVR
    conf
      httpd.conf
      i5OSStartup.properties
      lwi-plugin-cfg.xml
    htdocs
    logs
    lwi
      apps
      bin
      conf
      lib
      logs
      native
      rollback_config
      runtime
      security
      version.properties
    iasCreateINTAPPSVR.properties
```

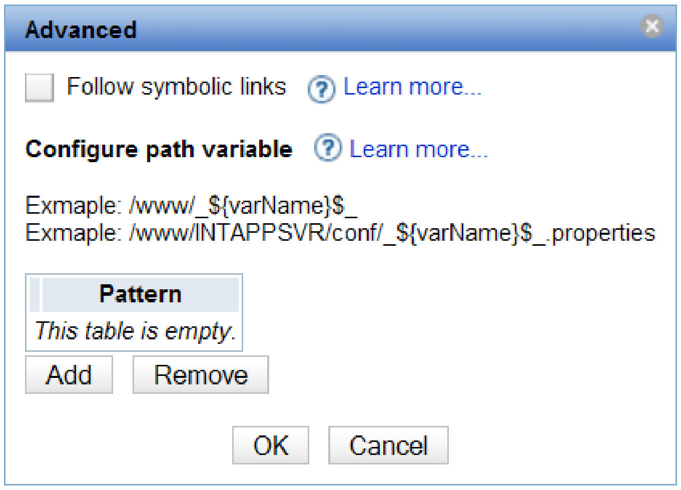
Legend
- ✓ The entire directory tree is selected
- ▶ Only the directory is selected
- ■ The directory and/or some portions of its sub-tree are selected
- ☐ Directory is not selected

Collection name: intappsvrDirTree

Attributes Learn more...
- ✓ Data authority
- ✓ Object authority
- ✓ Owner
- ✓ Primary group
- ✓ Authorization list

Figure 2 – Selecting files and directories to verify

Notice the Advanced button in the lower right corner of the page. This button is where you access the dialog where replacement variables are created. When you click the button, the following dialog is displayed:



Exmaple: /www/_${varName}$_
Exmaple: /www/INTAPPSVR/conf/_${varName}$_.properties

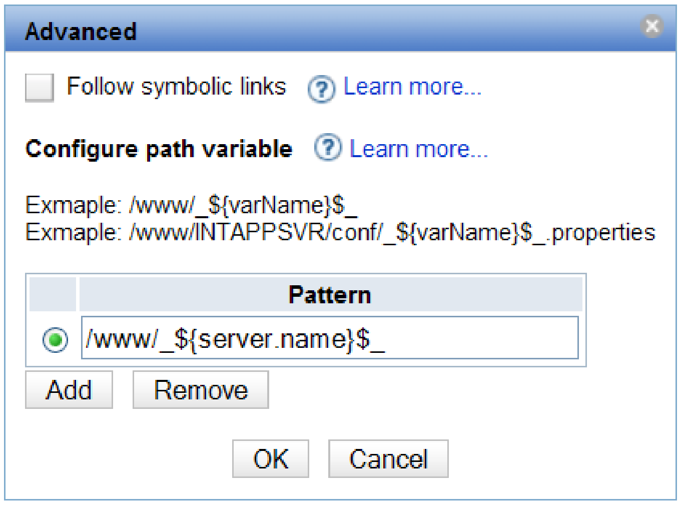
Pattern
This table is empty.

Figure 3 – Advanced dialog

The ‘Configure path variable’ table allows you to specify what part of the fixed path (the pattern) is replaced, or substituted, with a value that is provided at runtime. In the previous page we selected the entire ‘/www/INTAPPSVR’ directory tree. This directory tree represents the desired structure, including file/directory authorities, for any integrated Web application server instance. This means that the ‘ReplacementVariableDemo’ template can be used to verify the authority information for *any* integrated Web application server directory tree, except for one problem: the selected directory structure, /www/INTAPPSVR, contains the instance name (INTAPPSVR). That means the template only works to verify the INTAPPSVR integrated Web application server directory tree. Using it to verify another server, say INTAPPSVR**1**, would fail because INTAPPSVR1 is located in the /www/INTAPPSVR**1** directory tree. So here we have a perfect example of where a replacement variable is needed. What we need to be able to do is, at runtime, substitute the application server name into the directory path that is being verified by the template. To do this, we define a path replacement variable.

The ‘Configure path variable’ table allows one or more variables to be defined. The variable name can be whatever you like, although most special characters are not allowed. The replacement variable is defined as a pattern to match with the directory tree that is selected. So in our example, the directory tree chosen was /www/INTAPPSVR. We would like to have the INTAPPSVR portion of that directory tree be a replacement variable, which means its value is determined at runtime. To achieve this, the replacement variable is defined like this:

/www/_${server.name}$_



Advanced

Follow symbolic links Learn more...

Configure path variable Learn more...

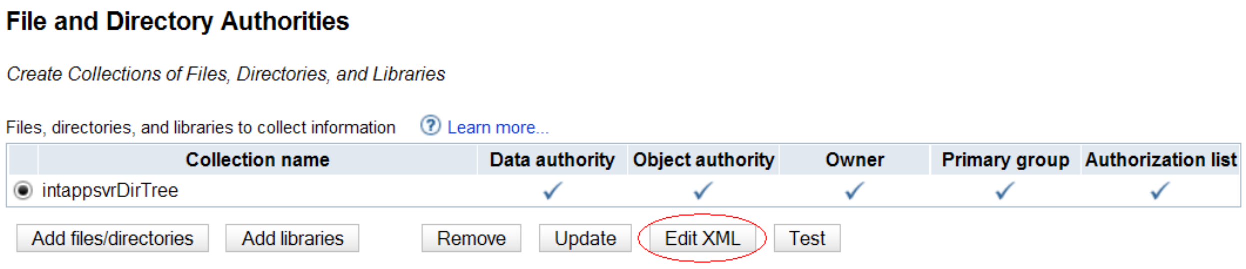
Exmaple: /www/_${varName}$_
Exmaple: /www/INTAPPSVR/conf/_${varName}$_.properties

	Pattern
●	/www/_${server.name}$_

Figure 4 – Defining a replacement variable

In the above example, the replacement variable is named ‘server.name’. The purpose of the leading _${ and trailing }$_ characters is purely to make the replacement variable easy to find and identify by ARE. It is important to understand that for the purposes of referring to the replacement variable name, the name is only the value between the braces – ‘server.name’ in the example above.

Click the OK button to accept the replacement variable you have defined, and also click the OK button on the Select Files and Directories page. This brings you back to the Collections page for file and directory authorities.



File and Directory Authorities

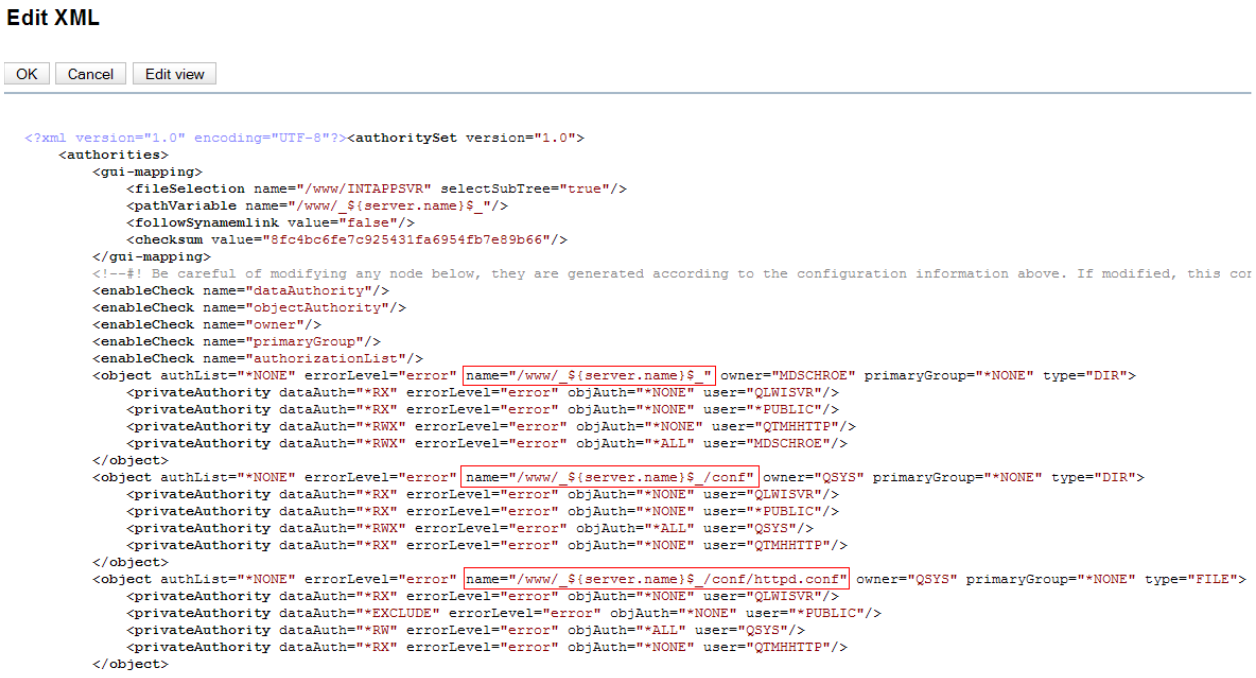
Create Collections of Files, Directories, and Libraries

Files, directories, and libraries to collect information Learn more...

	Collection name	Data authority	Object authority	Owner	Primary group	Authorization list
●	intappsvrDirTree	✓	✓	✓	✓	✓

Figure 5 – File directory and authorities collections page

If you would like to see exactly how the replacement variable is used by the deployment template editor, you can take a look at the XML that gets generated for the collection where you defined the replacement variable. To view the XML, click on the ‘Edit XML’ button on the Collections page (see figure 5 above). Clicking on the ‘Edit XML’ button takes you to the XML editor for the selected collection. Here is an example of what you see:



Edit XML

```
<?xml version="1.0" encoding="UTF-8"?><authoritySet version="1.0">
    <authorities>
        <gui-mapping>
            <fileSelection name="/www/INTAPPSVR" selectSubTree="true"/>
            <pathVariable name="/www/_${server.name}$_"/>
            <followSynamemlink value="false"/>
            <checksum value="8fc4bc6fe7c925431fa6954fb7e89b66"/>
        </gui-mapping>
        <!--#! Be careful of modifying any node below, they are generated according to the configuration information above. If modified, this cor
        <enableCheck name="dataAuthority"/>
        <enableCheck name="objectAuthority"/>
        <enableCheck name="owner"/>
        <enableCheck name="primaryGroup"/>
        <enableCheck name="authorizationList"/>
        <object authList="*NONE" errorLevel="error" name="/www/_${server.name}$_" owner="MDSCHROE" primaryGroup="*NONE" type="DIR">
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="QLWISVR"/>
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="*PUBLIC"/>
            <privateAuthority dataAuth="*RWX" errorLevel="error" objAuth="*NONE" user="QTMHHTTP"/>
            <privateAuthority dataAuth="*RWX" errorLevel="error" objAuth="*ALL" user="MDSCHROE"/>
        </object>
        <object authList="*NONE" errorLevel="error" name="/www/_${server.name}$_/conf" owner="QSYS" primaryGroup="*NONE" type="DIR">
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="QLWISVR"/>
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="*PUBLIC"/>
            <privateAuthority dataAuth="*RWX" errorLevel="error" objAuth="*ALL" user="QSYS"/>
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="QTMHHTTP"/>
        </object>
        <object authList="*NONE" errorLevel="error" name="/www/_${server.name}$_/conf/httpd.conf" owner="QSYS" primaryGroup="*NONE" type="FILE">
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="QLWISVR"/>
            <privateAuthority dataAuth="*EXCLUDE" errorLevel="error" objAuth="*NONE" user="*PUBLIC"/>
            <privateAuthority dataAuth="*RW" errorLevel="error" objAuth="*ALL" user="QSYS"/>
            <privateAuthority dataAuth="*RX" errorLevel="error" objAuth="*NONE" user="QTMHHTTP"/>
        </object>
```

Figure 6 – XML, including replacement variable, for the collection

As you can see, the replacement variable that was defined is embedded directly into the XML generated for this plugin.

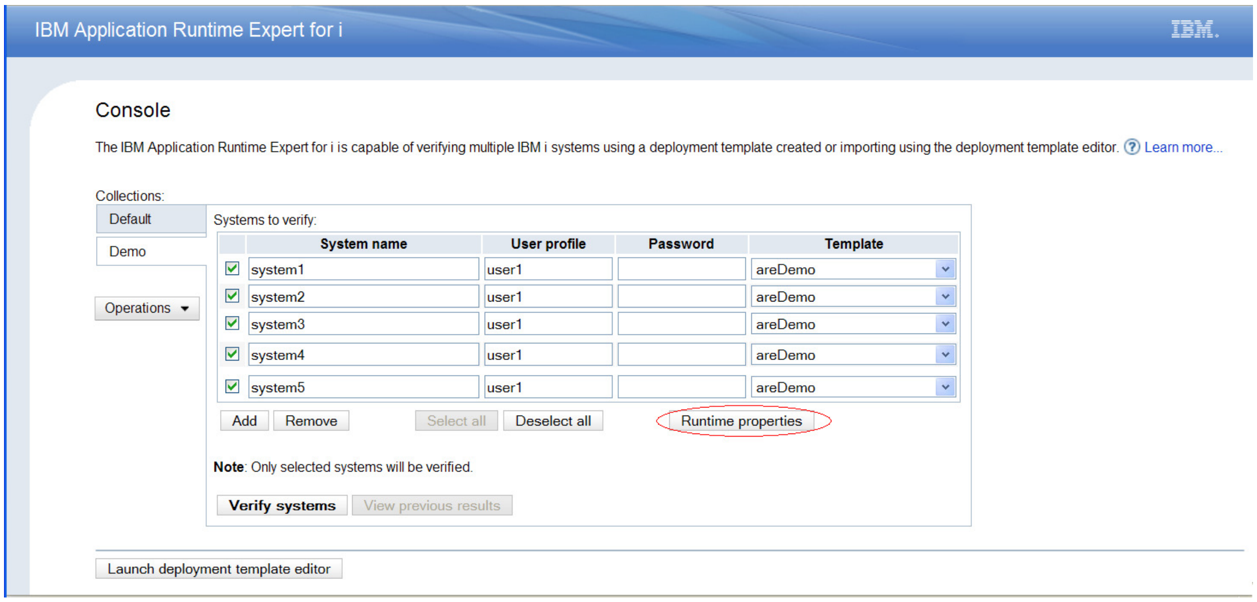
So now that the replacement variable has been defined, how do you specify the value for that variable when you go to use your template to verify a system? There are two different ways to specify the value for a replacement variable:

1. Specify the value via an input property. This is done via the ‘Runtime properties’ in the console, or by passing the value as a parameter on the command line when using the runARE.sh script to verify a system.
2. Specify the value programmatically. During the startup of the ARE core you can have Java code, written by you, be run in order to set the value for one or more replacement variables.

Both of these options are explained in detail below.

Option 1 – Verification via the Console GUI

When verifying a system using the ARE console, specifying the value for a replacement variable is simple. On the page where you supply system information, there is a ‘Runtime properties’ button.


IBM Application Runtime Expert for i

Console

The IBM Application Runtime Expert for i is capable of verifying multiple IBM i systems using a deployment template created or importing using the deployment template editor. Learn more...

Collections: Default, Demo

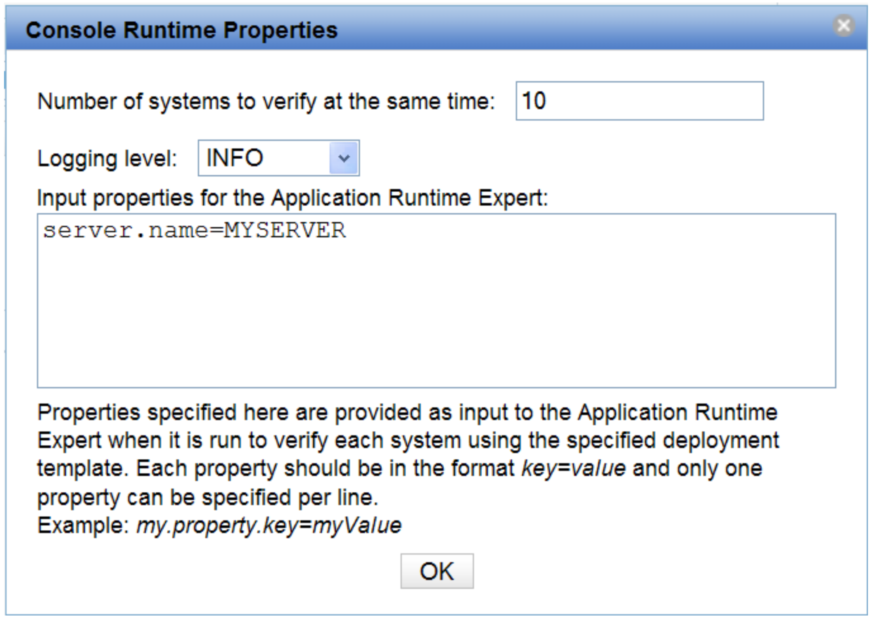
Systems to verify:

	System name	User profile	Password	Template
✓	system1	user1		areDemo
✓	system2	user1		areDemo
✓	system3	user1		areDemo
✓	system4	user1		areDemo
✓	system5	user1		areDemo

Note: Only selected systems will be verified.

Figure 7 – Runtime properties in the console

Clicking on the ‘Runtime properties’ button displays a dialog where runtime properties, to be passed to the ARE core, can be specified. The runtime properties are specified in the standard Java property format: *key=value* The *key* is the replacement variable name, which in this example is ‘server.name’. The value is whatever value you want substituted for the replacement variable at runtime. In figure 8 below, we are setting the replacement variable value to MYSERVER:


Console Runtime Properties

Number of systems to verify at the same time: 10

Logging level: INFO

Input properties for the Application Runtime Expert:

```
server.name=MYSERVER
```

Properties specified here are provided as input to the Application Runtime Expert when it is run to verify each system using the specified deployment template. Each property should be in the format *key=value* and only one property can be specified per line.
Example: *my.property.key=myValue*

Figure 8 – Runtime properties in the console

To continue our example, the console passes the 'server.name' property to the ARE core, which uses the specified value (MYSERVER) for the 'server.name' replacement variable during the system verification.

For more information about using the ARE console, see the 'Using the console to verify a system' document on the ARE product Web site:

http://www.ibm.com/systems/power/software/i/are/documentation.html

Option 2 – Specify the Value Programmatically

Background

In some cases, it may be much easier to write a small piece of Java code that can determine the appropriate value for a replacement variable, and then set that value prior to its use by the ARE core to verify a system. This is easily accomplished by creating a *Service*. A service, much like a plugin, is a Java class that is recognizable by the ARE core. Services are different than plugins in that conceptually a service is not intended to do system verification and report results; that's what a plugin does. Rather, a service is intended to be a provider, whether its providing information, functionality, or something else. Services are typically a provider to plugins, the ARE core itself, or both. In the context of this example, the service we create is a provider of replacement variable information (i.e what value should be assigned to our 'server.name' replacement variable). This information is used by any plugin where the 'server.name' replacement variable is used.

Creating a service is conceptually similar to creating a plugin. A full description of a service, and its methods, is outside the scope of this document. For the purposes of this example, we'll show you what a fully implemented service that sets the value for the replacement variable looks like – as you can see the amount of code to write is trivial.

```
/*
 *****************************************************************
 * LICENSE AND DISCLAIMER
 * ----------------------
 * This material contains IBM copyrighted sample programming source
 * code ( Sample Code ).
 * IBM grants you a nonexclusive license to compile, link, execute,
 * display, reproduce, distribute and prepare derivative works of
 * this Sample Code.  The Sample Code has not been thoroughly
 * tested under all conditions.  IBM, therefore, does not guarantee
 * or imply its reliability, serviceability, or function. IBM
 * provides no program services for the Sample Code.
 *
 * All Sample Code contained herein is provided to you "AS IS"
 * without any warranties of any kind. THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NON-INFRINGMENT ARE EXPRESSLY DISCLAIMED.
 * SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED
 * WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU.  IN NO
 * EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT,
 * SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THE SAMPLE
 * CODE INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS
 * INTERRUPTION, LOSS OF PROGRAMS OR OTHER DATA ON YOUR INFORMATION
 * HANDLING SYSTEM OR OTHERWISE, EVEN IF WE ARE EXPRESSLY ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGES.
 *
 *  <START_COPYRIGHT>
 *
 *  Licensed Materials - Property of IBM
 *
 *  5733-ARE
 *
 *  (c) Copyright IBM Corp. 2010, 2010
 *  All Rights Reserved
 *
 *  U.S. Government Users Restricted Rights - use,
 *  duplication or disclosure restricted by GSA
 *  ADP Schedule Contract with IBM Corp.
 *
 *  Status: Version 1 Release 0
 *  <END_COPYRIGHT>
 */
package com.ibm.are.sample;

import java.util.List;
```

```java
import com.ibm.are.common.Version;
import com.ibm.are.service.AutoStartSingletonService;
import com.ibm.are.service.BaseSingletonService;
import com.ibm.are.xml.Translator;

public class InitReplacementVariables extends BaseSingletonService implements AutoStartSingletonService {

      protected void startImpl(List startArgs) {
            //
            // This is where you can set your replacement variable's value.  The value you set is global
            // in the entire ARE core environment, which is why it is important to make your replacement
            // variable name somewhat unique.  Because this class implements the AutoStartSingletonService
            // interface, the code in this method is guaranteed to be run by the ARE core *before*
            // any plugins are run.  This allows you to set the replacement variable's value prior to any
            // possible usage of that replacement variable by a plugin.
            //
            // Replacement variables are managed by a class in the ARE core environment called a Translator.
            // The Translator maintains a single, global map of replacement variable names and
            // their associated values.
            //
            // This means that once you set the replacement variable value, it is available for use by
            // every plugin that is run during a system verification.
            //
            // Set the replacement variable to its desired value here.  Any specific processing needed
            // to determine the value for the replacement variable can be done here as well.
            //
            Translator.setReplacementVariable("server.name", "MYSERVER");
      }

      protected void stopImpl(List stopArgs) {
      }

      public String getCommonName() {
            return "Initialize Replacement Variables";
      }

      public String getDescription() {
            return "A service to set the value for one or more replacement variables";
      }

      public Version getVersion() {
            return new Version(1, 0, 0);
      }
}
```

The real interesting method in the example class above is the startImpl() method. In this method, we set the ‘server.name’ replacement variable value to MYSERVER. Obviously, we could have done any amount of processing and lookup in this method to determine the appropriate value to set this replacement variable to.

So *how* does this code actually get run? Note that this example service implements the AutoStartSingletonService interface. This interface tells the ARE core that this service should be automatically started and, when a service is started, its startImpl() method is invoked by the ARE core. So the answer to “how does this code run” is that the ARE core runs it for you automatically.

So *when* does this code actually run? This is a very important question, because we need to ensure the replacement variable value gets set prior to any plugins (which may use the replacement variable) being run. Again, the answer lies in the fact that this example service implements the AutoStartSingletonService interface. When the ARE core is starting, it builds a list of all auto-start services. Once the core has been fully initialized, these auto-start services are started prior to any plugins. So setting the replacement variable value in this auto-start service guarantees it gets set prior to any plugin making use of the value.

Compiling The Service

You can use the standard Java compiler tools to compile your service. Because you are extending and using classes that are part of the ARE core, you need to include the ARE core jar in your classpath when you compile. The ARE core jar can be found here:

/QIBM/ProdData/OS/OSGi/healthcheck/lib/arecore.jar

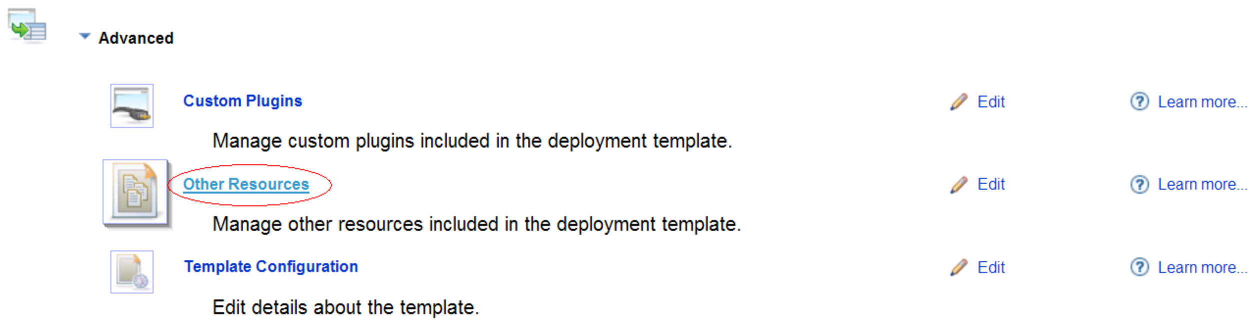
If your plugin uses the IBM Toolbox for Java, you also need to add that jar to your classpath. That jar can be found here:

/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar

Note that both of the aforementioned jar files are already part of the ARE runtime classpath, so you do not need to do anything to have access to ARE core or IBM Toolbox for Java classes at runtime.

Adding the Service to Your Template

Once the service has been compiled, you need to add the service to your template. To add the service, you need to make use of the ‘Advanced > Other Resources’ page.


Advanced

Custom Plugins — Edit — Learn more...

Manage custom plugins included in the deployment template.

Other Resources — Edit — Learn more...

Manage other resources included in the deployment template.

Template Configuration — Edit — Learn more...

Edit details about the template.

Figure 9 – Adding other resources to a template

For more information about adding other resources to a template, see the ‘Adding other resources to a template’ document on the ARE product Web site:

http://www.ibm.com/systems/power/software/i/are/documentation.html