SAP Edge Services
Persistence Service – Guide
Version 1709
# Table of Contents

ABOUT THIS DOCUMENT ........................................................................................................................................ 3

PERSISTENCE SERVICE (DATA STORAGE) ........................................................................................................... 4
Local Generic SQL Database ................................................................................................................................. 4
Automated Edge Storage of IoT Services Data ..................................................................................................... 4
Persistence Service API ......................................................................................................................................... 4
Bundle Registration ............................................................................................................................................... 5
Bundle Permissions ............................................................................................................................................... 6
Persistence Service System Schema ..................................................................................................................... 7
Bundle Configuration ........................................................................................................................................... 8
Executing Dynamic SQL ...................................................................................................................................... 8
Using ExecuteSQL API ......................................................................................................................................... 9
PSDataObject .......................................................................................................................................................... 9
PSSStatementObject extends PSDataObject ......................................................................................................... 10
Query Execution with Result Sets ......................................................................................................................... 10
Accessing Device and Measure System Data ..................................................................................................... 11
Using the Persistence Service API ....................................................................................................................... 11
Using SQL ............................................................................................................................................................ 12
ABOUT THIS DOCUMENT
This overview guide provides a starting point for understanding and using SAP Edge Services. It describes the primary concepts and key functionality, as well as references to all other relevant documentation.

This document is intended for the following audiences:
- Consultants
- Partners
- Customers
- Developers
PERSISTENCE SERVICE (DATA STORAGE)

The Persistence Service provides two broad sets of functionality. These features are intended to be leveraged by other custom OSGi bundles deployed to the IoT Services gateway.

1. Local Generic SQL Database. A local SQL database implementation that can be leveraged by custom bundles as a secure, generic persistence store.
2. Automated Edge Storage of IoT Services Data. IoT Services data (measures and device data) that originated on the gateway are automatically stored on the gateway, in the Persistence Service.

Local Generic SQL Database

The Persistence Service contains several methods for interaction, including registering a custom bundle and executing SQL.

Automated Edge Storage of IoT Services Data

By default, any inbound device and measure data is automatically persisted in the Persistence Service database. There are several configuration settings for the length of time these objects are stored. By default, each individual type of measure data is persisted for a maximum of 30 days or a maximum of 10,000 measures. The Persistence Service provides an API to retrieve this data from the database. The data is also made available (read only) to SQL inside the database. For example, a bundle can write a SQL statement that can join to these tables.

Annotations can be used to simplify the service reference registration by adding a @Reference annotation to the binding (or setting) method, along with any desired attributes. The annotation automatically creates the necessary Component Definition file, that will expect the unbinding method to have ‘un’ prepended to the method name. The service to bind is derived from the class/interface passed into the method. For example, the sample below binds the IPersistenceService interface. The IPersistenceService reference parameter contains the methods for bundle registration and query execution.

```java
@Reference(cardinality = ReferenceCardinality.MANDATORY)
public void setPersistenceService(IPersistenceService serviceRef) {
    IPersistenceService persistenceService = serviceRef;
    ...
}

public void unsetPersistenceService(IPersistenceService serviceRef) { ... }
```

Persistence Service API

There is a single Persistence Service API that allows access to both the Local generic SQL database and read-only access to the Automated Edge Storage of IoT Services data.

The Persistence Service provides the following APIs:

- RegisterBundleForAccess(String canonicalName, char[] password)
  - Any bundle can use the Persistence Service APIs. Each bundle must first register with the Persistence Service, providing their bundle’s canonical name and a password to use when accessing the service. The password must be a minimum of 8 characters.
- GetDeviceTopologyByMac(String macAddress)
  - Returns device objects for all devices the Persistence Service has captured while running. These can be optionally filtered by providing a macAddress as a parameter.
- GetMeasuresByTypeID(Integer profileId, Integer objectId)
o Returns measure objects for all measures that the Persistence Service has captured while running. These objects can be optionally filtered by providing a profileId (there can be multiple profiles in the gateway) and optionally objectId (identifies a measure). A combination of profileId and objectId uniquely identify a measure.

- ExecuteSQL(String token, String sql)
  o A token is essentially a session ID object (see below). Any arbitrary SQL can be sent to the database. The bundle has full read / write access to their own schema. Each bundle also has read only access to device and measures.

**Bundle Registration**

Each time the gateway is started, before interacting with the Persistence Service reference, your bundle must first be registered for access through the RegisterBundleForAccess() method. The registration process returns a Persistence token (consider it a session) that can be used until the Persistence Service is restarted. When registering with the Persistence Service, you must provide your bundle name and a password (minimum of 8 characters).

```java
//Register a bundle for persistence access
String canonicalName = bundle.getSymbolicName();
char[] passwd = "S@mp1ePa$$wd".toCharArray();
String persistenceToken = persistenceService.RegisterBundleForAccess( yourBundleCanonicalName, passwd );
```

If the Persistence Service is restarted, the same information must be provided when calling the RegisterBundleForAccess() method. If a different password is used, an authentication error is reported and no Persistence token is returned. **Any bundle registering with the Persistence Service must ensure they provide the same password each time they register.**

Here is a flow diagram depicting the access route to the Persistence Bundle:
Requesting & Using a Bundle Schema

Bundle Permissions

The Persistence Service is a shared resource. It provides services and data to other bundles and provides a database schema to be used by custom bundles.

- All the system objects provided by the Persistence Service provide read only access to all registered bundles.
- Each registered bundle has their own database schema and full write access to that schema. They can create/alter/drop objects and manipulate the data within that schema.
- While registered bundles have full access to their own schema and read only access to the system schema, bundles have no access to other bundle’s schema.

The following diagram summarizes permissions:
Persistence Service System Schema

The Persistence Service uses the following database schema

- **SYSTEM_PROPERTY**
  - Properties used to manage itself.

- **CONFIG_PROPERTY**
  - Property names used to configure measures.

- **CONFIG_MEASURE**
  - For each measure in the system, for each property, store a value (i.e. shelf life, maximum stored).

All the Persistence Service objects are under the **EFPS** schema name.
- **BUNDLE_REGISTER**
  o Indicates which bundles have registered with the Persistence Service.

- **DEVICE**
  o Store some properties provided by the gateway when a new device is encountered.

- **DEVICE_MEASURE_TYPE**
  o Links a device to a specific measure.

- **MEASURE_TYPE**
  o Stores the name and unique identifier of each measure.

- **MEASURE**
  o Store some properties provided by the gateway when a new measure is encountered.

### Bundle Configuration

Measures can be configured via the IoT Services Cockpit by defining IoT Gateway Properties using specific patterns. This allows an administrator to configure any gateway remotely.

There are three different formats for the properties the Persistence Service is expecting. Size is a number representing the number of rows.

1. **An override for all profiles, and all measures**
   - Change the **SIZE** property for:
     a. All profiles
     b. All measures
   - Format
     \[ PS\_MEASURE\_\{(SIZE | AGE)\}\_ALL = n \]
   - Equivalent to
     \[
     \begin{align*}
     PS\_MEASURE\_SIZE\_ALL & = 10000 \\
     PS\_MEASURE\_AGE\_ALL & = 30
     \end{align*}
     \]

2. **An override for all profiles, but a specific measure**
   - Change the **SIZE** property for:
     a. All profiles
     b. A specific measure
   - Format
     \[ PS\_MEASURE\_\{(SIZE | AGE)\}\_ALL\_{OBJECT\_ID} = n \]
   - Equivalent to
     \[
     \begin{align*}
     PS\_MEASURE\_SIZE\_ALL\_5 & = 10000 \\
     PS\_MEASURE\_AGE\_ALL\_5 & = 30
     \end{align*}
     \]

3. **A specific measure**
   - Requires specifying a ProfileId and ObjectId (uniquely identifies a measure)
   - Format
     \[ PS\_MEASURE\_\{(SIZE | AGE)\}_{PROFILE\_ID}_{OBJECT\_ID} = n \]
   - Equivalent to:
     \[
     \begin{align*}
     PS\_MEASURE\_SIZE\_0\_2 & = 10000 \\
     PS\_MEASURE\_AGE\_3\_15 & = 30
     \end{align*}
     \]

### Executing Dynamic SQL

The Persistence Service provides a dynamic SQL API that allows a custom bundle to execute SQL within the bundle’s own schema.
**Using ExecuteSQL API**

SQL commands are executed with the `ExecuteSQL()` service method, which requires the previously generated Persistence token (for bundle authentication) and the SQL command. The method returns a `PSStatementObject` object with several exposed methods that can be utilized to determine the results of the statement execution.

There is no state information maintained in the Persistence Service, therefore there is no need to open or close statements. The `PSStatementObject` object contains all the required information.

Statements can either return a result set (i.e. Select statement) or return a row count (i.e. Insert, Update, Delete statement). You can distinguish between the two types of statements by checking the `hasResultList()` method.

```java
// Delete data from the table
IPersistenceService persistenceService;
PSStatementObject statement;
String sql = "DELETE FROM sampleTable WHERE id='1'";
statement = persistenceService.ExecuteSQL(persistenceToken, sql);

if (statement.hasSuccess()) {
    System.out.println("DELETE statement succeeded, rows deleted:" +
    statement.getRowsAffected());
} else {
    System.out.println("DELETE statement failed, SQLCode:" +
    statement.getSQLCode() + " SQLState:" + statement.getSQLState() + " Error:" +
    statement.getMessage());
}
```

**PSDataObject**

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasSuccess()</td>
<td>Boolean</td>
<td>Indicates that the statement executed successfully.</td>
</tr>
<tr>
<td>hasWarnings()</td>
<td>Boolean</td>
<td>Indicates that the statement executed with a warning.</td>
</tr>
<tr>
<td>hasErrors()</td>
<td>Boolean</td>
<td>Indicates that the statement executed with an error.</td>
</tr>
<tr>
<td>getSQLCode()</td>
<td>Integer</td>
<td>Returns the statement execution SQL code.</td>
</tr>
<tr>
<td>getSQLState()</td>
<td>String</td>
<td>Returns the statement execution SQL state.</td>
</tr>
<tr>
<td>getMessage()</td>
<td>String</td>
<td>Returns the statement execution database message.</td>
</tr>
<tr>
<td>getMetadata()</td>
<td>String</td>
<td>Returns the data type of the column.</td>
</tr>
<tr>
<td>getColumnName()</td>
<td>String</td>
<td>Returns the column name.</td>
</tr>
<tr>
<td>getValue()</td>
<td>String</td>
<td>Returns the column value.</td>
</tr>
</tbody>
</table>
**PSSStatementObject extends PSDataObject**

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasSuccess()</td>
<td>Boolean</td>
<td>Indicates that the statement executed successfully.</td>
</tr>
<tr>
<td>hasErrors()</td>
<td>Boolean</td>
<td>Indicates that the statement executed with an error.</td>
</tr>
<tr>
<td>hasResultList()</td>
<td>Boolean</td>
<td>Indicates that the statement returns a result set.</td>
</tr>
<tr>
<td>getRowsAffected()</td>
<td>Integer</td>
<td>Returns the affected row count.</td>
</tr>
<tr>
<td>getResultList()</td>
<td>List&lt;List&lt;PSDataObject&gt;&gt;</td>
<td>Returns the result set from statement execution.</td>
</tr>
<tr>
<td>getSQLCode()</td>
<td>Integer</td>
<td>Returns the statement execution SQL code.</td>
</tr>
<tr>
<td>getSQLState()</td>
<td>String</td>
<td>Returns the statement execution SQL state.</td>
</tr>
<tr>
<td>getMessage()</td>
<td>String</td>
<td>Returns the statement execution database message.</td>
</tr>
</tbody>
</table>

**Query Execution with Result Sets**

SQL commands can also return result sets. The method returns a PSSStatementObject object with several exposed methods that can be utilized to iterate through the result set returned from the statement.

```java
IPersistenceService persistenceService;
PSStatementObject statement;

// Create a table with
// an INTEGER (generated value),
// a VARCHAR (supplied value),
// a DATETIME (a defaulted value)
String sql = "CREATE TABLE IF NOT EXISTS test1(
  id INT DEFAULT AUTOINCREMENT,
  name VARCHAR(100),
  date_added DATETIME DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY(is) )";
statement = persistenceService.ExecuteSQL(persistenceToken, sql);

// Error checking removed for brevity (see previous example)

// Add a new row "Test1"
String sql = " insert into test1(name) values('Test1')"
statement = persistenceService.ExecuteSQL(persistenceToken, sql);

// Add a new row "Test2"
String sql = " insert into test1(name) values('Test2')"
statement = persistenceService.ExecuteSQL(persistenceToken, sql);

// Query the new data added
```
statement = service.ExecuteSQL(token, "SELECT TOP 10 * FROM test1");

if (statement.hasResultList()) {
    System.out.println(" QUERY RESULTS");
    System.out.println(" ------------");
    Iterator<List<PSDataObject>> rows = statement.getResultList().iterator();
    while(rows.hasNext()){
        Iterator<PSDataObject> columns = rows.next().iterator();
        StringBuilder rowText = new StringBuilder();
        while(columns.hasNext()){
            PSDataObject column = columns.next();
            rowText.append("        " + column.getValue());
            rowText.append("(" + column.getColumnName() + ":" +
            column.getMetadata() + ")  ");
        }
        System.out.println(rowText);
    }
}

QUERY RESULTS
-------------
1(id:int)   Test1(name:varchar)   2017-05-31
15:39:53.502000(date_added:datetime)
2(id:int)   Test2(name:varchar)   2017-05-31
15:39:53.502000(date_added:datetime)

Accessing Device and Measure System Data

A bundle can retrieve the device and measure data from the Persistence Service using the built in Persistence Service APIs or directly within the database using SQL statements.

Using the Persistence Service API

A bundle can retrieve the device and measure data from the Persistence Service using the GetDeviceTopologyByMac() and GetMeasuresByTypeID() APIs. These APIs return a PSStatementObject (refer to the examples in Using ExecuteSQL API and Query Execution with Result Sets) that allows the bundle to iterate through the data using Java.

System.out.println("Retrieving All Measures");
List<Measure> measureLst = service.GetMeasuresByTypeID(null, null);

System.out.println(" ID                                   P O      Values
DeviceID");
measureLst.forEach ((temp) -> {
    System.out.println(" " + temp.getMeasureID() +
    " " + temp.getProfileId() +
    " " + temp.getObjectId() +
    " " + temp.getValuesForMeasure() +
    " " + temp.getProfileName() +
    " " + temp.getObjectTag() +
    " " + temp.getProfileTag() +
    " " + temp.getObjectValue() +
    " " + temp.getProfileValue() +
    " " + temp.getObjectValue());
});
" " + temp.getDeviceAddress();
});

Using SQL

The ExecuteSQL() API call also provides a means to access this data. Using the Entity Relationship Diagram (ER Diagram) shown earlier, you can write SQL statements to access the data. The schema/owner name EFPS must be used when referencing these tables.

// Query the Measure data
statement = service.ExecuteSQL(token, "SELECT TOP 10 * FROM EFPS.MEASURE ORDER BY DATE_RECEIVED DESC");

// Query the Measure data looking up the identifier by name
statement = service.ExecuteSQL(token, "SELECT TOP 10 m.* FROM EFPS.MEASURE m JOIN EFPS.MEASURE_TYPE mt ON (m.PROFILE_ID = mt.PROFILE_ID AND m.OBJECT_ID = mt.OBJECT_ID) WHERE mt.MEASURE_NAME = 'Temperature' ORDER BY m.DATE_RECEIVED DESC");

This allows a custom bundle to access saved data and run analytics across the data set.