MIRROR Persistence Service

Version 0.3 - January 6, 2014

XMPP Integration

The MIRROR Persistence Service is registered as component of the XMPP domain and therefore acts on the same level as the MIRROR Spaces Service. Whilst the Spaces Service is mandatory within the MIRROR Spaces Framework (MSF), the Persistence Service is optional.
In its current implementation, the service is realized as plug-in for the Openfire XMPP server.

Setting Up the Persistence Service

In order to use the Persistence Service, a new version of the MIRROR Spaces Plugin (version 0.5+) is required. Both plugins can be deployed by using the Openfire administration console or by copying the *.jar files in the `plugins`-directory of your installation.

Note: Existing spaces and data objects stored within the related pubsub nodes are preserved. It is therefore save to upgrade existing versions.

The Persistence Service has be registered at the Spaces Service in order to store data objects. To do so, activate the following entry on the administration console:

MIRROR Spaces ⇒ Settings ⇒ General Settings
⇒ Connect to the MIRROR Persistence Service

Due to some changes to the space configuration (see below), an update of the MIRROR Spaces SDKs used for application development is required. Compatible implementations of the MIRROR Spaces SDK are:

- Spaces SDK for Java 1.2+
- Spaces SDK for Android 1.2+
- Spaces SDK for JavaScript 1.2+

Note: New versions of the MIRROR Spaces SDK works also for old version of the MIRROR Spaces Service. An application using the latest SDK can therefore connect to all MSF domains.

Configure a Space to Be Persisted

More details on XMPP communication in general and the space configuration in particular can be found in the documentation of the MIRROR Spaces Service.

The space configuration has been slightly modified to support the persistence service. The `spaces#persistent` field now has three valid options:

1. `false` to disable persistence of data objects published on the pubsub node of the space.
2. `true` to enable persistence without any limitations to the lifetime of published data objects. Data objects can still be deleted manually.
3. A XSD duration string.[1] The given duration specifies the lifetime of all *subsequently* published data objects. Expired data objects will be purged automatically.

```
[…]
<configure>
  <x xmlns="jabber:x:data" type="submit">
    <field var="FORM_TYPE" type="hidden">
      <value>
        urn:xmpp:spaces:config
      </value>
    </field>
    <field var="spaces#type">
      <value>team</value>
    </field>
    <field var="spaces#persistent">
      <value>P1D</value>
    </field>
    <field var="spaces#name">
      <value>TOP team</value>
    </field>
    <field var="spaces#members">
      <value>john.doe@mydomain</value>
      <value>jane.doe@mydomain</value>
    </field>
    <field var="spaces#moderators">
      <value>john.doe@mydomain</value>
    </field>
  </x>
</configure>
[…]
```

Note: In the current implementation, only instances of MIRROR data models are persisted, i.e., data objects with a namespace `mirror:*`.

Accessing the Persistence Service

If a space is configured to be persisted, an additional space channel is available representing Persistence Service XMPP component:

[1] Further examples for xsd:duration strings can be found here.

```
[…]
<spaces xmlns="urn:xmpp:spaces">
  <channels space="alice">
    […]
    <channel type="persistence">
      <property key="jid">persistence.mirror-demo.eu</property>
    </channel>
    […]
  </channels>
</spaces>
[…]
```

Querying Data Objects

Queries are performed by sending an IQ stanza of type `GET` to the Persistence Service component. The IQ contains a single child named `query` with the namespace `urn:xmpp:spaces:persistence`. Each query has the following structure:

```
<query xmlns="urn:xmpp:spaces:persistence">
  <!-- space or id information -->
  <!-- filters (optional) -->
</query>
```

As response either an IQ of type `RESULT` is sent to the client, containing a list of data objects (which may be empty), or IQ of type `ERROR`.

```
<query xmlns="urn:xmpp:spaces:persistence">
  <result>
    <ping xmlns="mirror:application:ping:ping"
publisher="alice@localhost/persistenceTest" modelVersion="1.0" cdmVersion="1.0"
id="41692734-a6a2-46c2-89eb-ea3644272a16" timestamp="2013-04-08T16:06:58.557+02:00"
schemaLocation="mirror:application:ping:ping
http://data.mirror-demo.eu/application/ping/ping-1.0.xsd">
      <content>Sample content.</content>
    </ping>
    <ping xmlns="mirror:application:ping:ping"
publisher="alice@localhost/persistenceTest" modelVersion="1.0" cdmVersion="1.0"
id="d49996ba-338d-4695-adcb-65a068f4555c" timestamp="2013-04-08T16:02:40.730+02:00"
schemaLocation="mirror:application:ping:ping
http://data.mirror-demo.eu/application/ping/ping-1.0.xsd" />
  </result>
</query>
```

Data objects can be queried either by space or by data object identifiers. For both options commands are available to address multiple entities.

Query by ID

If the ID for a data object is known, e.g. as reference, object can be retrieved using the following query:

```
<query xmlns="urn:xmpp:spaces:persistence">
  <object id="41692734-a6a2-46c2-89eb-ea3644272a16" />
</query>
```

To request multiple objects in a single query, you can place multiple `object` elements within an `objects` tag:

```
<query xmlns="urn:xmpp:spaces:persistence">
  <objects>
    <object id="41692734-a6a2-46c2-89eb-ea3644272a16" />
    <object id="d49996ba-338d-4695-adcb-65a068f4555c" />
  </objects>
</query>
```

Queries by ID are performed over all spaces persisted by the service. The MIRROR Spaces Access Model (MSAM) is applied to all queries. If the requesting user is not allowed to access one or more of the requested data objects, i.e., one of the spaces the data objects are published on, the respective error is returned.

Query by Space

It is also possible to retrieve all data objects stored for a space or even multiple spaces.

```
<query xmlns="urn:xmpp:spaces:persistence">
  <objectsForSpace id="team#123" />
</query>
```

```
<query xmlns="urn:xmpp:spaces:persistence">
  <objectsForSpaces>
    <space id="orga#5" />
    <space id="team#123" />
  </objectsForSpaces>
</query>
```

Filter Queries

Filters can be applied to all queries to restrictthe result to data objects with specific properties. They are specified within the query in a filters tag right after the entity selection, e.g.:

```
<query xmlns="urn:xmpp:spaces:persistence">
  <objectsForSpaces>
    <space id="orga#5" />
    <space id="team#123" />
  </objectsForSpaces>
  <filters>
    <period from="2013-04-09T10:30:00+01:00" to="2013-04-09T11:00:00+01:00" />
  </filters>
</query>
```

It is possible to combine different filters for further restriction. The following filters are available:

Period of Publishing

Description	Restricts the period in time the data object was published.
Attributes	• `from`

	Specifies the earliest point in time the object was published. Optional. • `to` *Specifies the latest point in time the object was published. Optional*
Example	`<period from="2013-04-09T10:30:00+01:00" to="2013-04-09T11:00:00+01:00" />`

Publisher

Description	Only data objects from the given publisher (bare-JID or full-JID) are returne Removes all non-personalized data objects.

Example	`<publisher>`
	`  alice@mirror-demo.eu`
	`</publisher>`

Namespace

Description	Restricts the query to specific namespaces, e.g. "mirror:application:moodmap:mood".
Attributes	• `compareType` *Defines the type of comparision. Either "strict" (default), "contains", o "regex". The choice influences also the performance of the filter. Optional.*

Example	`<namespace compareType="contains">`
	`  mirror:application:moodmap`
	`</namespace>`

Data Model

Description	Filter for data model information.
Attributes	• `namespace` *Namespace of the data model the data object have to implement. Checke for equality.* • `version` *Version of the data model. Optional.*
Example	`<dataModel namespace="mirror:application:ping:ping" version="1.0" />`

Object Reference

Description	Request only data objects which refer to a specific object.
Attributes	• `id` *Data object identifier of the referenced object.*
Example	`<references id="d49996ba-338d-4695-adcb-65a068f4555c" />`

Deleting Data Objects

Data objects can be deleted manually by sending an IQ SET request to the Persistence Service component:

```
<delete xmlns="urn:xmpp:spaces:persistence">
  <object id="d49996ba-338d-4695-adcb-65a068f4555c" />
</delete>
```

```
<delete xmlns="urn:xmpp:spaces:persistence">
  <objects>
    <object id="41692734-a6a2-46c2-89eb-ea3644272a16" />
    <object id="d49996ba-338d-4695-adcb-65a068f4555c" />
  </objects>
</delete>
```

As the lifetime of a data object is usually handled by the service itself, this operation is only permitted for space moderators. It is possible to break object dependencies by deleting objects manually.