**Topic: MUnit**

MUnit is a Mule application testing framework that enable us to easily build automated tests for integrations and APIs. It provides full suite of integration and unit test capabilities.

With MUnit we can:

* Create our test by writing Mule code/Java code
* Mock processors
* Spy any processor
* Verify processor calls
* Enable or ignore particular tests
* Tag tests
* Check visual coverage in Studio
* Generate coverage reports

MUnit is fully integrated with Anypoint Studio. You can use Studio’s graphical interface to:

* Create and design MUnit tests
* Run your tests
* View test results and coverage
* Debug your tests

**Example**

**Implement B2B using File Transfer**

**PGP – Pretty Good Privacy**

**Encrypts and compresses data before transferring data over network**

**Useful for large size of data**

**Connector**

**Object Store**

Ref: <https://dzone.com/articles/object-store-with-mule-esb>

<https://docs.mulesoft.com/object-store-connector/1.1/>

<https://docs.mulesoft.com/object-store-connector/1.1/object-store-to-store-and-retrieve>

<https://docs.mulesoft.com/object-store-connector/1.1/object-store-to-define-a-new-os>

Object Store Connector lets us access default in-memory object stores, custom object stores and persistent object stores. We can use this connector for easier access to object store and perform various operations on the object store data.

Internally, Mule uses object stores in various filters, routers, and other message processors that need to store state between messages. By default, object stores are transient, i.e. the data is stored in memory and would be lost when the application is restarted.

**Two scopes: private and Global**

Object store can be configured to be accessed only in a particular flow i.e. ‘Edit inline’ or have Global reference across the application. To enable global reference, an object store configuration is created under global configurations.

An Idempotent Message Validator Component uses object store to check if the incoming message is unique. We can either configure the is to be stored in a private object store (Edit inline) available only in flow or Global reference available anywhere in the application.

**Modes of storage: transient and persistent**

If persistent, the object store will locally persist on the disk at

 <MULE\_HOME>/.mule/<<app>>/objectstore/.

It can be accessed from the application even if it is restarted.

The different operations that can be performed using Object Store are:

* Clear
* Contain
* Store
* Retrieve
* Retrieve all
* Retrieve all keys
* Remove

**Usage:**

* Store synchronization information like Watermark column to manipulate only unique rows
* Store temporal information like access tokens
* Store user information
* Storing synchronization information, such as watermarks.
* Storing temporal information such as access tokens.
* Storing user information.

Additionally, Mule Runtime uses Object Stores to support some of its own components, for example:

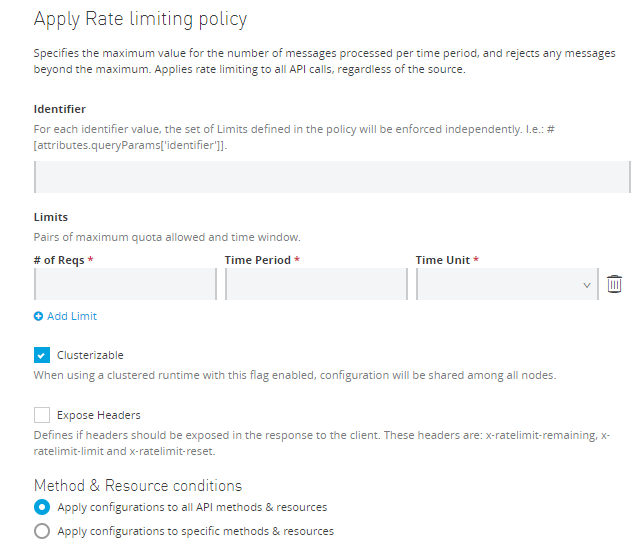
* The Cache module uses an Object Store to maintain all of the cached data.
* The OAuth module (and every OAuth enabled connector) uses Object Stores to store the access and refresh tokens.

Cache Scope and Object Store: <https://apisero.com/cache-scope-and-object-store-in-mule-4/>

**Rate Limiting Policy**

A Rate Limiting policy limits the number of requests an API accepts within a window of time. The API rejects requests that exceed the limit. You can configure multiple limits with window sizes ranging from milliseconds to years.

Generally, we configure the Rate Limiting policy from the AnyPoint Platform API Manager, but we can enforce Rate Limiting or any other Policy using API Manager API. MuleSoft provides a set of APIs that can be used to enforce policies in API Manager.



**Design Pattern**

Data is an extremely valuable business asset, but it can sometimes be difficult to access, orchestrate or interpret. When data is moving across systems, data is not always in the standard format. It is necessary as part of data integration to make the data agnostic and usable quickly across business for the data to be accessed and handled by its components. To make this process quicker and easier, certain data integration patterns are used to standardize the integration process.

Enterprise integration patterns are accepted solutions to recurring problems within a context. The patterns provide a framework to designing, building messaging and integration systems.

<https://docs.mulesoft.com/mule-runtime/4.3/understanding-enterprise-integration-patterns-using-mule>

**Aggregation Pattern**

Aggregation is the act of taking or receiving data from multiple systems and inserting into one. For example, [customer data integration](https://www.mulesoft.com/resources/esb/customer-data-integration) could reside in three different systems, and a data analyst might want to generate a report which uses data from all of them.

**Why is it useful?**

The aggregation pattern derives its value from allowing you to extract and process data from multiple systems in one [united application](https://www.mulesoft.com/resources/esb/erp-integration-application-architecture). This means that the data is up to date at the time that you need it, does not get replicated, and can be processed or merged to produce the dataset you want.

**When is it useful?**

The aggregation pattern is valuable if you are creating [orchestration](https://www.mulesoft.com/resources/esb/service-orchestration-and-soa) APIs to “modernize” [legacy systems](https://www.mulesoft.com/resources/esb/legacy-system-integration), especially when you are creating an API which gets data from multiple systems, and then processes it into one response. Another use case is for creating reports or dashboards which similarly have to pull data from multiple systems and create an experience with that data. Finally, you may have systems that you use for compliance or auditing purposes which need to have related data from multiple systems.

**Mapping to Mule Object**

Aggregators Module contains different types of aggregators capable of storing values and releasing them all together based on the configuration specified. An aggregator is a component used to collect those values until a condition is met, then execute a processor’s chain with the list of aggregated elements as the payload.

An aggregator is a pass-through router, meaning that the same message that is processed by the aggregator is going to be processed by the components that follow it. The only modifications are the addition of some message attributes that provide information about the aggregation.

Operations:

* Group-based aggregator
* Size-based aggregator
* Time-based aggregator

**Issues regarding Object Store**

1. **MaxEntries not working as expected in ObjectStore**

Symptom: Having configured a maxEntries value for the object store connector does not get honored when testing it

Causes:

* 1. *Specifying a default object store reference, such as****\_defaultInMemoryObjectStore***  
     In that case we're not creating new ObjectStore but we use the provided one  
     The problem is that customer specified a default object store reference **\_defaultInMemoryObjectStore**.  In that case we're not creating new ObjectStore but we use the provided one and we're not able to modify it.
  2. *Field* ***ExpirationInterval****is not appropriately configured.*  
     Field **ExpirationInterval** is very important here because this is the interval for checking the MaxEntries in current object store  
     MaxEntries field does not mean that no more fields are allowed to get stored, but rather it means that ObjectStore connector checks in every N milliseconds **(ExpirationInterval)** if there are more entries than **MaxEntries;** If there are more entries than the ones defined, it deletes them.
  3. ***retrieveAllKeys****is configured right after****store****operation.*  
     This then causes that connector doesn't have enough time (1000ms set in ExpirationInterval) to delete the specified object and prints out that there are 2 entries (even that it's not true because after 1000ms, OS deletes this entry and there will be only 1 entry) .

**SOLUTIONS**  
1) *Specifying a default object store reference, such as****\_defaultInMemoryObjectStore***  
If you want property MaxEntries to be honored, then an object store reference **must not** be specified, and instead ensure to configure **MaxEntries, EntryTtl and ExpirationInterval** fields.  
  
2)*Field* ***ExpirationInterval****is not appropriately configured.*  
If you'll use the **maxEntries** value, then you'll need to configure the **expirationInterval** parameter with a value that suits your use case. For example we have an object store with **ExpirationInterval set to 1000ms** and **MaxEntries: 1**, where we want to store **KEY1 : VALUE1.** We store that object, and 1000ms after storing the value object store checks and finds out that there is only 1 entry so it will let it be. We are going to store **KEY2 : VALUE2**. We store it and 1000ms after storing the second value object store finds out that there are 2 entries so it deletes the **KEY1 : VALUE1.**  
  
3) ***retrieveAllKeys****is configured right after****store****operation.*  
In order to tackle this corner case, it would be better to move **retrieveAllKeys** to a separate flow and call it independently to see keys actually included in the ObjectStore and avoid encountering false positives.

**Ref:** <https://help.mulesoft.com/s/article/MaxEntries-not-working-as-expected-in-ObjectStore>

1. **ObjectStore TTL does NOT expire**

Elements in ObjectStore are not removed after TTL is reached

SYMPTOM

The elements in an ObjectStore are not expiring after TTL has been reached.

Configuration examples

<objectstore:config doc:name="ObjectStore: Connector" entryttl="5000" expirationinterval="500" maxentries="10" name="ObjectStore\_\_Connector" objectstore-ref="\_defaultInMemoryObjectStore">

The TTL is controlled by Monitored mechanics. Although, there is a limitation that only the partitionable Object Stores are monitored. Thus, if there is no partition define, the Object Store will not be monitored.

SOLUTION

In order for the TTL to be used and Object Store which implements the Partitionable interface is required

org.mule.api.store.PartitionableObjectStore<T>

This can be achieved by adding a partition in the object store connector,

<objectstore:config name="ObjectStore\_\_Connector" entryTtl="10000" expirationInterval="5000" maxEntries="10" doc:name="ObjectStore: Connector" partition="test”/>

Ref: <https://help.mulesoft.com/s/article/ObjectStore-TTL-does-NOT-expire>