Terracotta REST Developer Guide

Version 4.3.4

April 2017

This document applies to BigMemory Max Version 4.3.4 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: BMM-RESTDG-434-20171015

Table of Contents

Using the Terracotta Managment REST API....5
Overview of the Management Components....6
Connecting to the Management Service REST API....6
REST API Versions....7

Constructing URIs for HTTP Operations....9
The URI for the Terracotta Management Service....10
Security for REST API usage without TMC....11
Special Resource Locations....11

Specifications for HTTP Operations....15
Response Headers....16
Examples of URIs....16
DELETE....16
GET and HEAD....16
OPTIONS....22
PUT....23

Using Query Parameters in URIs....25
Using Query Parameters in URIs....26

JSON Schema....27
JSON Schema....28

REST API for the Terracotta Server Array....29
REST API for TSA....30
Statistics....30
Topology Views....30
Configuration....31
Diagnostics....31
Backups....32
Operator Events....32
Logs....33

1 Using the Terracotta Managment REST API

- Overview of the Management Components 6
- Connecting to the Management Service REST API 6
- REST API Versions 7

Overview of the Management Components

Terracotta provides the following management components:

- Management agents, embedded in BigMemory and the Terracotta Server Array (TSA), that provide a REST interface.
- The Terracotta Management Server (TMS), a process that provides a REST interface that bridges the cluster management agents.
- The Terracotta Management Console (TMC), served up by the TMS, which communicates with that TMS using its REST interface.

With the TMS management REST interface, you can also write custom scripts, or create a custom Rich Internet Application (RIA) in place of the Terracotta Management Console (TMC).

Note: For simplicity, many of the examples given in this document assume a TMS that is running locally, and therefore "localhost" is used for the host address.

Connecting to the Management Service REST API

The REST API is available by connecting to the REST management service running on the Terracotta Management Server or a node running a REST agent. Use the URLs shown below.

Connecting to a TMS

```
http://<host>:<port>/tmc/api
```

where `<port>` is 9889 if running the TMS with the default container. If using your own container, substitute the port configured for that container.

Connecting Directly to a Standalone Node

```
http://<host>:<port>/tc-management-api
```

where `<port>` is configured in the `<managementRESTService>` element's bind attribute, in the Ehcache configuration file (`ehcache.xml` by default). If you do not specify a value for this port, the default is the port of the REST management agent on the host. The default bind value is "0.0.0.0:9888".

Connecting to a TSA

```
http://<host>:<port>/tc-management-api
```

where `<port>` is the management port. This value is configured in a server's `<management-port>` element in the Terracotta configuration file (tc-config.xml by default). The default value for the management port is 9540.

REST API Versions

The REST API includes two versions, the original version and version 2 (v2). The original version was previously the only version, but with BigMemory 4.2 and higher, either original or v2 may be used.

- /agents/cacheManagers/ — original version
- /v2/agents/cacheManagers/ — version 2, available with Terracotta 4.2 and higher

With the v2 REST API, agent resources are accessible via a v2 path, and methods return responses that vary in content and/or format from the original API.

To access the v2 REST API, include v2 as part of the path, for example:

```
http://<host>:<port>/tmc/api/v2
```

or

```
http://<host>:<port>/tc-management-api/v2
```

Note: : The original REST API is still available with BigMemory 4.2 and higher, and it will continues to perform as it did with pre-4.2 BigMemory.

All nodes should use the same REST API version to avoid the risk of issues due to differing features and capabilities. You can discover the API version of connected REST agents using a GET operation with an `/agents/info` URI (see GET and HEAD). Note that the REST API version is unrelated to the version of Terracotta products or any other Terracotta API.

Differences in REST API versions can affect the features and functionality offered by the monitoring tools you create. Over time, version mismatches can arise between the TMS and TSA (when using an external TMS), and between the TMS and standalone nodes.

The TMS may be able to compensate agents with API versions older than its own version by exposing only their available capabilities. Newer agent API versions can cause inconsistent behavior or malfunction if the TMS is unable to handle unfamiliar schema, functionality, or other differences in APIs.

2 Constructing URIs for HTTP Operations

- The URI for the Terracotta Management Service 10
- Security for REST API usage without TMC 11
- Special Resource Locations 11

The URI for the Terracotta Management Service

The typical URI used to connect to the Terracotta management service has the following format:

```
<scheme>://<host>[:<port>]/<path>?<query>
```

These URIs use the standard scheme and domain, with "http" assumed as the scheme. HTTP operations access the REST API through URIs. The URI allows query strings under certain circumstances.

The URI Path

The `<path>` portion of the URI specifies resource locations using the following hierarchy:

1. **Agent IDs** – List of the desired clients using unique identifiers. If the connection is to a TMS and no IDs are given, all known clients are accessed. If the connection is made directly to a Terracotta client, then no IDs are used because these are identified by host:port addresses.

   Since a TSA REST interface can also provide access to client Rest APIs, you do need to specify an AgentID when you connect to a TSA REST interface, so that TSA rest interface can determine whether you are connecting to a client (agentId: "192.168.99.100_36364" for example) or the TSA (agentId: "embedded").

   All standalone (including TSA) REST interfaces return the agent ID "embedded".

2. **CacheManager names** – List of the CacheManagers using their configured names. If "cacheManagers" is specified in the URI but no names are given, all CacheManagers for the specified clients are accessed.

3. **Cache names** – List of the caches using their configured names. If "caches" is specified in the URI but no names are given, all caches belonging to the accessed CacheManagers are accessed. In the case where access is broad, a substantial amount of data might be returned by a GET operation.

The structure of the path takes the following form:

```
/agents[;ids={comma_sep_agent_ids}]/cacheManagers[;names={
   comma_sep_cache_manager_names}]/caches[;names={comma_sep_cache_names}
```

To connect to cache managers and caches in a cluster, use "agents/clusters" in the URI:

```
/agents/clusters/cacheManagers/caches/
```

This interface returns entities and attributes concerning the state of the clustered cache managers and/or caches.

Important: If you use this interface for access via the TMS, the interface returns information for all defined cache managers, regardless of whether they are in use (online) or not (offline). If you use the interface for access via a REST management agent, no information is returned for cache managers that are

currently offline; you cannot access offline data anywhere other than from the TMS.

You can also use this interface to delete any cache managers or caches that are no longer in use.

Security for REST API usage without TMC

When issuing HTTP requests that are not through the Terracotta Management Console, every request must include two security headers, OWASP_CSRFTOKEN and X-Requested-With.

The value of OWASP_CSRFTOKEN is dynamic, and is found in the header of each response from the TMS.

The value of X-Requested-With is static, and is always equal to `OWASP CSRFGuard Project`.

When doing a POST, DELETE, or PUT request on the TMS REST API, these two HTTP headers are required, for example:

```
DELETE http://localhost:9889/tc-management-api/v2/agents;
  id=client01/cacheManagers;names=foo/caches;names=bar/elements
  OWASP_CSRFTOKEN: M9DI-BUMD-2PPK-C45I-T6QM-ZTBE-WKKK-YT8M
  X-Requested-With: OWASP CSRFGuard Project
```

In the example above, OWASP_CSRFTOKEN: M9DI-BUMD-2PPK-C45I-T6QM-ZTBE-WKKK-YT8M is from the latest response received from the TMS.

For more information about the CSRFGuard Project, refer to the Open Web Application Security Project site.

Special Resource Locations

Certain resource locations provide specific monitoring and administration services.

Discovery

A "discovery" URI format uses the path `/agents/info`. Used with a Terracotta Management Server (TMS), this URI returns metadata on all agents known (through configuration) to that TMS. Used with an embedded web service, metadata on that agent is returned (or a 404 if that agent is not reachable). For more information about discovery URIs, refer to the examples provided in "Discover All Known Agents" on page 17.

Viewing Configuration

A URI format for viewing the configuration of CacheManagers and caches uses the path agents/cacheManagers or agents/cacheManagers/caches. Agents, cacheManagers, and caches can be specified using IDs and names. The data is returned in its native XML format.

To get the configuration of one or more CacheManagers, use the following format:

```
/agents[;ids={comma_sep_agent_ids}]/cacheManagers[;names={
comma_sep_cache_manager_names}]/configs
```

Note: If no client IDs are specified in the request, all of the clients' cacheManagers are returned. However, if no client IDs are specified in the request and the number of clients is more than the default maximum of 64, an error is returned in the JSON response. The JVM argument `com.terracotta.agent.defaultMaxClientsToDisplay` can be used to change the maximum number of clients to display.

To get the configuration of one or more caches, use the following format:

```
/agents[;ids={comma_sep_agent_ids}]/cacheManagers[;names={
comma_sep_cache_manager_names}]/caches[;names={comma_sep_cache_names}/configs
```

Setting Configuration

Cache resource locations can also be specified for setting specific cache-configuration attributes using resource representations. The following is a comprehensive list of the attributes that can be set:

- enabled – A boolean for enabling (true, DEFAULT) or disabling (false) cache operations. For example, to disable a cache's operation: PUT {"enabled":true}.
- statsEnabled – A boolean for enabling (true) or disabling (false, DEFAULT) the gathering of cache statistics.
- sampledStatsEnabled – A boolean for enabling (true) or disabling (false, DEFAULT) the sampling of cache statistics.

Probing a New Connection URI

To probe the existence of an agent at a given location, use an URL with the following format:

```
    http://127.0.0.1:9889/tmc/api/agents/probeUrl/$urlToProbe
```

For example, the following should return information about the REST agent running at the given address (localhost:4343):

```
http://127.0.0.1:9889/tmc/api/agents/probeUrl
   /http%253A%252F%252Flocalhost%253A4343
```

If the agent is available, a (status code 200 AgentMetadataEntity) response similar to the following is returned:

```
    {"agentId":"embedded","agencyOf":"Ehcache","available":true,"secured":false,
"sslEnabled":false,"needClientAuth":false,"licensed":false,"sampleHistorySize"
:8640,"sampleIntervalSeconds":10,"enabled":false,"restAPIVersion":"2.7.0"}
```

If not available, the status code should be "204 No Content".

Getting the authentication status

The authentication status is saved in the properties file `${user.home}/.tc/mgmt/settings.ini`. There is a REST resource to view the status (authentication on or off).

To GET the authentication status, use the following URI:

```
http://localhost:9889/tmc/api/config/settings/authentication
```

Either "true" or "false" is returned.

Note that to change the authentication status or settings, you must do it through the Terracotta Management Console.

3 Specifications for HTTP Operations

- Response Headers 16
- Examples of URIs 16

Response Headers

For a typical HTTP request, the response header is similar to the following:

```
   -- response --
200 OK
Content-Type:  application/vnd.sun.wadl+xml
Allow:  OPTIONS,GET,HEAD
Content-Length:  602
Server:  Jetty(7.5.4.v20111024)
```

Examples of URIs

The flexibility of the management-service REST API in turn makes available a flexible URI syntax. The examples in this section illustrate HTTP responses to specific URIs. These examples of the data returned by the listed HTTP operations are shown below without response headers.

Note: If no agent IDs are specified in a URI, all known agents are included.

DELETE

Clears a cache, or clears cache statistics.

The following DELETE examples are organized by task and URI.

Clear a Cache

```
/agents;id=client01/cacheManagers;names=foo/caches;names=bar/elements
```

Removes the elements from the cache "bar" of CacheManager "foo" on the Ehcache node "client01".

Clear Cache Statistics

```
/agents;id=client01/cacheManagers;names=foo/caches;names=bar/configs
```

Clears all cache statistics for cache "foo" and resets counters to zero.

Possible HTTP Status Codes for DELETE

400 – URI does not specify a single resource.

404 – Resource specified in the URI cannot be found.

GET and HEAD

Returns a JSON array representing the details of all specified resources, or an XML representation of data whose native format is XML.

Note: HEAD operations return the same metadata as GET operations, but no body.

The following GET examples are organized by task and URI.

Discover All Known Agents

```
/agents/info
```

Used with a TMS, this URI returns metadata on all agents known (through configuration) to that TMS. Used with an embedded web service, metadata on that agent is returned.

The following is a response from a TMS that has agents "foo" and "goo" configured, and both are responding:

```
[{"restAPIVersion":"1.0.0","available":true,"agentId":"foo","agencyOf":"Ehcache"},
{"restAPIVersion":"1.0.0","available":true,"agentId":"goo","agencyOf":"Ehcache"}]
```

The following is a response from a TMS that has agents "foo" and "goo" configured, but with only "foo" responding:

```
[{"restAPIVersion":"1.0.0","available":true,"agentId":"foo","agencyOf":"Ehcache"},
{"restAPIVersion":null,"available":false,"agentId":"goo","agencyOf":null}]
```

Note that the metadata returned includes the API version running on the agent, as well as the type of client ("agencyOf") the API is serving.

With the v2 REST API, `/v2/agents/info` returns a response with additional information, for example:

```
{"agentId":"TMS","apiVersion":"v2","entities":[{"agentId":"MyCluster",
"productVersion":"4.3.0","agencyOf":"TSA",
"available":true,"secured":true,"sslEnabled":true,"needClientAuth":false,
"licensed":true,"sampleHistorySize":670,
"sampleIntervalSeconds":4,"enabled":true},
{"agentId":"MyCluster$localhost_50808","productVersion":"2.10.0",
"agencyOf":"Ehcache","available":true,"secured":true,"sslEnabled":true,
"needClientAuth":false,"licensed":true,"sampleHistorySize":30,
"sampleIntervalSeconds":1,"enabled":true}],
"exceptionEntities":[{"agentId":"MyCluster-1",
"message":"javax.ws.rs.ProcessingException:
 java.net.SocketTimeoutException: connect timed out","stackTrace":
"java.util.concurrent.ExecutionException: javax.ws.rs.ProcessingException:
 java.net.SocketTimeoutException: connect timed out\n\tat...
```

Get Details on Specific Agents

```
/agents;ids=client01,client02
```

JSON representing an array all available agent detail. If no agent IDs are included, all agents available are returned.

Get Details on Specific Caches

```
/agents;ids=client01/cacheManagers;names=foo/caches;names=bar
```

Get Configuration of Specific CacheManager

```
/agents;ids=client01/cacheManagers;names=foo/configs
```

Returns an XML representation of the CacheManager "foo". For example, the following is an XML representation returned from a standalone Ehcache node:

```
<configurations agentId="embedded" version="1.0.0-SNAPSHOT">
  <configuration cacheManagerName="foo">
    <ehcache maxBytesLocalDisk="300M" maxBytesLocalHeap="100M"
maxBytesLocalOffHeap="200M"
    monitoring="on" name="CM1">
    <diskStore path="/var/folders/nn/lxsg77756534qfn7z14y5gtm0000gp/T/"/>
      <managementRESTService bind="0.0.0.0:9889" enabled="false"/>
      <cache name="Cache11">
    <persistence strategy="localTempSwap"/>
        <elementValueComparatorclass=
"net.sf.ehcache.store.DefaultElementValueComparator"/>
        <terracotta clustered="false">
          <nonstop/>
        </terracotta>
      </cache>
    </ehcache>
  </configuration>
</configurations>
```

Certain operations can only be executed against specific targets. Specifying multiple agents, CacheManagers, or caches generate an error response (code 400).

Get Configuration of Specific Caches

```
/agents;ids=client01/cacheManagers;names=foo/caches;names=baz/configs
```

Get All CacheManager Details

```
/agents/cacheManagers
```

The following example shows a JSON object returned by this URI when the GET is executed against a standalone Ehcache node with two CacheManagers, each with one cache:

```
    [{"name":"CM2","attributes":{"ClusterUUID":"03e505092b6a4b1a9af5d1b035a7d5ed","
Enabled":true,"HasWriteBehindWriter":false,"MaxBytesLocalDiskAsString":"300M","
CacheAverageSearchTime":0,"CachePutRate":84,"CacheOnDiskHitRate":0,
"CacheMetrics":{"Cache12":[2,84,84]},
"CacheRemoveRate":0,"CacheOffHeapHitRate":0,"Searchable"
:false,"CacheOnDiskMissRate":84,"CacheNames":["Cache12"],"
TransactionRolledBackCount":0,"CacheInMemoryHitRate":2,"WriterQueueLength":0,"
CacheOffHeapMissRate":0,"Transactional":false,"CacheHitRate":2,"
TransactionCommitRate":0,"CacheExpirationRate":0,"CacheUpdateRate":0,"
MaxBytesLocalHeap":104857600,"CacheAverageGetTime":0.027891714,"
TransactionRollbackRate":0,"CacheEvictionRate":0,"CacheInMemoryMissRate":84,"
MaxBytesLocalDisk":314572800,"MaxBytesLocalOffHeapAsString":"200M","
CacheSearchRate":0,"TransactionCommittedCount":0,"TransactionTimedOutCount":0,"
Status":"STATUS_ALIVE","MaxBytesLocalOffHeap":209715200,"WriterMaxQueueSize":0,"
StatisticsEnabled":true,"MaxBytesLocalHeapAsString":"100M","CacheMissRate":84},"
agentId":"embedded","version":"1.0.0-SNAPSHOT"},{"name":"CM1","attributes":
{"ClusterUUID":"03e505092b6a4b1a9af5d1b035a7d5ed","Enabled":true,"
HasWriteBehindWriter":false,"MaxBytesLocalDiskAsString":"300M","
CacheAverageSearchTime":0,"CachePutRate":166,"CacheOnDiskHitRate":8,"
CacheMetrics":{"Cache11":[7,83,83],"Cache12":[6,83,83]},"CacheRemoveRate":0,"
CacheOffHeapHitRate":0,"Searchable":false,"CacheOnDiskMissRate":166,"CacheNames"
:["Cache11","Cache12"],"TransactionRolledBackCount":0,"CacheInMemoryHitRate":5,"
WriterQueueLength":0,"CacheOffHeapMissRate":0,"Transactional":false,"
CacheHitRate":13,"TransactionCommitRate":0,"CacheExpirationRate":0,"
CacheUpdateRate":0,"MaxBytesLocalHeap":104857600,"CacheAverageGetTime":0.061820637,
```

```
"TransactionRollbackRate":0,"CacheEvictionRate":0,"CacheInMemoryMissRate":174,"
MaxBytesLocalDisk":314572800,"MaxBytesLocalOffHeapAsString":"200M","
CacheSearchRate":0,"TransactionCommittedCount":0,"TransactionTimedOutCount":0,"
Status":"STATUS_ALIVE","MaxBytesLocalOffHeap":209715200,"WriterMaxQueueSize":0,"
StatisticsEnabled":true,"MaxBytesLocalHeapAsString":"100M","CacheMissRate":166},
"agentId":"embedded","version":"1.0.0-SNAPSHOT"}]
```


Note: When no client IDs are specified in the request, all of the clients' cacheManagers are returned. However, if the number of clients is more than the default maximum of 64, an error is returned in the JSON response. The JVM argument `com.terracotta.agent.defaultMaxClientsToDisplay` can be used to change the maximum number of clients to display.

With the v2 REST API, `/v2/agents/cacheManagers` returns a response such as:


```
{"agentId":"TMS","apiVersion":"v2","entities":[
{"agentId":"MyCluster$localhost_50808","name":"MyCluster-1","attributes":
{"ClusterUUID":"b769bf9f44c54242a5d6eff8b1ad9dc3","Enabled":true,
"HasWriteBehindWriter":false,"MaxBytesLocalDiskAsString":"0","Searchable":true,"
MaxBytesLocalDisk":0,"CacheNames":["bigMemorySample"],
"MaxBytesLocalOffHeapAsString":"4G","Status":"STATUS_ALIVE",
"MaxBytesLocalOffHeap":4000000000,"WriterMaxQueueSize":0,
"MaxBytesLocalHeapAsString":"1G","Transactional":false,"MaxBytesLocalHeap"
:1000000000}}],"exceptionEntities":[{"agentId":"MyCluster-1","message"
:"javax.ws.rs.ProcessingException: java.net.SocketTimeoutException:
connect timed out","stackTrace":...
```


Get Offline Data

The TMS Rest API allows you to request offline cache managers and offline cache information (and even the possibility to destroy offline cache managers and caches).

The various REST endpoints available for offline data are described here:

REST endpoint:

```
GET  /agents/clusters/cacheManagers
```

Use this to list all offline and online CacheManagers from all known agents (use `;ids` on agents to filter agents, and `;names` to filter cacheManagers).

Sample output:

```
[
  {
    version: "4.3.0.0.26",
    name: "__DEFAULT__",
    agentId: "MyCluster",
    attributes: {
      inUse: "true"
    }
  }
]
```

In this example, the cacheManager named __DEFAULT__ is still in use, meaning at least one client is using it, and therefore you can't delete it.

REST endpoint:

```
GET  /agents/clusters/cacheManagers/configs
```

Use this to list cache manager configurations.

Sample output:

```
[
  {
    version: "4.3.0.0.26",
    cacheManagerName: "__DEFAULT__",
    agentId: "MyCluster",
    xml: "<ehcache name="__DEFAULT__" updateCheck="false">
            <diskStore path="java.io.tmpdir"/>
            <defaultCache maxEntriesLocalHeap="0"/>
            <terracottaConfig url="tsa:9510"/>
          </ehcache> "
  }
]
```

REST endpoint:

```
GET  /agents/clusters/cacheManagers/caches
```

Use this to list caches (you can also use `;names` on /caches to filter caches).

Sample output:

```
[
  {
    version: "4.3.0.0.26",
    agentId: "MyCluster",
    name: "vets",
    cacheManagerName: "__DEFAULT__",
    attributes: {
      inUse: "true"
    }
  }
]
```

In this example, we can see that the cache named "vets" from the Cache Manager __DEFAULT__ is in use, and thus cannot be destroyed.

REST endpoint:

```
GET  /agents/clusters/cacheManagers/caches/configs
```

Use this to list cache configurations.

Sample output:

```
[
  {
    version: "4.3.0.0.26",
    cacheName: "vets",
    cacheManagerName: "__DEFAULT__",
    agentId: "MyCluster",
      xml: "<cache name="vets" maxEntriesLocalHeap="100"
              diskExpiryThreadIntervalSeconds="1" timeToLiveSeconds="60"
              maxEntriesInCache="10000000">
              <terracotta> <nonstop enabled="false"/> </terracotta>
            </cache> "
    }
]
```

REST endpoint:

```
DELETE /agents;ids=MyCluster/clusters/cacheManagers;names=__DEFAULT__/caches;names=vets
```

Use this to delete a cache. Note the mandatory use of all the filters ids, and names; this is because you need to be very precise when deleting a cache. As for all non GET operations on the TMS, you need to include CSRF tokens in the HTTP headers :

```
OWASP_CSRFTOKEN:XXXX (you need to get this from your last GET request)
X-Requested-With:OWASP CSRFGuard Project
{
error: "Failed to destroy the cache vets"
details: "Unable to lock cache vets for destruction"
stackTrace: null
}
```

In this example, you can see that we tried to delete a cache that was still in use.

If you try again after the cache is no longer in use (inUse:false) then you will receive an empty HTTP response, with the 204 status, indicating that your cache was deleted successfully.

REST endpoint:

```
DELETE /agents;ids=MyCluster/clusters/cacheManagers;names=__DEFAULT__
```

Use this to delete a cache. Note the mandatory use of all the filters ids, and names; this is because you need to be very precise when deleting a cache manager. As for all non GET operations on the TMS, you need to include CSRF tokens in the HTTP headers :

```
OWASP_CSRFTOKEN:XXXX (you need to get it from your last GET request)
X-Requested-With:OWASP CSRFGuard Project
{
error: "Failed to destroy the cacheManager __DEFAULT__"
details: "Unable to lock entity __DEFAULT__ of type interface com.terracotta.entity.ehcache.Clust
stackTrace: null
}
```

In this example, you can see that we tried to delete a cache manager that was still in use.

If you try again after the cache manager is no longer in use (inUse:false) then you will receive an empty HTTP response, with the 204 status. Your cache manager was deleted successfully.

Get Specific CacheManager Details

```
/agents/cacheManagers?show=CacheInMemoryHitRate&show=CacheHitRate&
show=CacheAverageGetTime
```

This URI returns a JSON array with only the specified statistics:

```
[{"name":"CM1","attributes":{"CacheAverageGetTime":0.26357448,
"CacheHitRate":47,"CacheInMemoryHitRate":3},"agentId":"embedded","version":
"1.0.0-SNAPSHOT"}]
```

Configuration attributes (for example, `MaxBytesLocalHeap`) can also be specified with the `show` query parameter.

Possible HTTP Status Codes for GET or HEAD

404 – Specified resource is not found.

OPTIONS

Retrieves the WADL describing all of the operations available on the specified resources.

The following OPTIONS examples are organized by task and URI. Examples executed against standalone nodes show a base URI ending in "/tc-management-api/", while those executed against a TMS have a base URI ending in "/tmc/api/".

Return WADL With Available Agent Operations

```
/agents;ids=client01,client02
```

The following is an example of a WADL returned by an embedded agent:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/"
    jersey:generatedBy="Jersey: 1.9.1
    09/14/2011 02:05 PM"/>
<grammars/>
<resources base="http://localhost:9888/tc-management-api/">
    <resource path="agents">
        <method name="GET" id="getAgents">
            <response>
                <representation mediaType="application/json"/>
            </response>
        </method>
        <resource path="/info">
            <method name="GET" id="getAgentsMetadata">
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
        </resource>
    </resource>
</resources>
</application>
```

Return WADL With Available CacheManager "Config" Operations

```
/agents/cacheManagers/configs
```

OPTIONS using /configs:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/"
    jersey:generatedBy="Jersey: 1.9.1 09/14/2011 02:05 PM"/>
<grammars/>
<resources base="http://localhost:9888/tc-management-api/">
  <resource path="agents/cacheManagers/configs">
      <method name="GET" id="getCacheManagerConfig">
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Return a WADL With Available Operations on a Specific Cache

```
/agents/cacheManagers;names=foo,goo/caches;names=bar
```

Returns information on the cache "bar" from all CacheManagers "foo" and "goo" on any agent reachable by the TMS:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
  <doc xmlns:jersey="http://jersey.java.net/"
      jersey:generatedBy="Jersey: 1.9.1 09/14/2011 02:05 PM"/>
  <grammars/>
  <resources base="http://localhost:9889/api/">
    <resource path="agents/cacheManagers;names=foo,goo/caches;names=bar">
      <method name="GET" id="getCaches">
        <response>
          <representation mediaType="application/json"/>
        </response>
      </method>
      <method name="DELETE" id="deleteCache"/>
      <resource path="/statistics">
        <method name="DELETE" id="wipeStatistics"/>
      </resource>
    </resource>
  </resources>
</application>
```

For information from specific agents, specify the agent ID:

```
/agents;ids=client01,client02/CacheManagers=foo,bar/caches=baz,goo
```

Returns a WADL, as shown above, but with more detailed resource locations (and more caches).

Used with an embedded web service, this URI returns information for the specified caches found on that agent only.

PUT

Creates the specified resource or updates a resource representation.

Allowed Resource Updates

Updating a resource representation means editing the value of one of the following boolean cache attributes:

- Enabled – Enable (true, DEFAULT) or disable (false) the cache.
- StatisticsEnabled – Enable (true, DEFAULT) or disable (false) statistics gathering for the cache. Disabling statistics can improve a cache's performance, but limits monitoring capabilities. Note that if statistics are disabled, then sampled statistics are automatically disabled.
- SampledStatiscsEnabled – Enable (true, DEFAULT) or disable (false) sampled statistics. Sampled statistics are used for providing averages and other aggregate values. If sampled statistics are enabled, statistics gathering is automatically enabled.

Updating a Cache Attribute

The following URI can update the cache attributes as specified in the content.

```
/agents;ids=MyConnectionGroup_MyEhcache/cacheManagers;names=foo/caches;names=baz
```

For example, to turn off statistics gathering for the cache "baz", you would use the following content:

```
{"attributes":{"StatisticsEnabled":false}}
```

The attributes you can modify in this way are described in "Allowed Resource Updates" on page 23.

Possible HTTP Status Codes for PUT

201 – The operation was successful.

204 – The cache was successfully updated.

400 – The URI does not specify a single resource.

409 – The resource with the given name already exists.

4 Using Query Parameters in URIs

■ Using Query Parameters in URIs 26

Using Query Parameters in URIs

GET and HEAD HTTP operations can execute queries on specific resources. Query parameters are executed using the `show` parameter:

```
/agents[;ids={comma_sep_agent_ids}]/cacheManagers[;names=
   {comma_sep_cache_manager_names}]/caches[;names={comma_sep_cache_names}
   ?show=[parameter]&show=[parameter]
```

For example, to retrieve the values for the parameters `HasWriteBehindWriter` and `MaxBytesLocalDiskAsString` for the CacheManager CM1 on an Ehcache with the ID "foo", use the following:

```
/agents;ids=foo/cachemanagers;names=CM1?show=HasWriteBehindWriter?
    show=MaxBytesLocalDiskAsString
```

This query returns a JSON object similar to the following:

```
[{"name":  "CM1","attributes":  {"HasWriteBehindWriter":true,
"MaxBytesLocalDiskAsString":"300M"},"guid":"95d40b093c9f44389f3cc122fbe1c30b",
"agentId":"embedded","version":"1.0.0"}]
```

5 JSON Schema

JSON Schema 28

JSON Schema

Use the schema as a guide to parsing the JSON objects returned by the REST API, and to validate the structure of data your scripts or RIA sends to agents.

Note that the schema is subject to change between API versions. You can use the REST API URIs to get examples of the JSON schema for the following:

- cacheManager
- cache
- cacheConfig
- cacheStatisticsSample

6 REST API for the Terracotta Server Array

- REST API for TSA 30
- Statistics 30
- Topology Views 30
- Configuration 31
- Diagnostics 31
- Backups 32
- Operator Events 32
- Logs 33

REST API for TSA

You can use the REST API to query the Terracotta Management Server regarding any connected Terracotta Server Array.

Statistics

Use the following URI extensions with the base extension `/agents/statistics` or `/v2/agents/statistics` to return statistical information.

DGC Runs

Get statistics on the last 1000 DGC runs:

```
/dgc
```

Server Statistics

Get statistics k, l, and m for servers a, b, c:

```
/servers;names=a,b,c?show=k,l,m
```

or

```
/servers;names=a,b,c?show=k&show=l&show=m
```

If no "names" are specified, statistics for all servers are requested. If "show" is omitted, all statistics are requested.

Client Statistics

Get statistics k, l, and m for clients x, y, and z:

```
/clients;ids=x,y,z?show=k,l,m
```

or

```
/clients;ids=x,y,z?show=k&show=l&show=m
```

If no "ids" are specified, statistics for all clients are requested. If "show" is omitted, all statistics are requested.

Topology Views

Use the following URI extensions with the base extension `/agents/topologies` or `/v2/agents/topologies` to return topological information.

To get a complete cluster topology (all servers and clients), end the base extension with a forward slash ("/"):

```
/agents/topologies/
```

To get only servers a, b, and c:

```
/servers;names=a,b,c
```

If no "names" are specified, all servers are included.

To get only clients x, y, and z:

```
/clients;ids=x,y,z
```

If no "ids" are specified, all clients are included.

Configuration

Use the following URI extensions with the base extension `/agents/configurations` or `/v2/agents/configurations` to return configuration information.

To get the configuration settings for all servers and clients, end the base extension with a forward slash ("/"):

```
/agents/configurations/
```

To get only servers a, b, and c:

```
/servers;names=a,b,c
```

If no "names" are specified, all servers are included.

To get only clients x, y, and z:

```
/clients;ids=x,y,z
```

If no "ids" are specified, all servers are included.

Diagnostics

Use the following URI extensions with the base extension `/agents/diagnostics` or `/v2/agents/diagnostics` to return information useful in diagnosing trouble or initiate a DGC cycle.

Thread Dumps

Get a full thread dump from all servers and clients:

```
/threadDump
```

Get a thread dump from servers a, b, and c:

```
/threadDump/servers;names=a,b,c
```

If no "names" are specified, all servers are included.

Get a thread dump from clients x, y, and z:

```
/threadDump/clients;ids=x,y,z
```

If no "ids" are specified, all clients are included.

Thread dumps are written to the logs of their respective nodes. To have all generated thread dumps saved to a zip file, use `threadDumpArchive` instead of `threadDump`.

To write cluster state information (including, for example, on locks) in addition to thread dumps for each node, use `dumpClusterState` instead of `threadDump`. This action generates substantially more information than getting only thread dumps.

DGC Cycles

To initiate a DGC cycle, post:

```
/dgc
```

Backups

You can initiate backups of the cluster data by posting with the following URI extension:

```
/agents/backups/
```

To get the status of a backup ("true" for a backup in progress), use a GET operation with the same URI extension:

```
/agents/backups/
```

Note that backup operations involve the entire TSA and cannot be delegated to specific servers.

Operator Events

You can return operator events using the URI extension `/agents/operatorEvents` or `/v2/agents/operatorEvents`. To limit the size of the returned data, use a `sinceWhen` query.

To get operator events for the last ten minutes:

```
/agents/operatorEvents?sinceWhen=10m
```

To filter by event levels, add `eventLevels`:

```
/v2/agents/operatorEvents?eventLevels=ERROR,WARN
```

The available event levels are: DEBUG, INFO, WARN, ERROR, and CRITICAL.

To limit events to certain types, add `eventTypes`:

```
/v2/agents/operatorEvents?eventTypes=topology.node.joined,topology.node.left,
resource.capacity.near
```

For event types, see "Monitoring Cluster Events" in the *BigMemory Max Administrator Guide*.

These parameters can be combined, for example:

```
/agents/operatorEvents?sinceWhen=10m&eventLevels=ERROR,WARN
```

Logs

You can return logs using the URI extension `/agents/logs` or `/v2/agents/logs`. To limit the size of the returned data, use a `sinceWhen` query.

To get logs for the last ten minutes:

```
/agents/logs?sinceWhen=10m
```