BEA WebLogic Server R
Harvester and Watch Notification

Version: 10.3 Tech Preview

Document Date: October 2007

Table of Contents

Harvester Enhancements

This section describes the enhancements made to the Harvester capabilities in WebLogic Server 10.3.

Harvesting of nested complex attributes and collections

Previous releases of WebLogic Server only allowed harvesting of attributes that are simple data types. There was no mechanism to collect data from nested complex data types or collections such as lists, arrays and maps. The WebLogic Diagnostics Framework (WLDF) Harvester now supports collecting and archiving data from MBean attributes that are nested bean structures or collections.

The drilldown syntax for nested attributes uses the dot notation to traverse complex structures. For example, `Foo.Bar.Yo` would mean that we are interested in harvesting the `Yo` attribute whose type is one of the simple types but is nested within the `Bar` bean which in turn is a member of the `Foo` bean structure.

In WebLogic Server there are only few examples of Runtime MBeans having complex attributes that are not MBeans but simple Java beans. If a user wanted to harvest the nested `State` property of the `HealthState` attribute on the ServerRuntime MBean, the diagnostics descriptor entry would be:

```
<harvester>
  <sample-period>10000</sample-period>
  <harvested-type>
   <name>weblogic.management.runtime.ServerRuntimeMBean</name>
   <harvested-attribute>HealthState.State</harvested-attribute>
  </harvested-type>
</harvester>
```

To harvest the elements of an array or a list, the Haster supports a subscript notation wherein a value is referred to by its index position in the aray or lst collection. For example to refer to the first element in the array attribute `URLPatterns` in the ServletRuntime MBean, refer to it as `URLPatterns[0]`. To refer to all the elements, specify `URLPatterns[*]`. For example:

```
<harvester>
  <sample-period>10000</sample-period>\
  <harvested-type>
   <name>weblogic.management.runtime.ServletRuntimeMBean</name>
   <harvested-attribute>URLPatterns[0]</harvested-attribute>
  </harvested-type>
</harvester>
```

To harvest the elements of a map, each individual value is referred by the key enclosed in parenthesis. Multiple keys can be specified as a comma delimited list, in which case values corresponding to specified keys in the map will be harvested.

For example, the following code example harvests the value from the map with key `Foo`:

```
<harvested-attribute>MyMap(Foo)</harvested-attribute>
```

For example, the following code example harvests the value from the map with keys `Foo` and `Bar`:

```
<harvested-attribute>MyMap(Foo,Bar)</harvested-attribute>
```

The `%` character is used as a wild card within a key specification. For example, the following code example harvests all values from the map if their keys start with `Foo` and end with `Bar`:

```
<harvested-attribute>MyMap(Foo%Bar)</harvested-attribute>
```

To harvest all values from a map, use a `*` key. For example:

```
<harvested-attribute>MyMap(*)</harvested-attribute>
```

Array and map attributes can be nested within other attributes. For example, if the MBean has a JavaBean attribute `MyBean` which has a nested attribute `MyMap` of type map, the following code example harvests the value from the map whose key is `Foo`:

```
<harvested-attribute>MyBean.MyMap(Foo)</harvested-attribute>
```

ObjectName Pattern Matching

In prior releases of WebLogic Server, the Harvester instance specification had to be in the form of an ObjectName. This requirement could be cumbersome and error prone since the ObjectNames of WebLogic Server MBeans are quite verbose in nature.

To alleviate this user problem, the WLDF now allows:

- Specification of an instance as an ObjectName in a non-canonical form.
- Specification of an instance as an ObjectName query pattern.
- The use of zero or more wildcard (`*`) characters in any of the values in the property list of an ObjectName, for example., `Name=*`.
- The use of zero or more wildcard (`*`) characters to replace any character sequence in a canonical ObjectName string. In this case, the burden is on the user to ensure that any properties of the ObjectName that are not wildcarded are in canonical form.

This first code example indicates that all instances of the WorkManagerRuntime MBean are to be harvested. Note that this is equivalent to not providing any instance-name qualification at all to the `<harvested-type>` declaration.

```
<harvested-type>
   <name>weblogic.management.runtime.WorkManagerRuntimeMBean</name>
   <harvested-instance>*</harvested-instance>
   <known-type>true</known-type>
   <harvested-attribute>PendingRequests</harvested-attribute>

</harvested-type>
```

This second code example shows a JMX ObjectName pattern as the `<harvested-instance>` value:

```
<harvested-type>
   <name>com.acme.WileECoyoteMBean</name>
   <harvested-instance>adomain:Type=MyType,*</harvested-instance>
```

```
   <known-type>false</known-type>
   </harvested-type>
```

This third code example illustrates how some of the values in the ObjectName property list are wildcarded:

```
<harvested-type>
   <name>com.acme.WileECoyoteMBean</name>
   <harvested-instance>adomain:Type=My*,Name=*,*</harvested-instance>
   <known-type>false</known-type>
   </harvested-type>
```

The final code example illustrates harvesting all harvestable attributes on all instaces of `com..acme.WileECoyoteMBean` where the instance name contains the stringName=mybean:

```
<harvested-type>
   <name>com.acme.WileECoyoteMBean</name>
   <harvested-instance>*Name=mybean*</harvested-instance>
   <known-type>true</known-type>
   </harvested-type>
```

Object names of several WebLogic Server MBeans embed the name of the WebLogic server instance where the MBeans reside. In earlier releases of WebLogic Server, it was not possible to specify instances without knowing the server where the MBeans would be created. The pattern based instance specification allows instances to be specified in server independent manner.

Harvesting from the DomainRuntime MBeanServer

By default, the Harvester gathers data from MBeans in the Runtime MBeanServer running in a WebLogic Server instance only. It was not possible earlier to harvest MBeans resident in the DomainRuntime MBeanServer that is running only on the Admin Server in the domain which represents a federated view of the domain configuration and runtime data via JMX. To support this use case, the harvester configuration now allows specification of a namespace attribute.

The Namespace attribute can have one of the two values:

- `serverRuntime` which is the default
- `domainRuntime` which harvests data from the DomainRuntime MBeanServer running on the Admin Server.

Note that it is efficient to harvest data from the local Runtime MBeanServer. For the most part, it is preferable to harvest data locally. However, there are a few MBeans types that only live on the DomainRuntime MBeanServer on the Admin Server (for example, the ServerLifecycleRuntimeMBean). These MBeans can now be harvested.

The following code example illustrates the configuration of the ServerLifecyeleRuntimeMBean from the DomainRuntime MBeanServer.

```
 <harvested-type>
    <name>weblogic.management.runtime.ServerLifeCycleRuntimeMBean</name>
    < namespace>domainRuntime</ namespace >
    <known-type>true</known-type>
    <harvested-attribute>StateVal</harvested-attribute>
 </harvested-type>
```

Note that the Namespace attribute is only applicable when the WLDF module is targeted to the Admin Server.

Watch Enhancements

This section describes the enhancements made to the Watch capabilities in WebLogic Server 10.3.

Complex Attribute Support

Similar to the support for nested complex and collections in the Harvester configuration, the Harvester Watch rules also fully supports the complex drill down syntax of the Harvester. While configuring instance based rules, object names for instances can be specified as patterns as described previously.

For example:

```
${ //[weblogic.management.runtime.ServerRuntimeMBean]//HealthState.State} > 0
```

The following code example illustrates how to drill down into the nested map structure:

```
<watch>
  <name>w1</name>
  <rule-expression>
${com.bea:Name=hello,Type=WLDFInstrumentationRuntime,*//MethodInvocationStatist
ics(*)(*)(*)(count)}&gt;=1
  </rule-expression>
  <alarm-type>AutomaticReset</alarm-type>
  <notification>smtp1</notification>
</watch>
```

Wildcard Support

In prior releases of WebLogic Server, instance based Harvester Watch rules required a full ObjectName to be specified. The wildcard support that is available for Harvester configuration is also being supported for the Watch rule specifications that have an instance name.

The following code example specifies that the `OpenSocketsCurrentCount` attribute for all instances of the ServerRuntime MBean that begin with the name `managed` should be used:

```
${com.bea:*Name=managed*Type=ServerRuntime*//OpenSocketsCurrentCount}
```

Alternately, the JMX ObjectName query patterns can be used:

```
${mydomain:Key1=MyMBean,*//simpleAttribute}
```

Note that the code example uses the ObjectName pattern syntax supported by JDK 1.5.

Namespace Support

With the integration of the DomainRuntime MBeanServer with the Harvester, it is now possible to reference such metrics from a Harvester WatchRule. In order to do so, the variable syntax for a Harvester WatchRule now allows the namespace specification, where the metric is registered.

The Harvester WatchRule variable syntax for both instance-based and type-based variables is as follows:

- Instance-based variable syntax:

  `${<namespace>//<object-name>//<attribute-name>}`

- Type-based variable syntax:

  `${<namespace>//[<type-name>]//<attribute-name>}`

where `<namespace>` is one of the following values:

- `serverRuntime`
- `domainRuntime`

In each case, the `<namespace>` specification is optional. If no namespace is specified, the default value will be `serverRuntime` for backward compatibility.