# Configure and Use Custom Memory Structures for Java



This topic describes how to configure custom memory structures and monitor large coarse grained custom cache objects.

AppDynamics provides different levels of memory monitoring for multiple JVMs. Ensure custom memory structures are supported in your JVM environment. See JVM Support.

> **CPU Overhead Caution**
> Due to high CPU usage, Custom Memory Structures monitoring should only be enabled while debugging a problem.

## Custom Memory Structures and Memory Leaks

Typically custom memory structures are used as caching solutions. In a distributed environment, caching can easily become a source of memory leaks. AppDynamics helps you to manage and track memory statistics for these memory structures.

AppDynamics provide visibility into:

- Cache access for slow, very slow,  and stalled business transactions
- Usage statistics, rolled up to the Business Transaction level
- Keys being accessed
- Deep size of internal cache structures

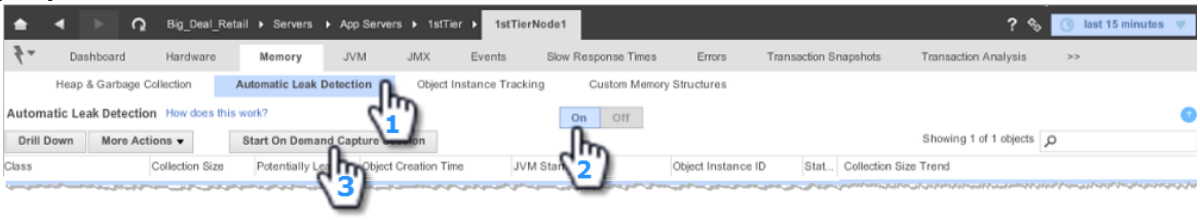
### Using Automatic Leak Detection vs Monitoring Custom Memory Structures

The automatic leak detection feature captures memory usage data for all map and collection libraries in a JVM session. However, custom memory structures might or might not contain

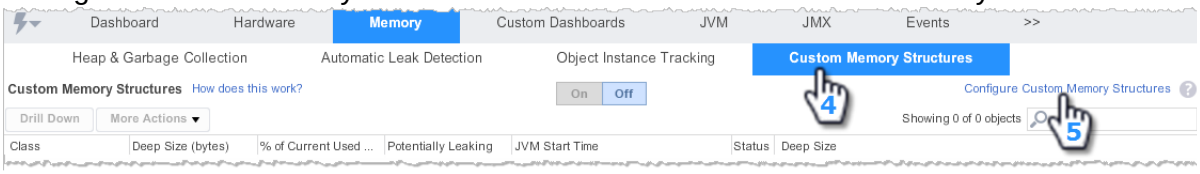
collections objects. For example, you may have a custom cache or a third party cache like Ehcache for which you want to collect memory usage statistics. Using custom memory structures, you can monitor any custom object created by the app and the size data can be traced across JVM restarts. Automatic leak detection is typically used to identify leaks and custom memory structures is used to monitor large coarse grained custom cache objects.

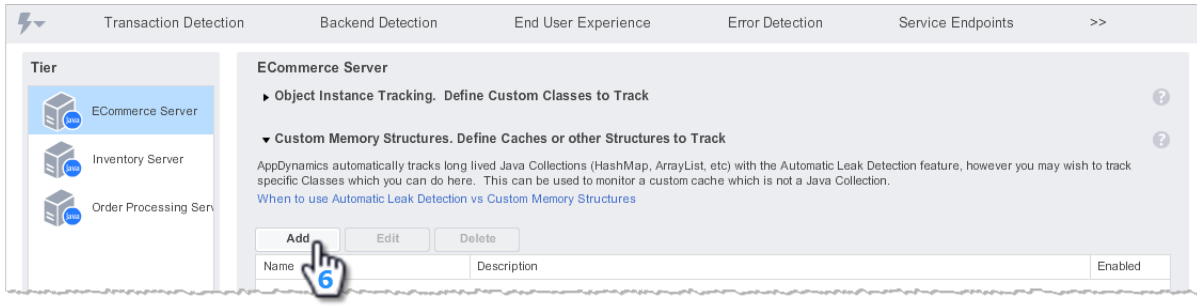
The following provides the workflow for configuring, monitoring, and troubleshooting custom memory structures. You must configure custom memory structures manually.

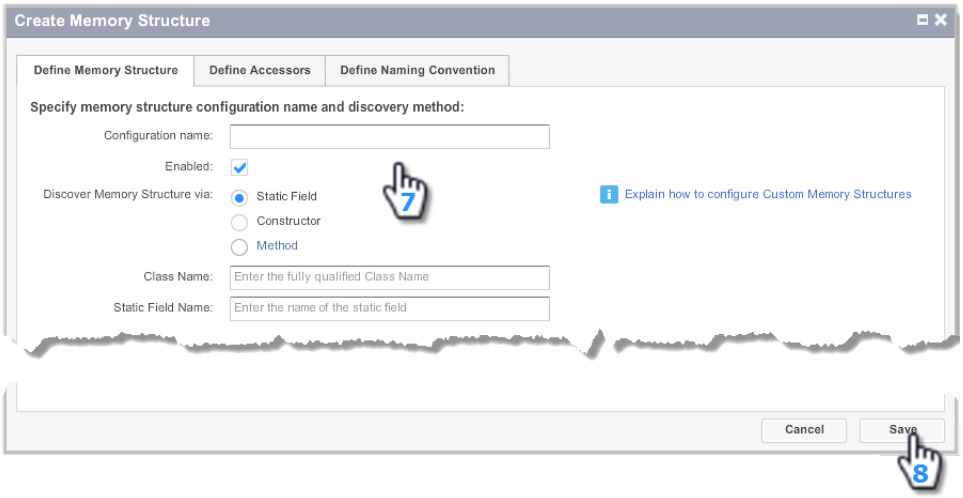
1. On the Node Dashboard, use the Automatic Leak Detection, On Demand Capture Session feature to determine which classes aren't being monitored, for example, custom or third party caches such as EhCache.



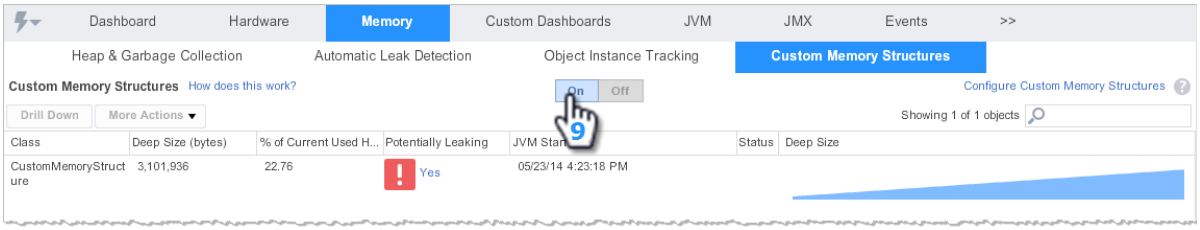

2. Configure Custom Memory Structures and then restart the JVM if necessary.




3. Turn on Custom Memory Structures monitoring to detect potential memory leaks in the custom memory structures you have configured.




4. Drill down into leaking memory structures for details that will help you determine where the leak is.

### To identify custom memory structures

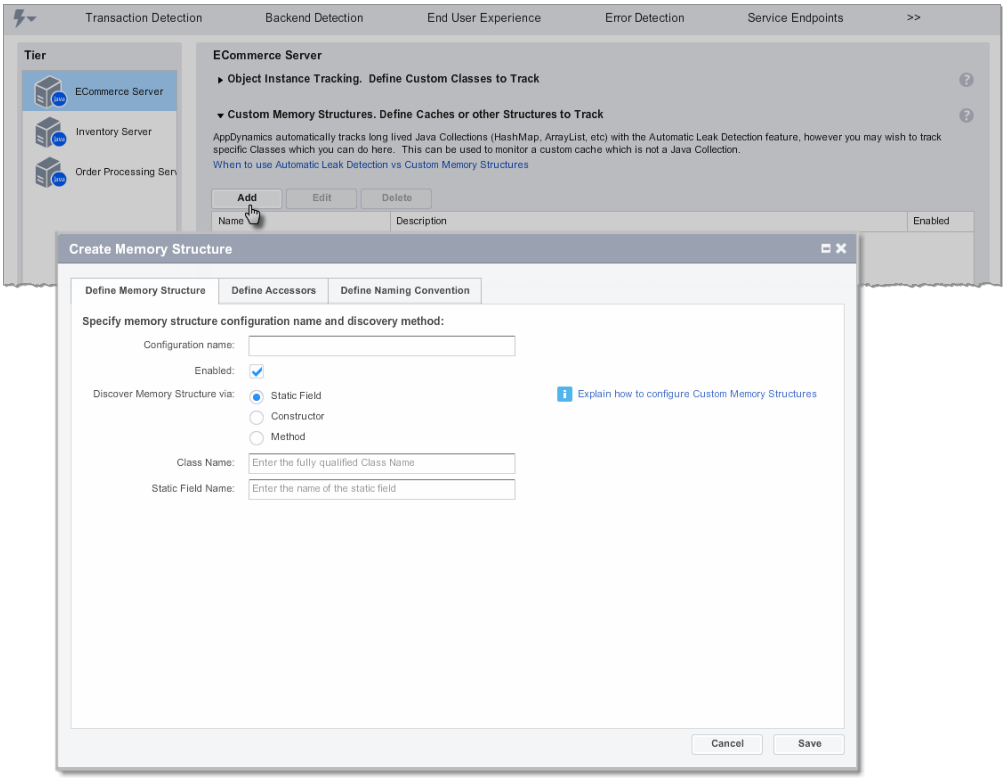
1. On the Automatic Leak Detection subtab of the Memory tab, click **On**.
2. Click **Start On Demand Capture Session** to capture information on which classes are accessing which collections objects. Use this information to identify custom memory structures.

AppDynamics captures the top 1000 classes, by instance count.

### To Add a Custom Memory Structure

These instructions provide an alternate method to accessing the Custom Memory Structures pane than the workflow above shows. Use the method that is most convenient to you.

1. From the left navigation pane select **Configure -> Instrumentation**.
2. In the Tier panel, click the tier for which you want to configure a custom memory structure and then click **Use Custom Configuration for this Tier**.
3. On the top menu, click the Memory Monitoring tab.
4. In the Custom Memory Structures panel, click **Add** to add a new memory structure.




a. In the Create Memory Structure window
   - Specify the configuration name.
   - Click **Enabled**.
   - Specify the discovery method.
     The discovery method provides three options to monitor the custom memory structure. The discovery method determines how the agent gets a reference to the custom memory structure. AppDynamics needs this reference to monitor the size of the structure. Select one of the three options for the discovery method:
     - Discover using Static Field.
     - Discover using Constructor.
     - Discover using Method.

   In many cases, especially with caches, the object for which a reference is needed is created early in the life cycle of the application.

| Example for using static field | Example for using Constructor | Example f |
|---|---|---|

| | | |
|---|---|---|
| `public class CacheManager { private static Map userCache<String> User>; }` | `public class CustomerCache { public CustomerCache(); }` | `pub pub get {} }` |
| Notes: Monitors deep size of this Map. | Notes: Monitors deep size of CustomerCache object(s). | Notes: Mo |

Restart the JVM after the discovery methods are configured to get the references for the object.

b. (Optional) Define accessors.
Click **Define Accessors** to define the methods used to access the custom memory structure. This information is used to capture the code paths accessing the custom memory structure.

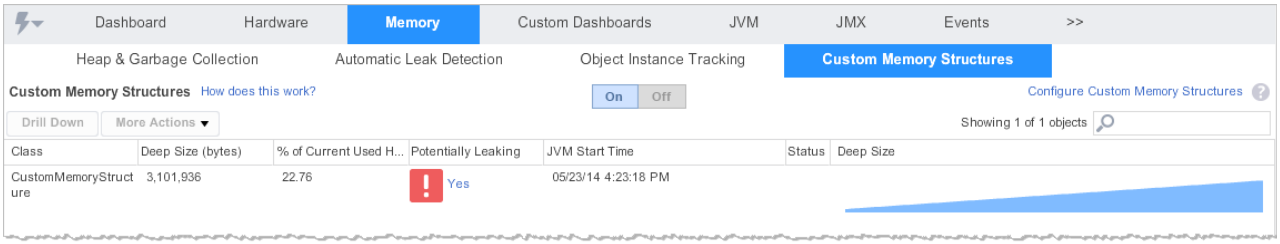
c. (Optional) Define the naming convention.
Click **Define Naming Convention**. These configurations differentiate between custom memory structures.

There are situations where more than one custom Caches are used, but only few of them need monitoring. In such a case, use the **Getter Chain** option to distinguish amongst such caches. For all other cases, use either value of the field on the object or a specific string as the object name.

d. Click **Save** to save the configuration.

## Identifying Potential Memory Leaks

Start monitoring memory usage patterns for custom memory structures. An object is automatically marked as a potentially leaking object when it shows a positive and steep growth slope. The Memory Leak Dashboard provides the following information:




The Custom Memory Structures dashboard provides the following information:

- Class: The name of the class or collection being monitored.
- Deep Size (bytes): The upper boundary of memory available to the structure. The

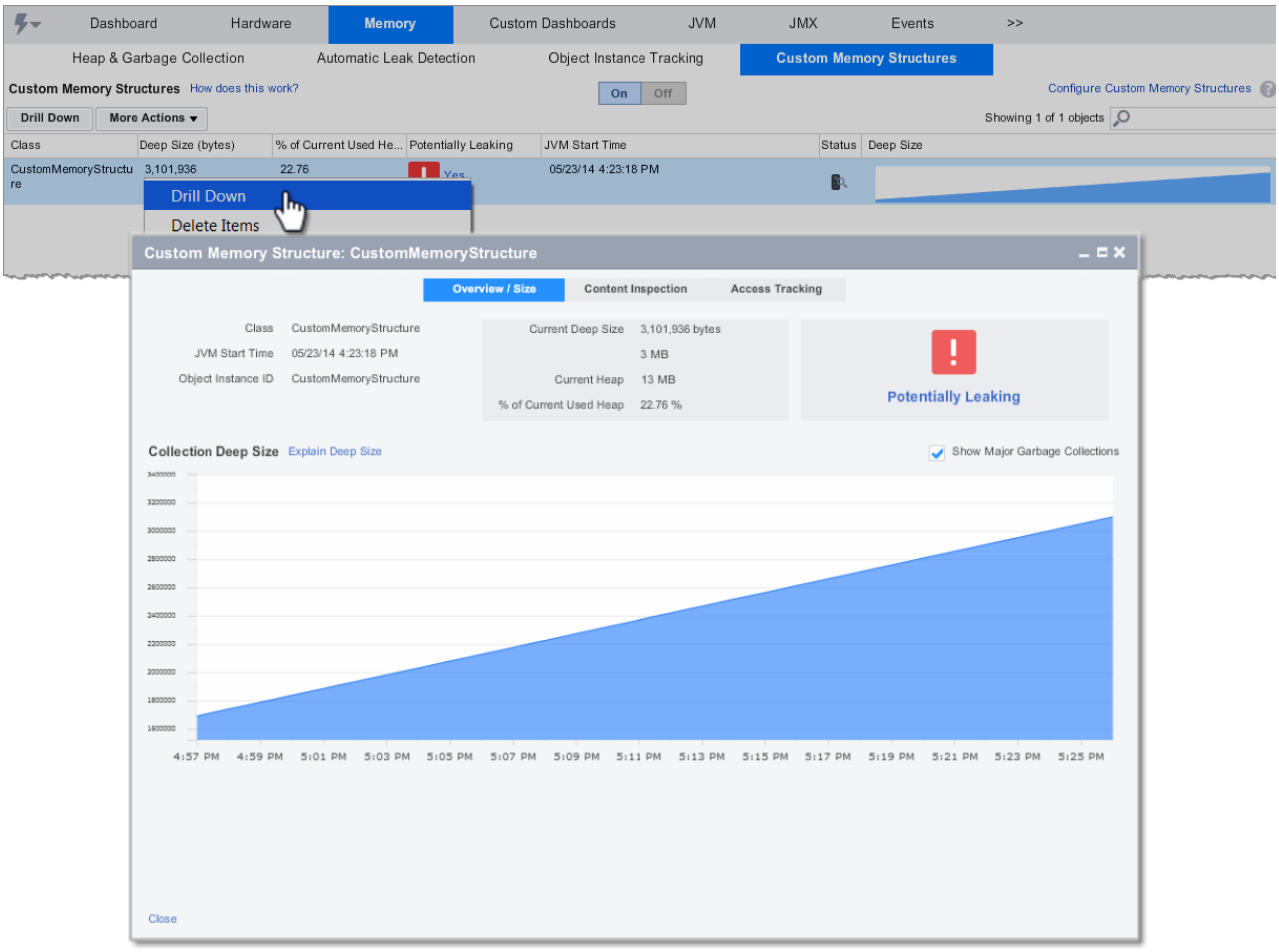
deep size is traced across JVM restarts

- % of Current Used Heap: The percentage of memory available for dynamic allocation.
- Potentially Leaking: Potentially leaking collections are marked as red. We recommend that you start a diagnostic session on potentially leaking objects.
- JVM Start Time: Custom Memory Structures are tracked across JVM restarts.
- Status: Indicates if a diagnostic session has been started on an object.
- Deep Size: A positive and steep growth slope indicates potential memory leak.

After the potentially leaking collections are identified, start the diagnostic session.

## Diagnosing memory leaks

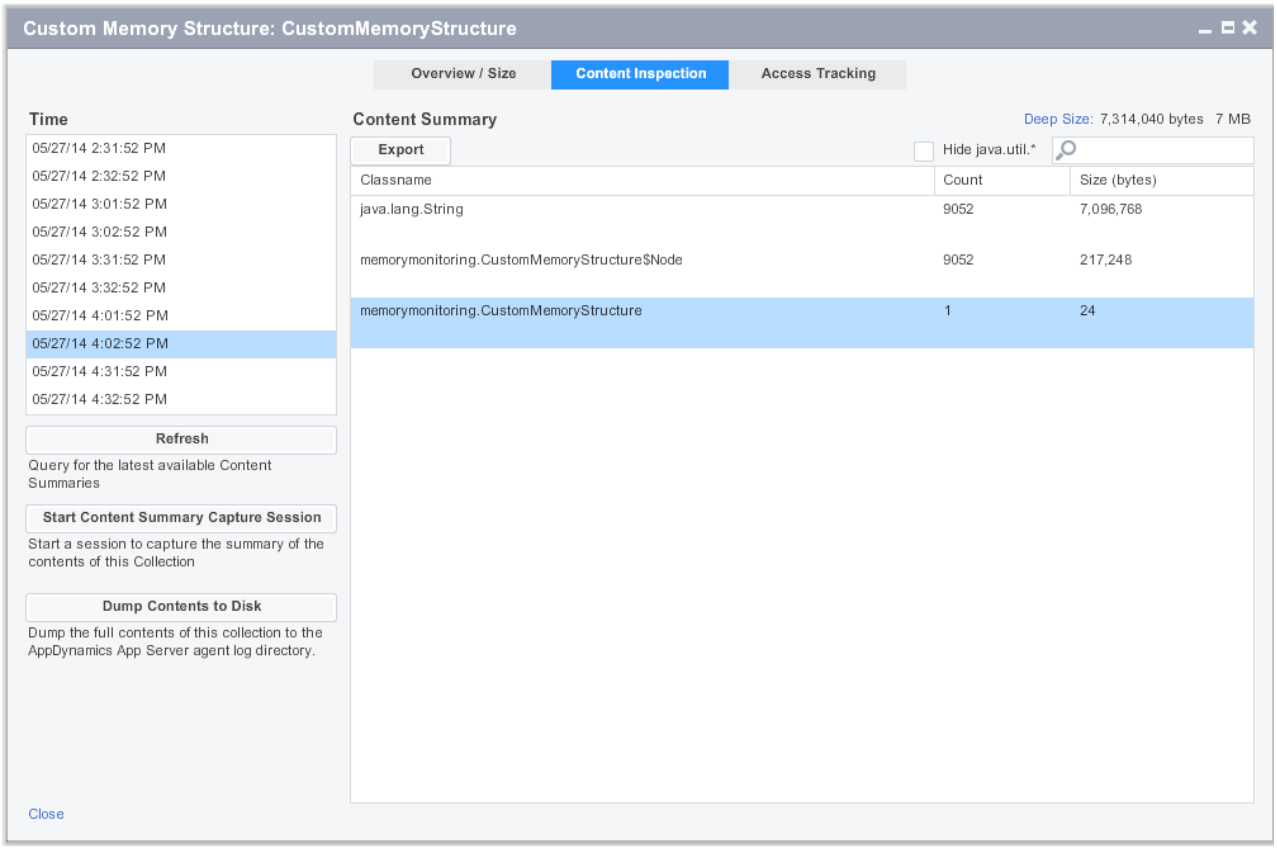
On the Custom Memory Structures Dashboard, select the class name to monitor and click **Drill Down** or right-click the class name and select **Drill Down**.




## Isolating a leaking collection

Use Content Inspection to identify to which part of the application the collection belongs. It allows monitoring histograms of all the elements in a particular memory structure. Start a diagnostic session on the object and then follow these steps:

1. Select the Content Inspection tab.
2. Click **Start Content Summary Capture Session**.
3. Enter the session duration. Allow at least 1-2 minutes for the data to generate.
4. Click **Refresh** to retrieve the session data.
5. Click a snapshot to view the details about that specific content summary capture session.

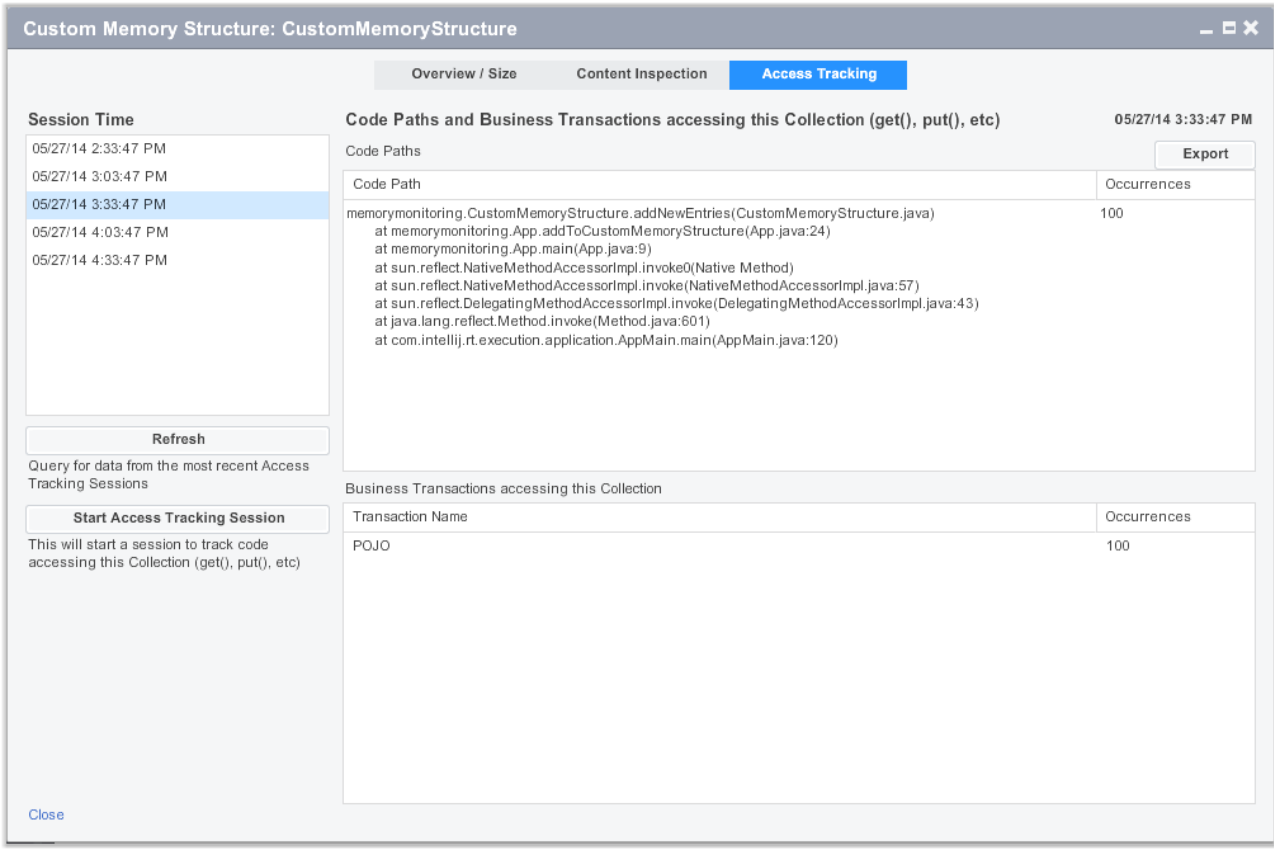



## Access Tracking

Use Access Tracking to view the actual code paths and business transactions accessing the memory structure. Start a diagnostic session on the object and follow these steps:

1. Select the Access Tracking tab.
2. Select **Start Access Tracking Session**.
3. Enter the session duration. Allow at least 1-2 minutes for data generation.
4. Click **Refresh** to retrieve the session data.

5. Click a snapshot to view the details about that specific content summary capture session.




## Learn More

- Troubleshoot Java Memory Leaks