Configure Custom Exit Points for Java

AppDynamics provides default automatic discovery for commonly-used backends. If a backend used in your environment is not discovered, first compare the list of default backends to determine whether you need to modify the default configuration. If it is not on the list then configure a custom exit point according to these instructions.

Default Backends Discovered by the App Agent for Java

The default backends for Java are:

- HTTP
- JDBC
- JMS
- Cassandra
- IBM MQ
- RABBITMQ
- RMI
- Thrift
- Web Service

To configure a default backend see Configure Backend Detection for Java.

Configure Custom Exit Points for Java Backends

Configure Custom Exit Points

Use custom exit points to identify backend types that are not automatically detected, such as file systems, mainframes, and so on. For example, you can define a custom exit point to monitor the file system read method. After you have defined a custom exit point, the backend appears on the flow map with the type-associated icon you selected when you configured the custom exit point.

You define a custom exit point by specifying the class and method used to identify the backend. If the method is overloaded, you need to add the parameters to identify the method uniquely.

You can restrict the method invocations for which you want AppDynamics to collect metrics by specifying match conditions for the method. The match conditions can be based on a parameter or the invoked object.

You can also optionally split the exit point based on a method parameter, the return value, or the invoked object.

You can also configure custom metrics and transaction snapshot data to collect for the backend.

To create a custom exit point

1. From the left navigation pane, click **Configure -> Instrumentation** and select the **Backend Detection** tab.

2. Select the application or tier for which you are configuring the custom exit point.

3. Ensure **Use Custom Configuration for this Tier** is selected.
Backend detection configuration is applied on a hierarchical inheritance model. See Hierarchical Configuration Model.

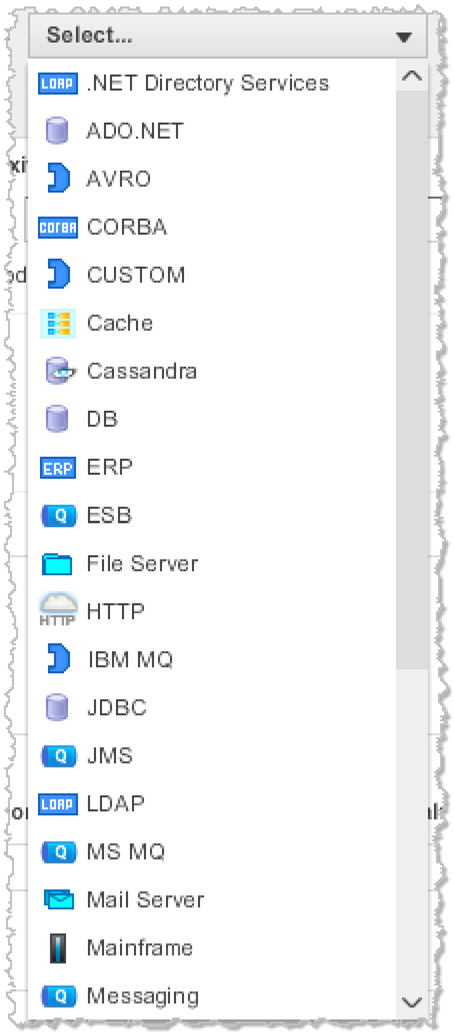
4. Scroll down to **Custom Exit Points** and click **Add** (the + icon).

5. In the **Create Custom Exit Point** window, click the **Identification** tab if it is not selected.

6. Enter a name for the exit point. This is the name that identifies the backend.

7. Select the type of backend from the **Type** drop-down menu.

This field controls the icon and name that appears on the flow maps and dashboards. Some of the values are shown in this screen shot:



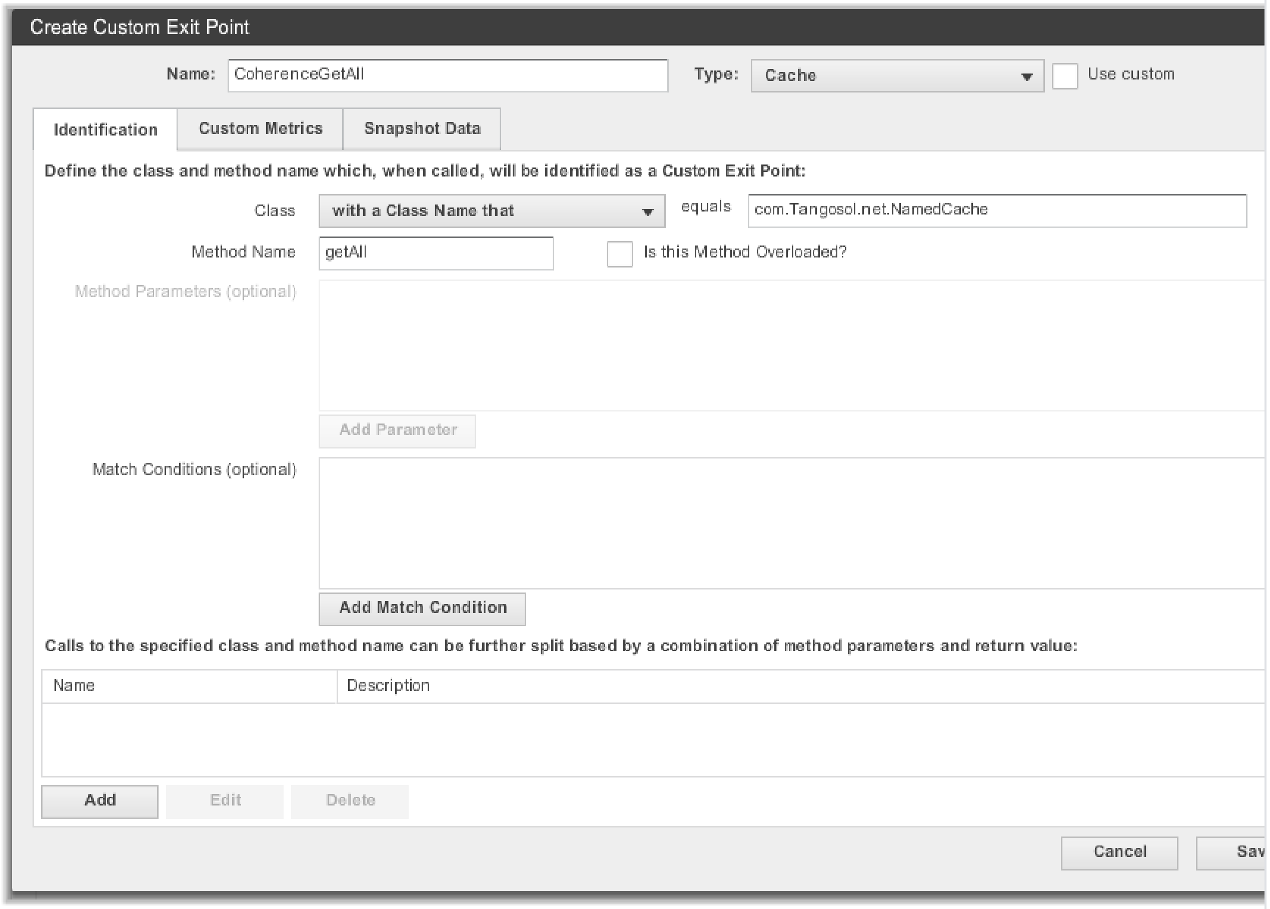
If the type is not listed, you can check **Use Custom** and enter a string to be used as the name on the dashboards.

8. Configure the class and method name that identify the custom exit point.
If the method is overloaded, check the Overloaded check box and add the parameters.

9. If you want to restrict metric collection based on a set of criteria that are evaluated at runtime, click **Add Match Condition** and define the match condition(s).
For example, you may want to collect data only if the value of a specific method parameter contains a certain value.

10. Click **Save**.

The following screenshot shows a custom exit point for a Cache type backend. This exit point is defined on the getAll() method of the specified class. The exit point appears in flow maps as an unresolved backend named CoherenceGetAll.



Create Custom Exit Point

Name: CoherenceGetAll

Type: Cache

☐ Use custom

Define the class and method name which, when called, will be identified as a Custom Exit Point:

Class: with a Class Name that equals com.Tangosol.net.NamedCache

Method Name: getAll

☐ Is this Method Overloaded?

Method Parameters (optional)

Match Conditions (optional)

Calls to the specified class and method name can be further split based by a combination of method parameters and return value:

Name	Description

To split an exit point

1. In the Backend Detection configuration window, click **Add**.

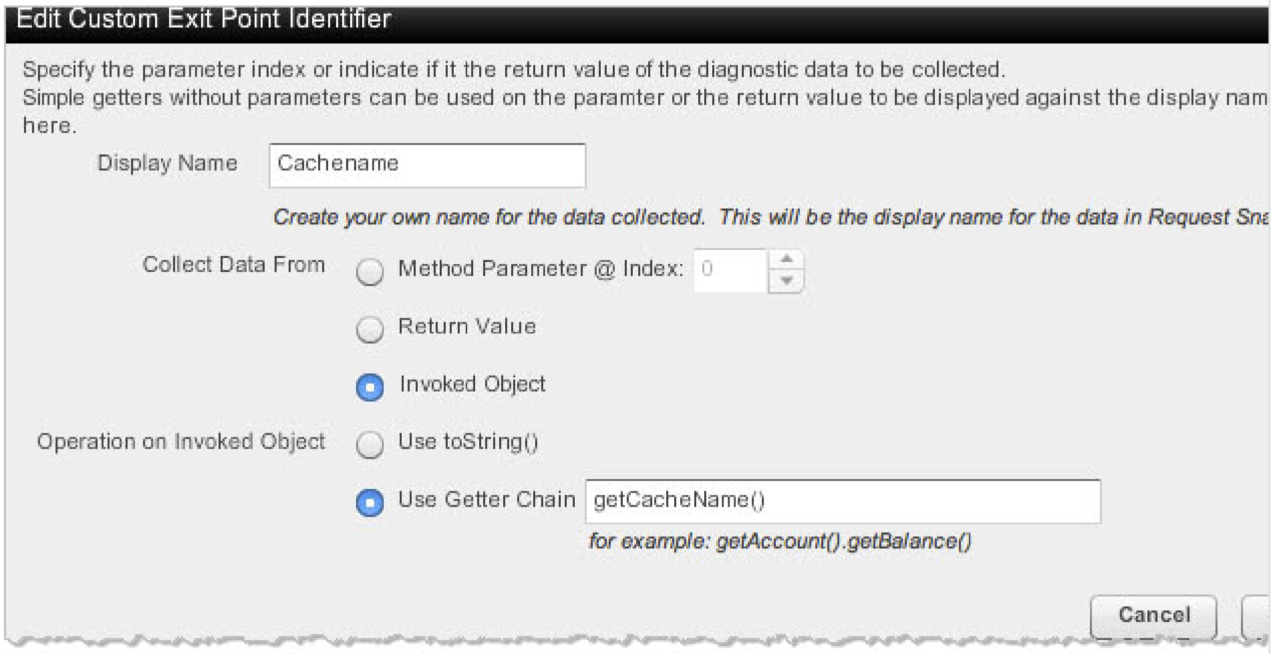
2. Enter a display name for the split exit point.

3. Specify the source of the data (parameter, return value, or invoked object).

4. Specify the operation to invoke on the source of the data: **Use toString()** or **Use Getter Chain** (for complex objects).

5. Click **Save**.

The following example shows a split configuration of the previously created CoherenceGetAll exit point based on the getCacheName() method of the invoked object.



Specify the parameter index or indicate if it the return value of the diagnostic data to be collected.
Simple getters without parameters can be used on the paramter or the return value to be displayed against the display nam
here.

Display Name: Cachename

Create your own name for the data collected. This will be the display name for the data in Request Sna

Collect Data From: Method Parameter @ Index: 0 / Return Value / [✓ Invoked Object]

Operation on Invoked Object: Use toString() / [✓ Use Getter Chain] getCacheName()

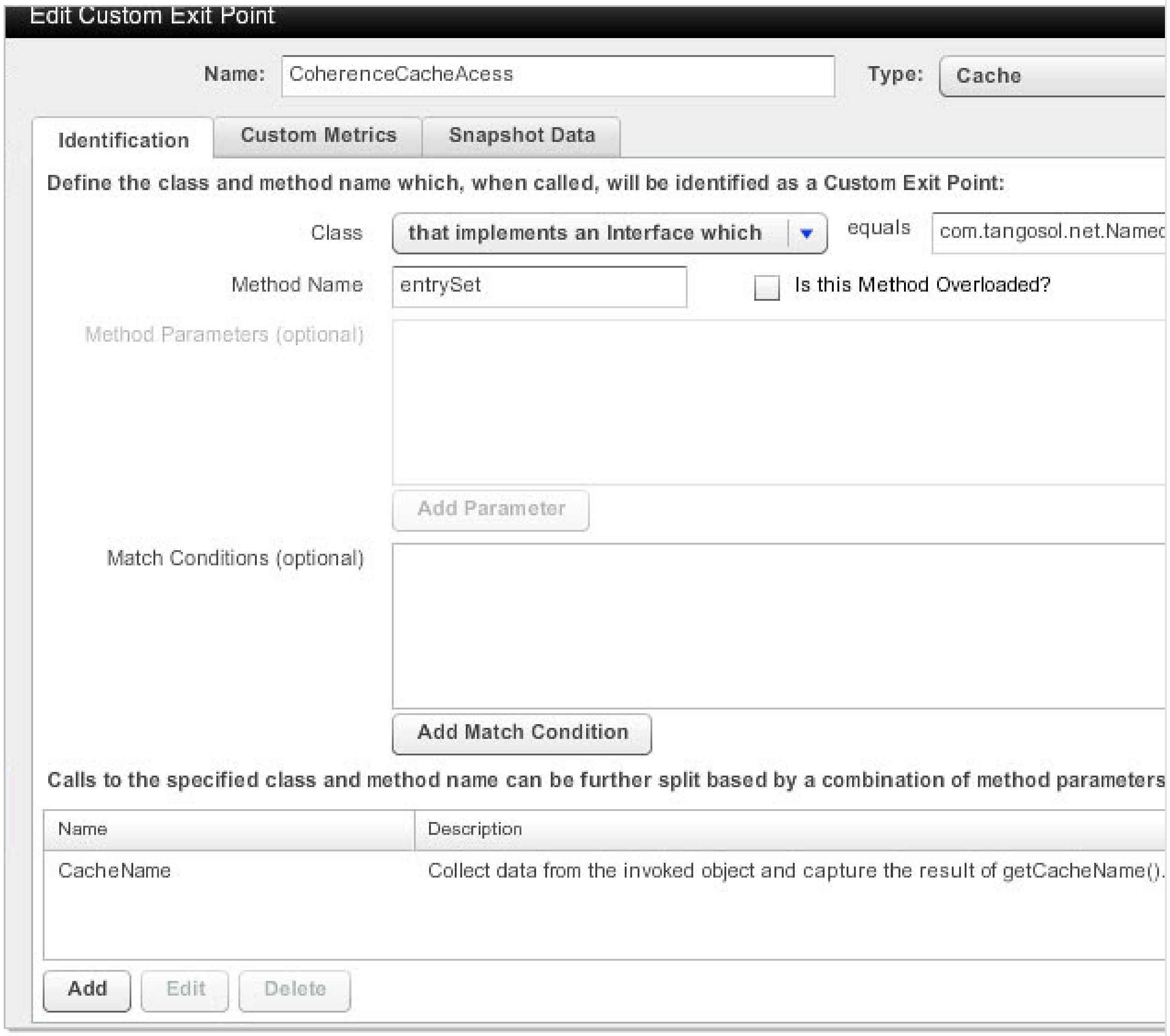
for example: getAccount().getBalance()

To group an exit point

You can group methods as a single exit point if the methods point to the same key.

For example, ACME Online has an exit point for NamedCache.getAll. This exit point has a split configuration of getCacheName() on the invoked object as illustrated in the previous screen shot.

Suppose we also define an exit point for NamedCache.entrySet. This is another exit point, but it has the split configuration that has getCacheName() method of the invoked object.


Edit Custom Exit Point

Name: CoherenceCacheAcess

Type: Cache

Identification
Custom Metrics
Snapshot Data

Define the class and method name which, when called, will be identified as a Custom Exit Point:

Class: that implements an Interface which equals com.tangosol.net.Named

Method Name: entrySet

Is this Method Overloaded?
Method Parameters (optional)
Add Parameter
Match Conditions (optional)
Add Match Condition

Calls to the specified class and method name can be further split based by a combination of method parameters

Name	Description
CacheName	Collect data from the invoked object and capture the result of getCacheName().

If the getAll() and the entrySet() methods point to the same cache name, they will point to the same backend.

Matching name-value pairs identify the back-end. In this case, only one key, the cache name, has to match. So, here both exit points have the same name for the cache and they resolve to the same backend.

To define custom metrics for a custom exit point

Custom metrics are collected in addition to the standard metrics.

The result of the data collected from the method invocation must be an integer value, which is either averaged or added per minute, depending on your data roll-up selection.

To configure custom business metrics that can be generated from the Java method invocation:

1. Click the **Custom Metrics** tab.

2. Click **Add**.

3. In the **Add Custom Metric** window type a name for the metric.

4. Select **Collect Data From** to specify the source of the metric data.

5. Select **Operation on Method Parameter** to specify how the metric data is processed.

6. Select how the data should be rolled up (average or sum) from the Data Rollup drop-down menu.

7. Click **Create Custom Metric**.

To define transaction snapshot data collected

1. Click the **Snapshot Data** tab.

2. Click **Add**.

3. In the Add Snapshot Data window, enter a display name for the snapshot data.

4. Select the Collect Data From radio button to specify the source of the snapshot data.

5. Select the Operation on Method Parameter to specify how the snapshot data is processed.

5. Click **Save**.

Learn More

- Configurations for Custom Exit Points for Java