INTERNAL USE ONLY

SAP Lumira
Document Version: 1.25 – 2015-04-14

Data Access Extensions for SAP Lumira Developer Guide

Content

1 Document History

The following table provides an overview of the most important document changes.

Version	Date	Description
1.24	March 2015	V2 architecture, JAVA only
1.19	September 2014	Added C# example data extension
1.17	June 2014	First release

2 Data Access Extension SDK for SAP Lumira

2.1 Introduction to Data Access Extensions

This SDK will help Java and JavaScript developers to create Data Access Extensions (DAE) to add to SAP Lumira using the Extension Manager.

Data Access Extensions extend the ability of SAP Lumira Desktop to import additional data sources such as XML files, or other unusually formatted static or online data sources, for example:

- event log files
- directory listings
- online social data sources such as Twitter
- Google Analytics
- Facebook
- Big data sources such as Google Big Query

Once added, the new datasource will be available to users when they create new documents.

Supported Workflows

Using data source extensions, users can perform the standard SAP Lumira data acquisition workflows that include the following:

- Preview - see what sample data looks like before it is imported.

  > Note
  >
  > The V1 SDK provides data preview functionality. The current V2 SDK gives developers the flexibility of creating a customized "Preview" user interface to allow full control of how their data is presented.

- Edit - modify what information to import, such as sales columns
- Refresh - refresh SAP Lumira with updated data

Developers can design their own user interface, including a browse dialog box that allows users to select a file or remote datasource, preview the data, and set import options.

Included in the documentation is an example Eclipse IDE project for a data access extension called `sampleextension`, which is a simple plugin to import CSV data into SAP Lumira. Iincluded is all the Java and JavaScript source code and auxiliary files required to build a SAP Lumira compatible plugin. For the third party JSON library *.java files used in this project, a download link is provided.

2.1.1 Data Access Extension SDK V2

As of SAP Lumira 1.24, the Data Extension Framework has been updated with a new architecture to improve the integration and usability of developer-created Data Access Extensions, and to improve end users' experience with SAP Lumira. New features include:

- Faster acquisition compared to the V1 Data Access Extension SDK
- JavaScript and HTML5 support
- Metadata provision for detailed column definitions
- A new transport unit (ZIP archive) packaging and deployment format
- Localization
- Customizable icons and descriptions for extensions
- Extension revision enforcement
- Support of the new Extension Manager
- Future compatibility with SAP Lumira server products, such as SAP Lumira Edge, and SAP Business Intelligence Platform

Developers who wish to take advantage of the new features must create extensions written using Java and JavaScript. The SAP Lumira 1.17 plugin format is still available; it is an executable format rather than the ZIP TransportUnit format that 1.24 uses.

2.1.2 Data Access Extension SDK V1

The SAP Lumira 1.17 style plugins can be programmed using any language, but they must be compiled as an application. Data extension applications are copied to the `SAP Lumira > Desktop > daextensions` folder, and must be enabled by editing the `SAPLumira.ini` configuration file.

For more information on using the earlier SDK, download the 1.22 version of the Data Access Extensions for SAP Lumira Developer Guide as follows:

1. Go to http://help.sap.com/boall_en/
2. Under Analytics Knowledge Centre, in the "all products" menu, select "Lumira".
3. In the "all releases" menu, select "SAP Lumira".
4. Under the "Available Titles" list, look for the Data Access Extensions for SAP Lumira Developer Guide 1.22, and select the download link.

For new projects, SAP strongly recommends using the latest SDK.

3 About This Guide

3.1 Who Should Use this Guide

This guide is a software development kit (SDK) intended for experienced application programmers who wish to extend and customize SAP Lumira Desktop's data access source capability. The guide includes information that may be of use to IT administrators and data analysts:

- Java Developers - For Java and JavaScript programmers who need to extend the data access capability of SAP Lumira.
- IT Administrators - For administrators who need to understand how to configure, install, remove, and troubleshoot data access extensions.
- Data Analysts - For analysts who identify the need for a new extension. Individuals that need to do a requirements analysis may also design the high level user interface, so this guide will provide information on the internal architecture of the Data Access Extension, and show how data is delivered to SAP Lumira.

3.2 What this Guide Contains

This guide contains the following information:

- An overview of data access extensions
- Information on how to develop, test, and deploy data access extensions
- Information on how to upgrade data access extensions
- Information on how to remove data access extensions from SAP Lumira
- An example Java project
- API Reference
- Troubleshooting information
- Limitations to consider while working with data access extensions
- Frequently asked questions (FAQs)

4 Getting Started

4.1 Introduction

This Getting Started section will provide an overview of the development requirements needed to use the Data Access Extension SDK.

As of version SAP Lumira 1.24, the Java-based data access extension SDK is included. The data acquisition extension framework allows developers to create easy to install plugin modules to extend the data import capability of SAP Lumira.

This section provides list of development requirements and a sample extension that quickly demonstrates the end-to-end process of compiling and installing a data access extension.

4.2 Hardware and Software Requirements

This section lists the hardware and software requirements for the SAP Lumira Data Access Extension SDK.

The hardware and software requirements that must be met before developing extensions is as follows:

Hardware Requirements

- Windows computer 32-bit or 64-bit, with at least 4GB RAM
- 3.5GB hard drive space for the SAP Lumira installation and data

SAP Software

- SAP Lumira 1.24 32-bit or 64-bit

Third-party Software

- Eclipse Luna Java EE IDE for Web Developers
- Java Development Kit 7, with Update 75 32-bit or 64-bit

Languages Supported

- English

4.3 Software Dependencies

List of additional software components that need to be included with the data extension SDK.

The data access extension SDK library JAR file is included, starting with SAP Lumira 1.24, `com.sap.bi.da.extension.sdk`. Typically, this library is located at:

```
<installation directory>SAP Lumira\Desktop\plugins
\com.sap.bi.da.extension.sdk_1.24...jar
```

- `com.sap.bi.da.extension.sdk`
  - DAEWorkflow
  - DAException
  - IDAEAcquisitionJobContext
  - IDAEAcquisitionState
  - IDAEClientRequestJob
  - IDAEDataAcquisitionJob
  - IDAEEnvironment
  - IDAEMetadataAcquisitionJob
  - IDAEProgress
  - IDAExtension
  - IDAELogger

As mentioned in the Development Considerations section that follows, any additional third-party components that are required for the example extension to run must be included within the build, not externally referenced.

For the **sampleextension** example project used in this guide, the JSON *.java library files may be downloaded from:

- `https://github.com/douglascrockford/JSON-java` (this includes the source code and binaries
- `https://raw.githubusercontent.com/douglascrockford/JSON-java/master/JSONWriter.java` is an example a direct link to one file.
- `http://www.json.org/java/` is the home of the JSON format

The `org.json.java` files included with the **sampleextension** project in this guide are as follows:

- `org.json.JSONObject`
  - JSONArray.java
  - JSONException.java
  - JSONObject.java
  - JSONString.java
  - JSONStringer.java
  - JSONTokener.java
  - JSONWriter.java
- Java collections used in the `sampleextension`:
  - java.io.File
  - java.nio.file.Files
  - java.util.EnumSet
  - java.util.Set

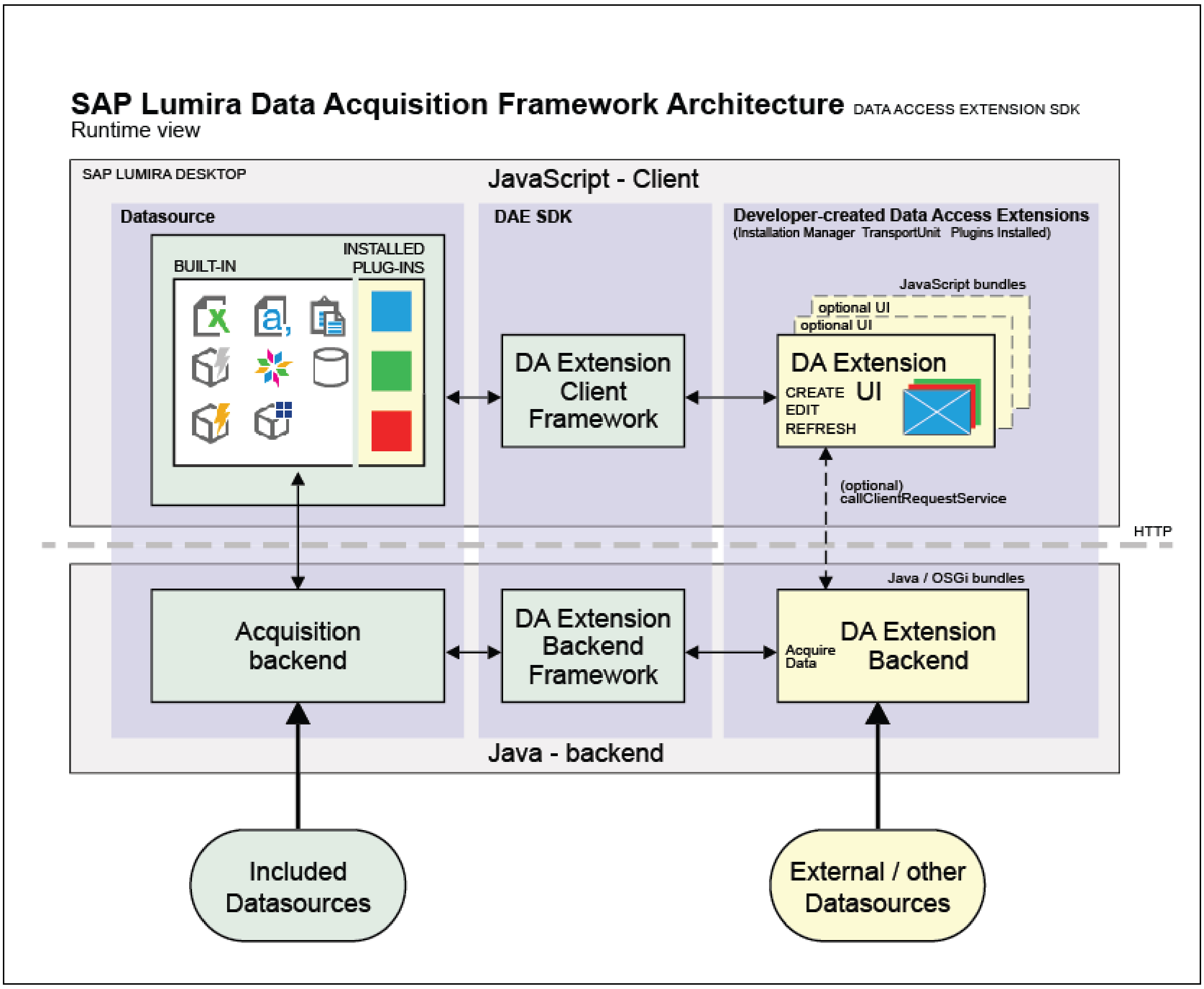
- JavaScript resources (listed here for information purposes only, as Eclipse adds this listing to the project automatically)
    - ECMAScript Built-In Library
    - ECMA 3 Browser Support Library

5 Data Access Extension Implementation

5.1 Architecture of the Data Access Extension

Diagram of the Data Access Extension architecture used in SAP Lumira 1.24 and later.

The following block diagram is a conceptual representation of the SAP Lumira Data Access Extension architecture. Note that the new SDK allows a breakout of JavaScript and HTML5 user interfaces for future server deployment.


SAP Lumira Data Acquisition Framework Architecture DATA ACCESS EXTENSION SDK
Runtime view
SAP LUMIRA DESKTOP
JavaScript - Client
Datasource
DAE SDK
Developer-created Data Access Extensions
(Installation Manager TransportUnit Plugins Installed)
BUILT-IN
INSTALLED PLUG-INS
DA Extension Client Framework
JavaScript bundles
optional UI
optional UI
DA Extension
CREATE
EDIT
REFRESH
UI
(optional) callClientRequestService
HTTP
Java / OSGi bundles
Acquisition backend
DA Extension Backend Framework
Acquire Data
DA Extension Backend
Java - backend
Included Datasources
External / other Datasources

- The **Datasource** shows the data sources included in SAP Lumira, the DAE SDK in the center, and on the right developer-created extensions that are installed using the Extension Manager.
- Data Access Extensions are encased in ZIP archives, called the TransportUnit that the Extension Manager uses to extract and install the components.

- In the SAP Lumira user interface, new data ources are symbolized by the red, green, and blue PLUG-IN icons on the yellow background.
- Developers must create their own icon. (Note that the `exampleextension` included in this guide, uses a solid blue icon.)
- The new extensions that use the SAP Lumira Data Access Extension Framework 1.24 or later are installed using the **Extension Manager** interface.
- Newly added Data Access Extensions (DAE) are called from within the SAP Lumira's *New Dataset* dialog box, (labelled Datasource in the diagram). New DAE's appear alongside the built-in data sources.
- The user interface (UI) components shown in yellow in the JavaScript Client part of the diagram. JavaScript/HTML5 components included in the TransportUnit ZIP file.
-
    - The yellow DA Extension UI box with a solid outline indicates that at least one interface must be made available. The dotted yellow boxes indicate that several other JavaScript components may be included.
    - The `callClientRequestService` is provided as an option to allow communication to the Java backend to include additional functionality.
    - The TransportUnit is the Data Access Extension ZIP plugin module that includes the JavaScript, Java and metadata files.
    - The dotted line between the UI JavaScript / HTML components and the Java back end indicates that there is communication between the two.
    - For future migration and compatibility with Cloud and Server Lumira and other SAP products, UI components must be separated into their own JavaScript, as opposed to embedding this within the Java code.
    - The DAE SDK (Data Access Extension SDK) JAR file included with SAP Lumira is the only dependent item that new extensions need to use. No other external dependencies are to be used to preserve future compatibility. The sampleextension illustrates the use of the JSON library files, which must be included in the project build because they cannot be externally referenced.
- Not shown in this diagram are data access extensions created using the V1 SDK released with SAP Lumira 1.17.
    - The executable extensions created with the original V1 DAE SDK will still function as of the SAP Lumira 1.24 release.
    - First generation extensions are manually installed (and removed) as described in earlier Data Access Extensions for SAP Lumira Developer Guides, versions 1.17 through to 1.22.
    - Data Access Extensions that use the original architecture will only run on SAP Lumira Desktop, not the Server, Cloud or Edge SAP Lumira products.

5.2 API components

A high level summary of the various API components used in the Data Acquisition Extension SDK V2.

AcquisitionState

The `AcquisitionState` object contains information required to perform the data acquisition.

- `info` - This includes all the information required to execute the query plus data persisted for the purpose of restoring the state of the acquisition UI. The contents of this member are serialized as extension-specific acquisition data as part of the Lumira document, and are made available to the extension when the document is reloaded.

Examples of persistent information include:
  - Query and prompt information required to perform acquisitions, plain SDK queries, information used by databases that do not use SQL queries.
  - Version number of the extension, to accommodate future updated versions of the query specification.
  - Form fields in the user interface that may include the user's name, server addresses and port numbers.
- `runtimeInfo` - this is extension-specific acquisition data which does not persist beyond the lifetime of a Lumira session. This member can be used to store transient or sensitive run time information such as login credentials, authentication tokens, and so on.
- `envProps` (environment information) - this is information that Lumira may provide, such as localization information, that would affect how data is imported.

DAEAcquisitionInfo

The `DAEAcquisitionInfo` object contains persistent information that includes:

- the acquisition extension identifier (ID),
- the AcquisitionState query information, such as:
  - an object representing a query for pull data acquisition,
  - a dataset identifier for push data acquisitions.

Data Acquisition Request Broker

The client returns the `AcquisitionState`, which is augmented and marshalled into the `DAEAcquisitionInfo`, which is received by the data acquisition request broker on the back end. This is unmarshalled and the unwrapped AcquisitionState is passed to the appropriate extension back end based on the contents.

The data acquisition request broker accepts the AcquisitionState information, uses the SAP Lumira extension framework to look up the appropriate data access extension to use. The broker acts as the request handler for an agent that makes a request for a data refresh or data acquisition.

SAP Lumira CSV data format

Data must be presented in a CSV format. The originating data must be translated by your Data Access Extension into the CSF format that SAP Lumira expects. Complete details are included in the CSV Data Format Reference section.

Metadata

Metadata that further describes data, such as the column description, hierarchies and so on, are declared in the Metadata file. Complete details are included in the appendix.

5.3 Data Access Workflow

This is an overview of the workflow and the Data Access Extension V2 APIs used.

The key API's used are as follows:

Client

- `doCreateWorkflow(AcquisitionState)`
- `doRefreshWorkflow(AcquisitionState)`

- `doEditWorkflow(AcquisitionState)`

Back end

- getDataAccessJob(props, AcquisitionState)

AcquisitionState

- The structure of the JSON object is as follows:

```
{
    "info" : "",
    "runtimeInfo" : "",
    "envProps" : {
        "datasetName" : "New Dataset",
        "productLocale" : "en",
        "preferredViewingLocale" : "en_US"
    }
}
```

The SAP Lumira Desktop workflow is as follows:

File > New

- Calls `doCreateWorkflow()`, returns a populated `AcquisitionState`
    - If opened from the most recently used shortcut, the corresponding AcquisitionState is passed to it to populate the user interface.
        - The `AcquisitionState` here is a serialized form, there is no `runtimeInfo` member. (no login credentials or authentication information such as tokens)
    - returns the `AcquisitionState` for data acquisition
- `AcquisitionState` is passed on to the back end
- Request broker calls `getDataAccessJob()` with `AcquisitionState`

Edit

- this calls `doEditWorkflow()`, which uses the existing `AcquisitionState`

Refresh

- calls `doRefreshWorkflow()` with existing `AcquisitionState`, which includes the last entered credentials

Save

- Saves the `LUMS` document, the most recently used (MRU) that saves the info member data from `AcquisitionState`

Publish

- Publishes the `LUMS` document with the info member from `AcquisitionState`

5.4 Workflow UI Returned Objects

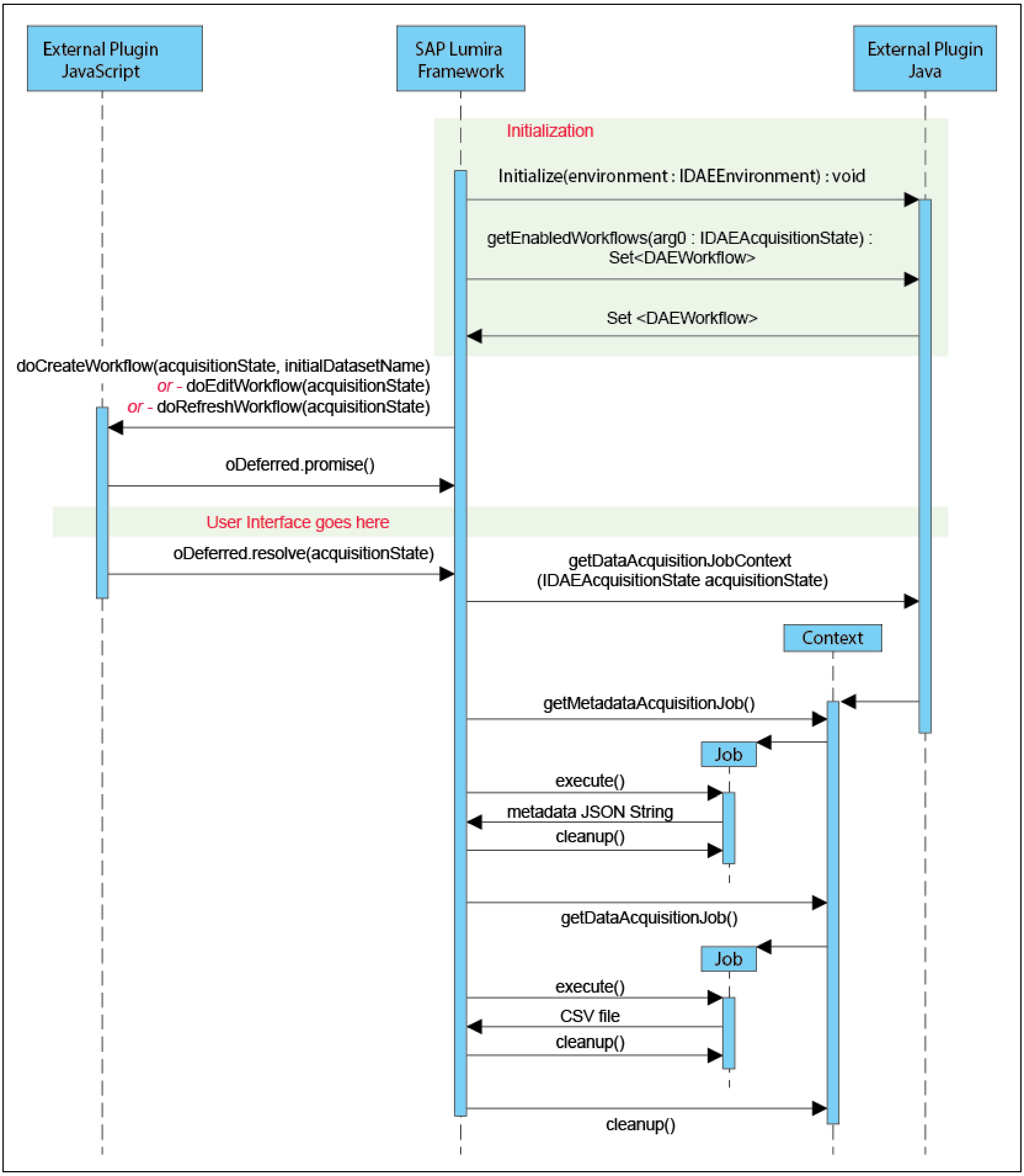
The `acquisitionState` functions will always return a `deferred.promise()` object that will be resolved when the UI is finished.

The promise's done functions will be passed a JSON object as follows:

```
extensionUI.doCreateWorkflow.done(function (acquisitionState, suggestedDatasetName)
{
    // This code runs when the UI has completed its task.
}).fail(function () {
   // This code runs if the UI was canceled or it failed.
   // Supply information as to why it failed.
});
```

5.5 Data Access Extension Development Workflow

The following UML sequence diagram represents the minimum use of the data access extension API that can be used to create a plugin.


External Plugin JavaScript
SAP Lumira Framework
External Plugin Java
Initialization
Initialize(environment : IDAEEnvironment) : void
getEnabledWorkflows(arg0 : IDAEAcquisitionState) : Set<DAEWorkflow>
Set <DAEWorkflow>
doCreateWorkflow(acquisitionState, initialDatasetName)
or - doEditWorkflow(acquisitionState)
or - doRefreshWorkflow(acquisitionState)
oDeferred.promise()
User Interface goes here
oDeferred.resolve(acquisitionState)
getDataAcquisitionJobContext (IDAEAcquisitionState acquisitionState)
Context
getMetadataAcquisitionJob()
Job
execute()
metadata JSON String
cleanup()
getDataAcquisitionJob()
Job
execute()
CSV file
cleanup()
cleanup()

The two right-most swim lanes, **SAP Lumira Framework** and **External Plugin Java** shows the sequence of API's that are used to initialize the data acquisition process when it is called from the SAP Lumira *Add new dataset* dialog box.

On the left, the External Plugin JavaScript is started when the user interface is called, and then a choice of creating a new dataset, editing an existing dataset or just refreshing a dataset is communicated.

Initialization (Java)

The initialize section Initialize is provided through the IDAEEnvironment <<interface>>, which does an environment check. Checks can include if 32-bit or 64-bit is required, the version of Lumira required, the trust level required, the location of the temporary directory, and so on.

Workflows (JavaScript)

The JavaScript user interface is the starting point for triggering what workflow is used. The workflows available to use with your data access extension are as follows:

- **Create** - `doCreateWorkflow(acquisitionState, initialDatasetName)` is invoked from a UI to establish the acquisition parameters and any login information or preliminary setup (to populate `initialDatasetName`) required to import new data into SAP Lumira for the first time.
- **Edit** - Using `doEditWorkflow(acquisitionState)` is Similar to Create, but Edit allows changes to an existing acquisition, which SAP Lumira will then use to update your visualization.
- **Refresh** - When a user wants to update their visualization with more recent data, `doRefreshWorkflow(acquisitionState)` is called. When Database credentials were previously entered since opening the document, they do not have to be re-entered, and all the parameters and queries used are remembered.

Additional user interface options may be included in the JavaScript portion of the extension. This is indicated in the green bar labelled User interface goes here. This might include previewing the data, turning on and off logging functions, unique login sequences, and so on.

- In the example sampleextension in this guide that demonstrates additional functionality that may be added, there is a "Ping" button that brings up a response dialog box "pong."

Job execution (Java)

Once the user interface interaction is completed, the process of acquiring the data starts with `getDataAcquisitionJobContext (IDAEAcquisitionState acquisitionState).`

Using the example sampleextension project included in this user guide for reference, the metadata, which is a JSON formatted file, is the `Job` that reads the associated information that describes the column `name`, the `id`, the data `type`, and `analyticalType` information.

The next `Job` reads the data into SAP Lumira, then the `cleanup()` completes the function of the extension.

5.6 Development Considerations

Dependency Considerations

Following these recommendations reduces the likelihood of future complications with subsequent releases of SAP Lumira.

- Data Access Extensions must be built as a single OSGi bundle (www.osgi.org).
- All dependencies (JAR files, for example), must be included in the bundle.
- The only external dependency that a Data Access Extension should have is the SAP Lumira Data Access Framework SDK library.
- Do not depend on any other SAP components that are included with SAP Lumira.

Security Considerations

Note that when dealing with sensitive information such as login credentials, care should be taken not to unintentionally include these as part of `AcquisitionState.info`, as this member is persisted with a Lumira document. Instead, use `AcquisitionState.runtimeInfo`.

User Interface Considerations

- The user interface supplied by the Data Access Extension must ensure all operations are completed within the HTML dialog modally, before allowing interaction with SAP Lumira.
- If the user interface is not HTML, it must provide an HTML dialog modally that then launches the non-HTML user interface.

Refresh Workflow Considerations

- Regardless of what client the extension runs on (Cloud, Server, Desktop), `Refresh` does not require a user interface. This means no prompting for input, filter selection and so on - only the previously supplied values are used.
- Exceptions are authentication prompts.
- In the event of an incomplete or cancelled refresh, your extension must throw a meaningful error that includes the fact the failure occurred, the reason for the failure, and the remedy.

Metadata and CSV data Considerations

- Thoroughly test the Data Access Extension data import functionality to avoid data corruption, errors or other complications arising from mismatched CSV and metadata information.
- The only acceptable delimiter as of this release is the comma. Other delimiters will not work and may not be reassigned.
- The CSV and Metadata must be encoded with ASCII or preferably, UTF-8.

Compatibility Considerations

Some thought should be given to the format of the `AcquisitionState.info` member, as its contents are managed entirely by the associated data access extension. One area of potential complexity is in the area of compatibility with respect to your extension's product evolution. Questions to ask:

- Will your extension evolve?
- Will the format of `info` also evolve? If so, consider including the following features within your extension:
    - Provision of version number detection as part of `AcquisitionState.info`, to allow users with older extensions to be alerted to the fact they are trying to use SAP Lumira LUMS document created with a newer version of the extension.
    - Use of an extensible format such as XML or JSON to make it easier to evolve the contents of `Acquisition.info`.

5.7 Extension Icon

Description of the custom icons that are used in the Data Access Extension.

The Data Access Extension icon must be supplied to confirm to format used by the built-in data sources. You can use a generic external datasource icon, or one of your own design. The `getIcon`## must return a path to an image with the size in pixels.

- 16 x 16 icon used in the New Dataset dialog box, in the most recently used list

- 24 x 24 icon not currently visible, but must be supplied
- 32 x 32 icon used in the New Dataset dialog box
- 32 x 32 white icon used in the New Dataset dialog box when the extension is selected
- 48 x 48 icon not currently used but must be supplied

Table 1: Extension Icons

Icon	Description
🟦	16 x 16 icon used in the New Dataset dialog box, in the most recently used list `getIcon16()`
🟦	24 x 24 icon not currently visible, but must be supplied `getIcon24()`
🟦	32 x 32 icon used in the New Dataset dialog box `getIcon32()`
🟦	32 x 32 white icon used in the New Dataset dialog box when the extension is selected `getIcon32_white()`
🟦	48 x 48 icon not currently used but must be supplied `getIcon48()`

6 SDK Use and API Reference

6.1 Naming your Data Access Extension

The name of a Data Access Extension must be a unique identifier `id`.

Before writing an extension, the first thing you need to decide on is a unique identifier of your extension, which is referred to as `id` shown in the `bundle.js` and `feature.json` metadata files. It is recommended that you follow a hierarchical naming convention, for example, yourcompany, department, extensionName.

For example:

- `sap` is the company name
- `lumira` is the department
- `sampleextension` is the extension name

```
sap.lumira.sampleextension
```

6.2 Data Access Extension Transport Unit

The file TransportUnit.zip contains component parts of a complete Data Access Extension.

The transport unit consists of the folowing folders:

- `features` - a JSON formatted file, `<extensionname>-feature.json`
- `eclipse` - a Java JAR file, `<domain>.<extensionname>.<versionNumber>.jar`
- `bundles` - icon images (5 sizes) and `<extensionfilename>-.bundle.js` and other supporting JavaScript files.

6.2.1 Format of the <-feature.json> file

Format of the `feature.json` file located inside the ZIP Transport Unit extension in JSON format. This is where metadata such as the name and version information for the extension is stored.

The file is located within the ZIP Data Access Extension Transport Unit archive at:

- `features > sap > lumira > <extensionname> > <extensionname>-feature.json`

Table 2: *-feature.json format

Name	Description	Example
`metadataVersion`	Extension short revision number.	`1.0`
`id`	Unique ID of the extension. This ID and the ID found in "bundles" (example: `sampleextension-bundle.js`) must be the same.	`sap.lumira.sampleextension`
`name`	English name of the extension.	`SAP Lumira Sample Extension`
`description`	Description text of the extension.	`for SAP Lumira Data Acquisition Framework`
`version`	Extension long version number.	`1.24.0.201503101453`
`vendor`	Parent for `name` and `url`.	
`name`	Child of `vender` - company name.	`SAP`
`url`	Child of `vendor` - http address.	`www.sap.com`
`requires`	SAP Lumira version required dependency section. This is the parent of `id`, `version`, and `match`.	
`id`	Unique id name of the extension library this extension depends on.	`sap.bi.da.extension.sdk`
`version`	The version of DAE SDK library JAR file the extension needs to run.	`1.24.0`
`match`	The version of SAP Lumira required, only a specific version number, or that version or later. Choices are `greaterOrEqual`, or `perfect`.	`greaterOrEqual`
`eclipse`	Parent for `plugins`	
`plugins`	Parent for `id` and `version`.	
`id`	Unique ID string.	`sap.lumira.sampleextension`
`version`	Revision number in major.minor.patch.build format.	`1.24.0.201503101453`

Example `feature.json` file inside the `features` folder within the Transport Unit ZIP archive:

```
{
  "metadataVersion": "1.0",
  "id": "sap.lumira.sampleextension",
  "name": "SAP Lumira Sample Extension",
  "description": "for SAP Lumira Data Acquisition Framework",
  "version": "1.24.0.201502171756",
  "vendor" : {
    "name": "SAP",
    "url": "www.sap.com"
  },
  "requires": [
    {
      "id": "sap.bi.da.extension.sdk",
      "version": "1.24.0",
      "match": "greaterOrEqual"
    }
  ],
  "eclipse": {
    "plugins": [
      {
        "id": "com.sap.lumira.sampleextension",
        "version": "1.24.0.201502171756"
      }
    ]
  },
  "bundles": [
    {
      "id": "sap.lumira.sampleextension",
      "version": "1.24.0.201502171756"
    }
  ]
}
```

6.2.2 Format of the <-bundle.js> file

Format of the `<extensionfilename>-bundle.js` file located inside the ZIP Transport Unit extension in JSON format. This is where metadata such as the name and version information is stored.

The file is located within the ZIP Data Access Extension Transport Unit archive at:

- bundles > sap > lumira > <extensionname> > <extensionname>-bundle.js

Table 3: *-bundle.js format

Name	Description	Example
`id`	The unique ID of the extension. This ID and the ID found in the "features" (example: `sampleextension-feature.json`) must be the same.	`sap.lumira.sampleextension`
`version`	The extension long version number.	`1.24.0.201503101453`
`components`	Parent for `id` and `provide` (etc.)	

Name	Description	Example
id	The unique ID of the extension (same as above).	sap.lumira.sampleextension
provide	The provider and extension domain name.	sap.lumira.sampleextension
module	The module name, no spaces.	SampleExtension
customProperties	Parent for displayName and description.	
displayName	The readable name, with spaces, of the extension as it will appear in the New Dataset dialog box.	SAP Lumira Sample Extension
description	The description phrase as it will appear in the New Dataset dialog box.	for SAP Lumira Data Acquisition Framework

Example of the samplextension-bundle.js inside the bundles folder within the Transport Unit ZIP archive:

```
define([], function() {
    return sap.bi.framework.declareBundle({
        "id" : "sap.lumira.sampleextension",
        "version" : "REPLACE_VERSION",
         "components" : [{
            "id" : "sap.lumira.sampleextension",
            "provide" : "sap.bi.da.extension.client",
            "module" : "SampleExtension",
            "customProperties" : {
                "displayName" : "SAP Lumira Sample Extension",
                "description" : "for SAP Lumira Data Acquisition Framework"
            }
        }],
        "dependencies": ["sap.bi.da.extension.sdk.clientRequestService"]
    });
});
```

6.2.3 TransportUnit (ZIP archive) Structure

This topic describes the components that are required in the Data Access Extension ZIP file.

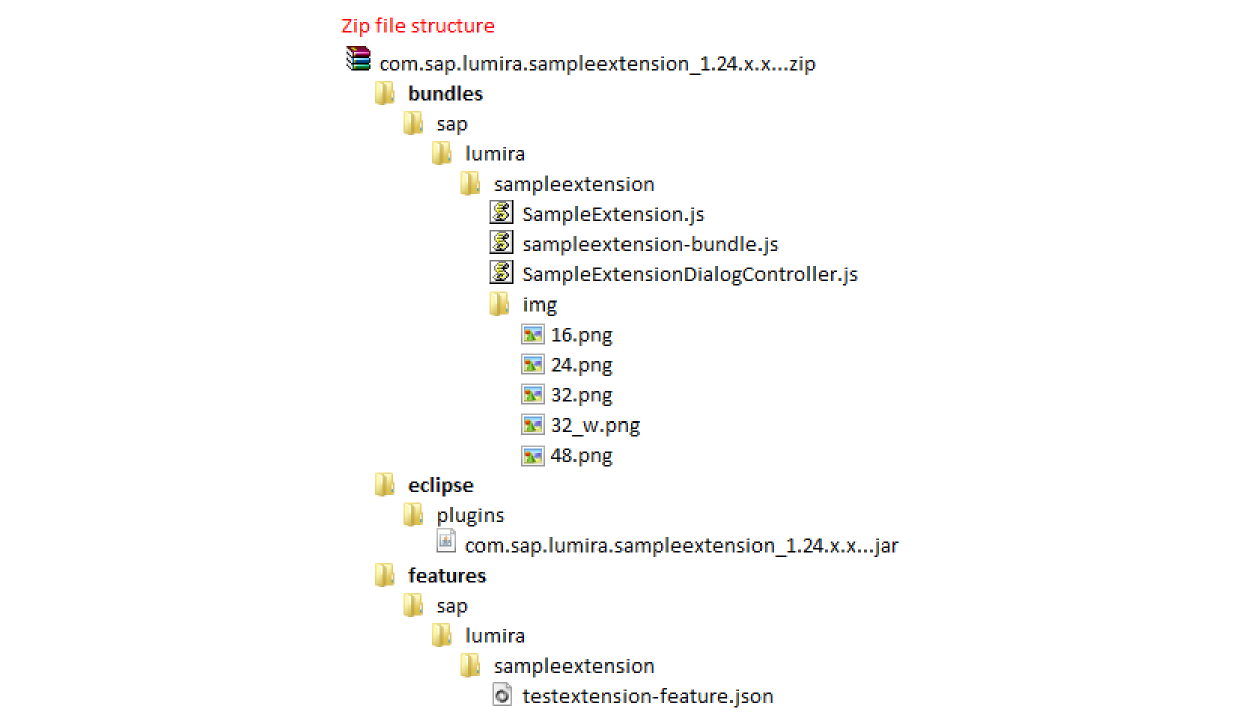
TransportUnit.ZIP

- eclipse > plugins > extensionName_versionNumber.jar
- features > sap > vi > desktop > extension > extensionName > extensionName-feature.json
- bundles > sap > lumira sampleextension > img (*.png folder of icons), and *.js (JavaScript files)

Table 4: Files in the TransportUnit archive

File inside the ZIP archive	Description
extensionName_versionNumber.jar	This is the Java archive that contains the extension Java code and Javascript support files. The file must follow a naming convention.
extensionName-feature.json	This is a JSON file that contains a description of the features of the extension for SAP Lumira, such as the version number, ID, description and so on.

An example of the structure of the `sampleextension` example ZIP file appears as follows:



Zip file structure

```
com.sap.lumira.sampleextension_1.24.x.x...zip
  bundles
    sap
      lumira
        sampleextension
          SampleExtension.js
          sampleextension-bundle.js
          SampleExtensionDialogController.js
          img
            16.png
            24.png
            32.png
            32_w.png
            48.png
  eclipse
    plugins
      com.sap.lumira.sampleextension_1.24.x.x...jar
  features
    sap
      lumira
        sampleextension
          testextension-feature.json
```

6.3 Client Plugin JavaScript

6.3.1 <bundle>-bundle.js

The name of the extension, which is also the name of last folder that contains this file, is prefixed to the "-bundle.js" name.

Here is an example of a bundle.js file:

```
define([], function() {
    return sap.bi.framework.declareBundle({
        "id" : "sap.lumira.sampleextension",
        "version" : "REPLACE_VERSION",
         "components" : [{
            "id" : "sap.lumira.sampleextension",
            "provide" : "sap.bi.da.extension.client",
            "module" : "SampleExtension",
            "customProperties" : {
                "displayName" : "SAP Lumira Sample Extension",
                "description" : "for SAP Lumira Data Acquisition Framework"
            }
        }],
        "dependencies": ["sap.bi.da.extension.sdk.clientRequestService"]
    });
});
```

6.3.2 <myUI>.js

The user interface is written using JavaScript.

This file must be in the same folder as the `bundle.js` file.

Here is an incomplete example of a `<myUI>.js` file, which according to the bundle.js example above would be called `SampleExtension.js`.

In this example, there is a second file called `SampleExtensionDialogController.js` alongside the main `<myUI>.js` file. This file is included through `RequireJS`.

```
define(["service!sap.bi.da.extension.sdk.clientRequestService",
"SampleExtensionDialogController"],
         function (ServiceHelper, SampleExtensionDialogController) {
    "use strict";
    function SampleExtension() {

        var EXTENSION_ID = "sap.lumira.sampleextension";
        this._path = jQuery.sap.getModulePath(EXTENSION_ID);
        // fServiceCall can be used to call the ClientRequestJob defined on the
Java side
        var fServiceCall = function(request, fSuccess, fFailure) {
            ServiceHelper.callClientRequestService(EXTENSION_ID, request, fSuccess,
fFailure);
        };
        this.doCreateWorkflow = function(acquisitionState) {
            var oDeferred = new jQuery.Deferred();
            ...
            return oDeferred.promise();
        };
        this.doEditWorkflow = function(acquisitionState) {
            var oDeferred = new jQuery.Deferred();
            ...
            return oDeferred.promise();
        };
        this.doRefreshWorkflow = function(acquisitionState) {
            var oDeferred = new jQuery.Deferred();
            ...
            return oDeferred.promise();
        };
    }
    // the framework will consider two connections with the same description
    // (title + subtitle) to be duplicates and remove one
    // if the function fails somehow the Framework will default to using the
    // dataset name as the title and have no subtitle
    SampleExtension.prototype.getConnectionDescription = function(acquisitionState)
{
        var info = JSON.parse(acquisitionState.info);
        return {
            Title: info.datasetName,
            SubTitle: info.csv
        };
```

```
     };
     SampleExtension.prototype.getIcon48 = function() {
         return this._path + "/img/48.png";
     };
     SampleExtension.prototype.getIcon32 = function() {
         return this._path + "/img/32.png";
     };
     SampleExtension.prototype.getIcon32_white = function() {
         return this._path + "/img/32_w.png";
     };
     SampleExtension.prototype.getIcon24 = function() {
         return this._path + "/img/24.png";
     };
     SampleExtension.prototype.getIcon16 = function() {
         return this._path + "/img/16.png";
     };
     return SampleExtension;
});
```

The function passed to the define call must return a constructor function. Functions can be declared in the constructor or as part of the prototype, depending on if it needs access to private variables declared in the constructor.

`ServiceHelper.callClientRequestService` is described below in the "Calling backend plugin code" section.

`acquisitionState.envProps` is an array of properties that SAP Lumira will pass to your extension. It represents information from the Lumira environment that your extension can use.

The `getIcon##` functions must return a relative path to icon files (square, with size in pixels corresponding to the number). The paths are relative to your `bundle.js`. For example, return "`icon64.png`" if your icon is in the same folder as your `bundle.js`. When the UI is done with a particular workflow it has to call resolve or reject on the Deferred objects returned in the workflow functions:

```
// workflow succeeded for edit/refresh
oDeferred.resolve(acquisitionState);
//
// workflow succeeded for create
oDeferred.resolve(acquisitionState, suggestedDatasetName);
//
// workflow failed
oDeferred.reject();
```

Currently supported properties

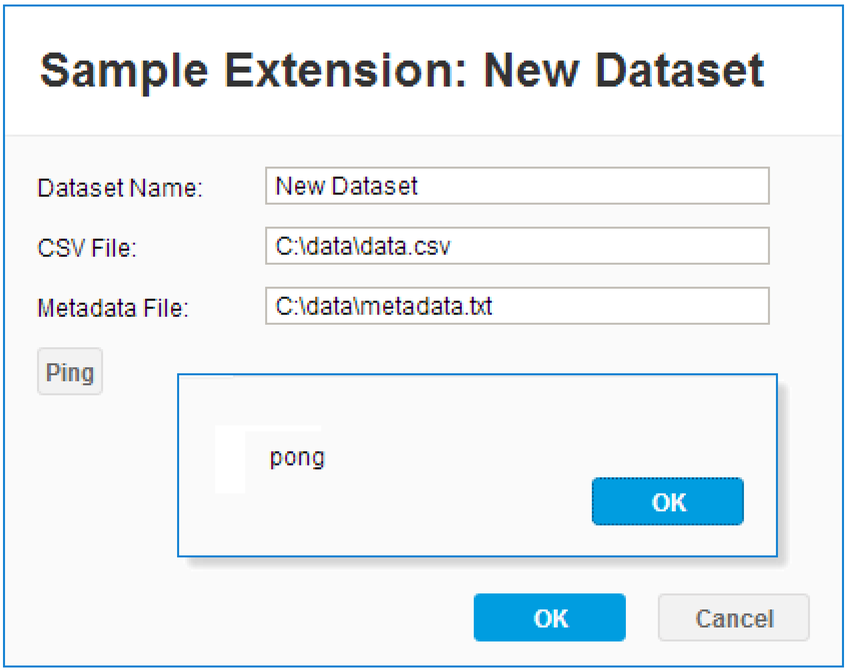
`acquisitionState.envProps.productLocale` is the locale of the Lumira product. Use this to make sure your extension user interface displays in the same language as the rest of the product.

`acquisitionState.envProps.preferredViewingLocale` is used to set the locale to ensure that culture-specific data is imported correctly.

6.3.2.1 Dialog Box Javascript Example

Example of Javascript used to create a dialog box.

The following dialog box illustrates the creation of a user interface dialog box that facilitates selection of a datasource, along with a number of other features. This is used in the sampleextension example, `SampleExtensionDialogController.js`.



Sample Extension: New Dataset

Dataset Name: New Dataset

CSV File: C:\data\data.csv

Metadata File: C:\data\metadata.txt

pong

The following example JavaScript code shows how to create the preceeding dialog box, and demonstrates the communication with the Data Access Extension back end.

```
define(function() {
    "use strict";
    var SampleExtensionDialogController = function(acquisitionState, oDeferred,
fServiceCall, workflow) {
        /*
        Create dialog controls
        */
        var dLayout = new sap.ui.commons.layout.MatrixLayout({
            layoutFixed : true,
            columns : 2,
            width : "570px",
            widths : [ "20%", "80%" ]
        });
        var datasetNameTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : "",
            enabled : workflow === "CREATE"
        });
        var datasetNameLbl = new sap.ui.commons.Label({
            text : "Dataset Name:",
            labelFor : datasetNameTxt
        });
        dLayout.createRow({
            height : "30px"
        }, datasetNameLbl, datasetNameTxt);
        // These paths correspond to the included sample data if the workspace was
unzipped to the C drive
        var datasetTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : 'C:\\workspace_SampleExtension\\Sample Data\\data.csv'
        });
        var datasetLbl = new sap.ui.commons.Label({
            text : "CSV File:",
            labelFor : datasetTxt
        });
        dLayout.createRow({
            height : "30px"
        }, datasetLbl, datasetTxt);
        var metadataTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : 'C:\\workspace_SampleExtension\\Sample Data\\metadata.txt'
        });
        var metadataLbl = new sap.ui.commons.Label({
            text : "Metadata File:",
            labelFor : metadataTxt
        });
        dLayout.createRow({
            height : "30px"
        }, metadataLbl, metadataTxt);
        // Client request call example
        var pingBtn = new sap.ui.commons.Button({
            press : [ function() {
                fServiceCall("ping", function(response) {
                    sap.ui.commons.MessageBox.alert(response);
                }, function(actionRequired, errorMessage, fullErrorObject) {
                    sap.ui.commons.MessageBox.alert(errorMessage);
                });
            }, this ],
            text : "Ping",
            enabled : true
        }).addStyleClass("button_ellipsis");
        dLayout.createRow({
            height : "30px"
        }, pingBtn);
        /*
        Button press events
        */
        var buttonCancelPressed = function() {
            dialog.close(); // dialog is hoisted from below
        };
        var buttonOKPressed = function() {
            var info = {};
            info.csv = datasetTxt.getValue();
            info.metadata = metadataTxt.getValue();
            info.datasetName =  datasetNameTxt.getValue();
            acquisitionState.info = JSON.stringify(info);
            oDeferred.resolve(acquisitionState, datasetNameTxt.getValue());
            dialog.close();
        };
        var okButton = new sap.ui.commons.Button({
            press : [ buttonOKPressed, this ],
            text : "OK",
            tooltip : "OK"
        }).setStyle(sap.ui.commons.ButtonStyle.Accept);
        var cancelButton = new sap.ui.commons.Button({
            press : [ buttonCancelPressed, this ],
            text : "Cancel",
            tooltip : "Cancel"
        }).addStyleClass(sap.ui.commons.ButtonStyle.Default);
        var onClosed = function() {
            if (oDeferred.state() === "pending") {
                oDeferred.reject();
            }
        };
        /*
        Modify controls based on acquisitionState
        */
        var envProperties = acquisitionState.envProps;
        if (acquisitionState.info) {
            var info = JSON.parse(acquisitionState.info);
            datasetTxt.setValue(info.csv);
            metadataTxt.setValue(info.metadata);
            envProperties.datasetName = info.datasetName;
        }
        datasetNameTxt.setValue(envProperties.datasetName);
        /*
        Create the dialog
        */
        var dialog = new sap.ui.commons.Dialog({
            width : "720px",
            height : "480px",
            modal : true,
            resizable : false,
            closed : onClosed,
            content: [dLayout],
            buttons : [okButton, cancelButton]
        });
        dialog.setTitle("Sample Extension: " + envProperties.datasetName);
        this.showDialog = function() {
            dialog.open();
        };
    };
    return SampleExtensionDialogController;
});
```

The corresponding Java code that responds to the clicking of the "Ping" button and sends back the "pong" response, is shown in the following code snippet, which is taken from the **sampleextension** example, `SampleExtension.java`.

```
private class SampleExtensionClientRequestJob implements IDAEClientRequestJob {
String request;
SampleExtensionClientRequestJob(String request) {
    this.request = request;
}
@Override
public String execute(IDAEProgress callback) throws DAException {
    if ("ping".equals(request)) {
        return "pong";
    }
    return null;
}
@Override
public void cancel() {
    // Cancel is currently not supported
}
@Override
public void cleanup() {
    // This function is NOT called
}
}
```

6.3.2.2 AcquisitionState - JSON Object

The structure of the JSON object is as follows:

```
{
    "info" : "",
    "runtimeInfo" : "",
    "envProps" : {}
}
```

See documentation for the interface IDAEAcquisitionState as that and this object are identical.

AcquisitionState will be passed to your UI through the UI workflows:

```
    doCreateWorkflow
    doRefreshWorkflow
    doEditWorkflow
```

The `doCreateWorkflow` should also return a `suggestedDatasetName` string.

It is your UI's responsibility to return a `AcquisitionState` object after each UI workflow. For example:

- Returning a query which represents your acquisition. This query is passed to your java extension at acquisition time.
- If you need some kind of authentication to perform your acquisition, returning valid credentials in the `runtimeInfo` field. Lumira will pass credentials that you supply, immediately to your java extension during acquisition time.

6.3.3 Calling Backend Plugin Code

In your JavaScript code, Lumira provides the a `RequireJS` service module for you to invoke:

```
define(["service!sap.bi.da.extension.sdk.clientRequestService"], function
(ClientRequestService) {
    "use strict";
    ...
    /**
     * @param request a string, which will be passed as-is to your backend
     *        IDAExtension#getClientRequestJob implementation
     * @param fSuccess a callback function, invoked upon success.
     *        It takes 1 argument - the response value returned by your
     *        backend IDAEClientRequestJob#execute implementation.  See below.
     * @param fFailure a callback function, invoked upon failure.  It takes 3
     *        arguments - actionRequired, errorMessage, fullErrorObject.
     */
    serviceHelper.callClientRequestService(extensionId, request, fSuccess,
fFailure);
});
```

6.4 Back End Plugin Class Reference

6.4.1 Extension API Summary List

Summary links of the available Data Access Extension interfaces.

Table 5: Interface Summary

Interface	Description
IDAEAcquisitionJobContext	The context for a data acquisition.
IDAEAcquisitionState	

Interface	Description
IDAEClientRequestJob	
IDAEDataAcquisitionJob	
IDAEEnvironment	
IDAEJob<T>	This interface represents a data acquisition extension job to be used with the **Lumira Data Acquisition framework**.
IDAEMetadataAcquisitionJob	
IDAEProgress	Callback interface for clients to be notified as job states are updated.
IDAExtension	This interface represents a data acquisition extension to be used with the **Lumira Data Acquisition framework** You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.

Table 6: Class Summary

Class	Description
DAEConstants	
DAEConstants.Metadata	
DAEConstants.Metadata.Keys	

Table 7: Enum Summary

Enum	Description
DAEConstants	
DAEConstants.Metadata	
DAEConstants.Metadata.Keys	

Table 8: Exception Summary

Exception	Description
DAException	Exception to be thrown when a problem is encountered during the execute() method of a DAE Job.

6.4.2 Overview of Classes

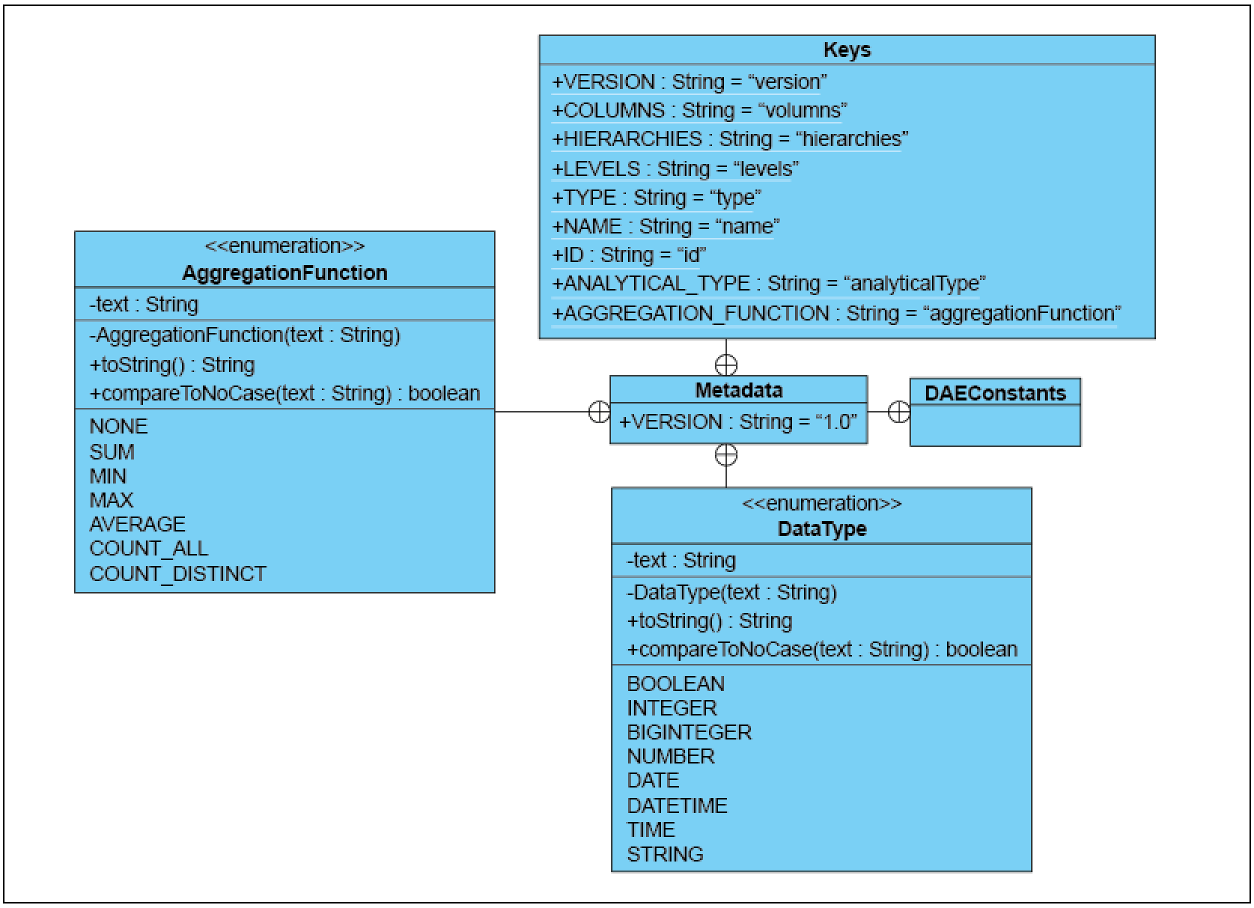
This is a listing of available classes in the SAP Lumira Data Acquisition SDK.

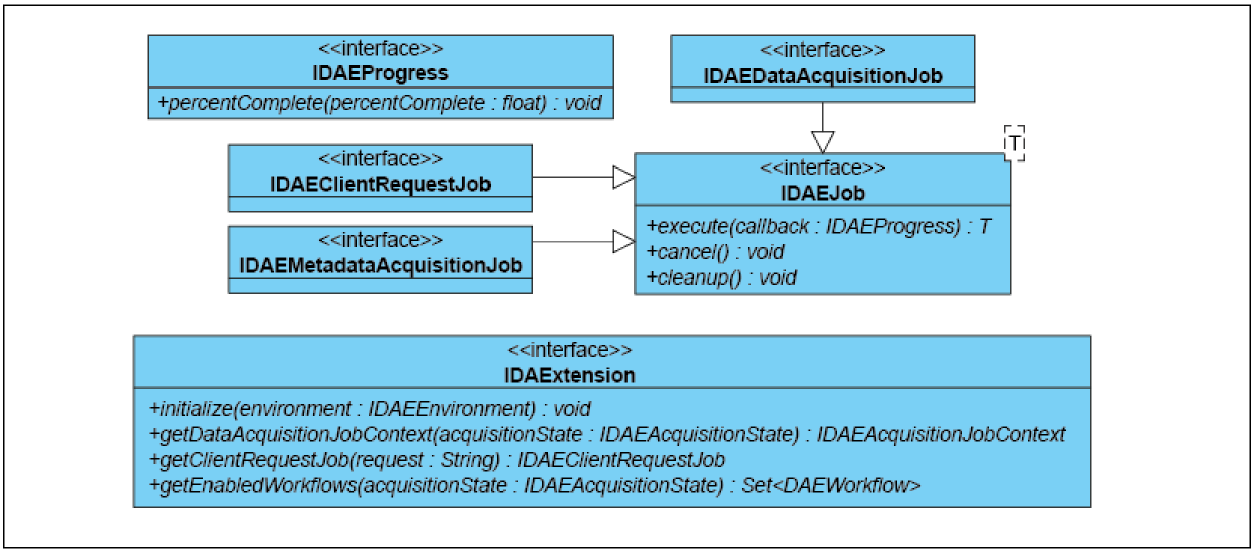
Table 9: Class List for SAP Lumira Data Access Extension SDK

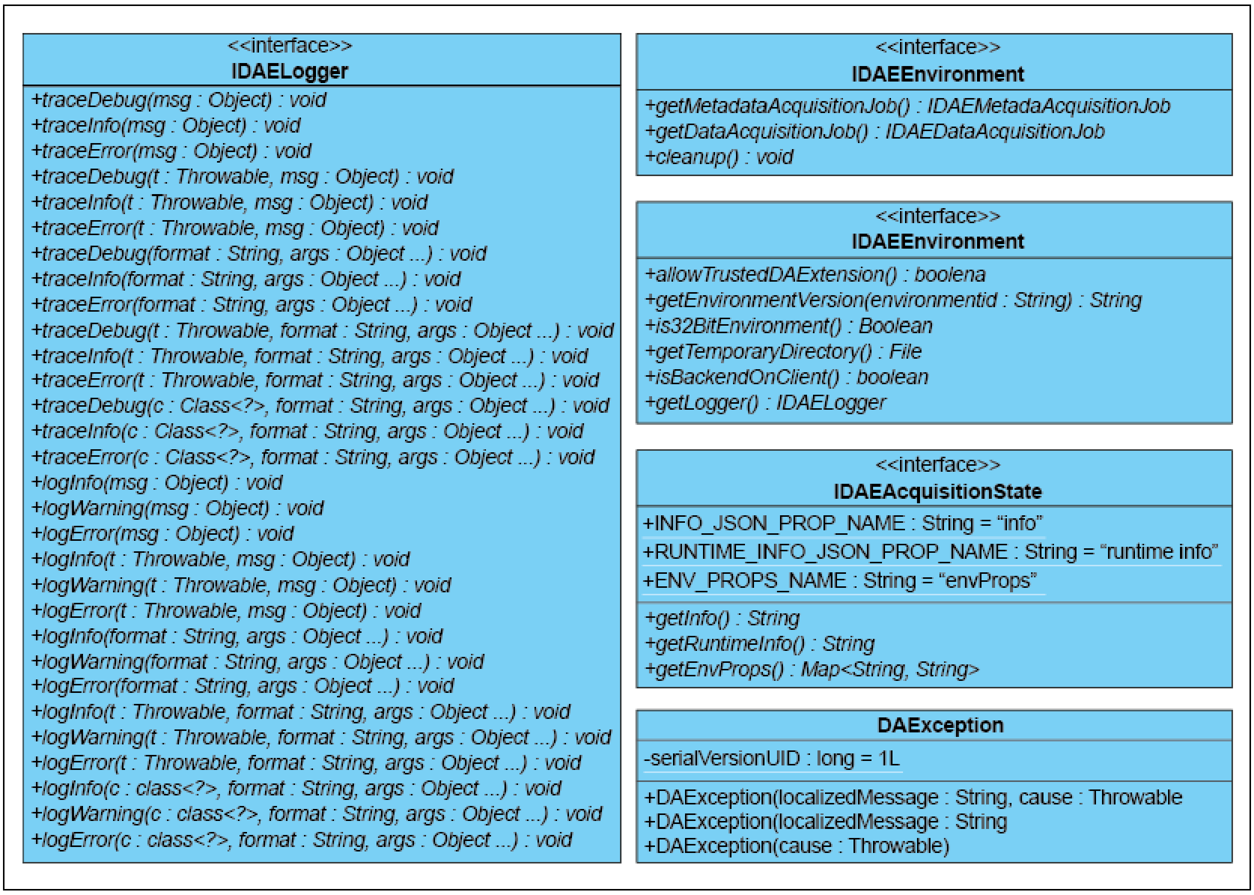
Name	Type	Description
DAEConstants	Class	
DAEConstants.Metadata	Class	
DAEConstants.Metadata.Keys	Class	
DAEException	Class	Exception to be thrown when a problem is encountered during the execute() method of a DAE Job.

Name	Type	Description
DAEConstants.Metadata.AggregationFunction	Enum	
DAEConstants.Metadata.DataType	Enum	
DAEWorkflow	Enum	
IDAEAcquisitionJobContext	Interface	The context for a data acquisition.
IDAEAquisitionState	Interace	
IDAEClientRequestJob	Interface	
IDAEDataAcquisitionJob	Interface	
IDAEEnvironment	Interface	
IDAEJob<T>	Interface	This interface represents a data acquisition extension job to be used with the **Lumira Data Acquisition framework**
IDAELogger	Interface	This interface represents the logger passed to the extension.
IDAEMetadataAcquisitionJob	Interface	
IDAEProgress	Interface	Callback interface for clients to be notified as job states are updated
IDAExtension	Interface	This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.

The following Class diagrams list the available API's in this SDK.


Keys
+VERSION : String = "version"
+COLUMNS : String = "volumns"
+HIERARCHIES : String = "hierarchies"
+LEVELS : String = "levels"
+TYPE : String = "type"
+NAME : String = "name"
+ID : String = "id"
+ANALYTICAL_TYPE : String = "analyticalType"
+AGGREGATION_FUNCTION : String = "aggregationFunction"
<<enumeration>>
AggregationFunction
-text : String
-AggregationFunction(text : String)
+toString() : String
+compareToNoCase(text : String) : boolean
NONE
SUM
MIN
MAX
AVERAGE
COUNT_ALL
COUNT_DISTINCT
Metadata
+VERSION : String = "1.0"
DAEConstants
<<enumeration>>
DataType
-text : String
-DataType(text : String)
+toString() : String
+compareToNoCase(text : String) : boolean
BOOLEAN
INTEGER
BIGINTEGER
NUMBER
DATE
DATETIME
TIME
STRING


<<interface>>
IDAEProgress
+percentComplete(percentComplete : float) : void
<<interface>>
IDAEDataAcquisitionJob
<<interface>>
IDAEClientRequestJob
<<interface>>
IDAEMetadataAcquisitionJob
T
<<interface>>
IDAEJob
+execute(callback : IDAEProgress) : T
+cancel() : void
+cleanup() : void
<<interface>>
IDAExtension
+initialize(environment : IDAEEnvironment) : void
+getDataAcquisitionJobContext(acquisitionState : IDAEAcquisitionState) : IDAEAcquisitionJobContext
+getClientRequestJob(request : String) : IDAEClientRequestJob
+getEnabledWorkflows(acquisitionState : IDAEAcquisitionState) : Set<DAEWorkflow>



```
<<interface>>
IDAELogger
+traceDebug(msg : Object) : void
+traceInfo(msg : Object) : void
+traceError(msg : Object) : void
+traceDebug(t : Throwable, msg : Object) : void
+traceInfo(t : Throwable, msg : Object) : void
+traceError(t : Throwable, msg : Object) : void
+traceDebug(format : String, args : Object ...) : void
+traceInfo(format : String, args : Object ...) : void
+traceError(format : String, args : Object ...) : void
+traceDebug(t : Throwable, format : String, args : Object ...) : void
+traceInfo(t : Throwable, format : String, args : Object ...) : void
+traceError(t : Throwable, format : String, args : Object ...) : void
+traceDebug(c : Class<?>, format : String, args : Object ...) : void
+traceInfo(c : Class<?>, format : String, args : Object ...) : void
+traceError(c : Class<?>, format : String, args : Object ...) : void
+logInfo(msg : Object) : void
+logWarning(msg : Object) : void
+logError(msg : Object) : void
+logInfo(t : Throwable, msg : Object) : void
+logWarning(t : Throwable, msg : Object) : void
+logError(t : Throwable, msg : Object) : void
+logInfo(format : String, args : Object ...) : void
+logWarning(format : String, args : Object ...) : void
+logError(format : String, args : Object ...) : void
+logInfo(t : Throwable, format : String, args : Object ...) : void
+logWarning(t : Throwable, format : String, args : Object ...) : void
+logError(t : Throwable, format : String, args : Object ...) : void
+logInfo(c : class<?>, format : String, args : Object ...) : void
+logWarning(c : class<?>, format : String, args : Object ...) : void
+logError(c : class<?>, format : String, args : Object ...) : void
```

```
<<interface>>
IDAEEnvironment
+getMetadataAcquisitionJob() : IDAEMetadaAcquisitionJob
+getDataAcquisitionJob() : IDAEDataAcquisitionJob
+cleanup() : void
```

```
<<interface>>
IDAEEnvironment
+allowTrustedDAExtension() : boolena
+getEnvironmentVersion(environmentid : String) : String
+is32BitEnvironment() : Boolean
+getTemporaryDirectory() : File
+isBackendOnClient() : boolean
+getLogger() : IDAELogger
```

```
<<interface>>
IDAEAcquisitionState
+INFO_JSON_PROP_NAME : String = "info"
+RUNTIME_INFO_JSON_PROP_NAME : String = "runtime info"
+ENV_PROPS_NAME : String = "envProps"
+getInfo() : String
+getRuntimeInfo() : String
+getEnvProps() : Map<String, String>
```

```
DAException
-serialVersionUID : long = 1L
+DAException(localizedMessage : String, cause : Throwable
+DAException(localizedMessage : String
+DAException(cause : Throwable)
```

6.4.3 Hierarchy for Data Access Extension SDK

Hierarchy For Package com.sap.bi.da.extension.sdk

Table 10:

Class Hierarchy
● java.lang.Object ○ com.sap.bi.da.extension.sdk.DAEConstants ○ com.sap.bi.da.extension.sdk.DAEConstants.Metadata ○ com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys ○ java.lang.Throwable (implements java.io.Serializable) ○ java.lang.Exception ○ com.sap.bi.da.extension.sdk.DAException

Table 11:

Interface Hierarchy
• com.sap.bi.da.extension.sdk.IDAEAcquisitionJobContext • com.sap.bi.da.extension.sdk.IDAEAcquisitionState • com.sap.bi.da.extension.sdk.IDAEEnvironment • com.sap.bi.da.extension.sdk.IDAEJob<T> ○ com.sap.bi.da.extension.sdk.IDAEClientRequestJob ○ com.sap.bi.da.extension.sdk.IDAEDataAcquisitionJob ○ com.sap.bi.da.extension.sdk.IDAEMetadataAcquisitionJob • com.sap.bi.da.extension.sdk.IDAEProgress • com.sap.bi.da.extension.sdk.IDAExtension

Table 12: Enum Hierarchy

Enum Hierarchy
• java.lang.Object ○ java.lang.Enum<E> (implements java.lang.Comparable<T>, java.io.Serializable) ○ com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType ○ com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction ○ com.sap.bi.da.extension.sdk.DAEWorkflow

6.4.4 Class DAEConstants

Class Reference: com.sap.bi.da.extension.sdk Class DAEConstants

java.lang.Object

- `com.sap.bi.da.extension.sdk.DAEConstants`

```
public final class DAEConstants
extends java.lang.Object
```

Table 13: Nested Class Summary

Modifier and Type	Class and Description
`static class`	`DAEConstants.Metadata`

Table 14: Constructor Summary

Constructor and Description
`DAEConstants()`

Table 15: Method Summary

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Table 16: Constructor Detail

DAEConstants
`public DAEConstants()`

6.4.5 Class DAEConstants.Metadata

Class Reference: com.sap.bi.da.extension.sdk Class DAEConstants

java.lang.Object

- com.sap.bi.da.extension.sdk.DAEConstants.Metadata

Enclosing class:

- DAEConstants

```
public static final class DAEConstants.Metadata
extends Object
```

Table 17: Nested Class Summary

Modifier and Type	Class and Description
`static class`	`DAEConstants.Metadata.AggregationFunction`
`static class`	`DAEConstants.Metadata.DataType`
`static class`	`DAEConstants.Metadata.Keys`

Table 18: Field Summary

Modifier and Type	Field and Description
`static class`	`VERSION`

Table 19: Constructor Summary

Constructor and Description
`DAEConstants.Metadata()`

Table 20: Method Summary

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Table 21: Field Detail

VERSION
`public static final String VERSION` See Also: • Constant Field Values

Table 22: Constructor Detail

DAEConstants.Metadata
`public DAEConstants.Metadata()`

6.4.6 Class DAEConstants.Metadata.Keys

Class Reference: com.sap.bi.da.extension.sdk

java.lang.Object

- com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys

Enclosing class:

- DAEConstants.Metadata

```
public static final class DAEConstants.Metadata.Keys
extends Object
```

Table 23: Field Summary

Modifier and Type	Field and Description
static String	`AGGREGATION_FUNCTION`
static String	`ANALYTICAL_TYPE`
static String	`COLUMNS`
static String	`HIERARCHIES`
static String	`ID`
static String	`LENGTH`
static String	`LEVELS`
static String	`NAME`
static String	`TYPE`
static String	`VERSION`

Table 24: Constructor Summary

Constructor and Description	Detail
`DAEConstants.Metadata.Keys()`	`public DAEConstants.Metadata.Keys()`

Table 25: Method Summary

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Table 26: Field Detail - See Also: Constant Field Values

Name	Usage
VERSION	public static final String VERSION
COLUMNS	public static final String COLUMNS
HIERARCHIES	public static final String HIERARCHIES
LEVELS	public static final String LEVELS
TYPE	public static final String TYPE
NAME	public static final String NAME
LENGTH	public static final String LENGTH
ID	public static final String ID
ANALYTICAL_TYPE	public static final String ANALYTICAL_TYPE
AGGREGATION_FUNCTION	public static final String AGGREGATION_FUNCTION

6.4.7 Class DAException

Class reference for DAException from com.sap.bi.da.extension.sdk

java.lang.Object

- java.lang.Throwable
    - java.lang.Exception
        - com.sap.bi.da.extension.sdk.DAException

All Implemented Interfaces:

- Serializable

```
public class DAException
extends Exception
```

Exception to be thrown when a problem is encountered during the execute() method of a DAE Job. Note that this message will be presented to the user, so should be localized using the locale provided by the DAE framework.

See Also:

- Serialized Form

Table 27: Constructor Summary and Detail

Constructor and Description	Detail
DAException(String localizedMessage)	public DAException(String localizedMessage, Throwable cause) Parameters: • localizedMessage - the localized message to be presented to the user • cause - the parent Throwable if any

Constructor and Description	Detail
DAException(String localizedMessage, Throwable cause)	public DAException(String localizedMessage) Parameters: - localizedMessage - the localized message to be presented to the user
DAException(Throwable cause)	public DAException(Throwable cause)

Table 28: Method Summary

Method...	Method List
Methods inherited from class java.lang.Throwable	addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString
Methods inherited from class java.lang.Object	equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

6.4.8 Enum DAEConstants.Metadata.AggregationFunction

Class Reference: com.sap.bi.da.extension.sdk

java.lang.Object

- java.lang.Enum<DAEConstants.Metadata.AggregationFunction>
  - com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction

All Implemented Interfaces:

- Serializable, Comparable<DAEConstants.Metadata.AggregationFunction>

Enclosing class:

- DAEConstants.Metadata

```
public static enum DAEConstants.Metadata.AggregationFunction
extends Enum<DAEConstants.Metadata.AggregationFunction>
```

Table 29: Enum Contants

Enum Constant	Detail
AVERAGE	public static final DAEConstants.Metadata.AggregationFunction AVERAGE
COUNT_ALL	public static final DAEConstants.Metadata.AggregationFunction COUNT_ALL
COUNT_DISTINCT	public static final DAEConstants.Metadata.AggregationFunction COUNT_DISTINCT
MAX	public static final DAEConstants.Metadata.AggregationFunction MAX
MIN	public static final DAEConstants.Metadata.AggregationFunction MIN
NONE	public static final DAEConstants.Metadata.AggregationFunction NONE

Enum Constant	Detail
`SUM`	`public static final DAEConstants.Metadata.AggregationFunction SUM`

Table 30: Method Summary

Modifier and Type	
`boolean`	`compareToNoCase(String text)`
`String`	`valueOf(String name)` Returns the enum constant of this type with the specified name.
`static DAEConstants.Metadata.AggregationFunction`	`values()`
`static DAEConstants.Metadata.AggregationFunction[]`	Returns an array containing the constants of this enum type, in the order they are declared.

Table 31: Methods inherited from class java.lang.Enum

Methods inherited from class java.lang.Enum
`compareTo, equals, getDeclaringClass, hashCode, name, ordinal, valueOf`

Table 32: Methods inherited from class java.lang.Object

Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, wait, wait, wait`

Method Detail

Table 33: values

Values
`public static DAEConstants.Metadata.AggregationFunction[] values()`

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (DAEConstants.Metadata.AggregationFunction c :
DAEConstants.Metadata.AggregationFunction.values())
    System.out.println(c);
```

Returns:

- an array containing the constants of this enum type, in the order they are declared

Table 34: valueOf

valueOf
`public static DAEConstants.Metadata.AggregationFunction valueOf(String name)` Returns the enum constant of this type with the specified name. The string must match exactly an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.) Parameters: • `name` - the name of the enum constant to be returned. Returns: • the enum constant with the specified name Throws: • `IllegalArgumentException` - if this enum type has no constant with the specified name • `NullPointerException` - if the argument is null

Table 35: toString

toString
`public String toString()` Overrides: • `toString in class Enum<DAEConstants.Metadata.AggregationFunction>`

6.4.8.1 DAEConstants.Metadata.AggregationFunction Enum Reference

Metadata Aggregation Functions

Table 36: Public Member Functions

Public Member Functions	
String	toString ()
boolean	compareToNoCase (String text)

Table 37: Public Member Functions

Public Attributes
NONE =("none")
SUM =("sum")
MIN =("min")
MAX =("max")
AVERAGE =("average")
COUNT_ALL =("count_all")
COUNT_DISTINCT =("count_distinct")

Table 38: Public Member Functions

Member Functions
boolean compareToNoCase (String text)
String Function.toString ( )

Table 39: Member Data

DAEConstants.Metadata.AggregationFunction.
NONE
SUM
MIN
MAX
AVERAGE
COUNT_ALL
COUNT_DISTINCT

6.4.9 Enum DAEConstants.Metadata.DataType

Class Reference: com.sap.bi.da.extension.sdk

java.lang.Object

- java.lang.Enum<DAEConstants.Metadata.DataType>
  - com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType

All Implemented Interfaces:

- Serializable, Comparable<DAEConstants.Metadata.DataType>

Enclosing class:

- DAEConstants.Metadata

```
public static enum DAEConstants.Metadata.DataType
extends Enum<DAEConstants.Metadata.DataType>
```

Table 40: Enum Constant Summary

Enum Constant	
BIGINTEGER	public static final DAEConstants.Metadata.DataType BIGINTEGER
BOOLEAN	public static final DAEConstants.Metadata.DataType BOOLEAN
DATE	public static final DAEConstants.Metadata.DataType DATE
DATETIME	public static final DAEConstants.Metadata.DataType DATETIME
INTEGER	public static final DAEConstants.Metadata.DataType INTEGER
NUMBER	public static final DAEConstants.Metadata.DataType NUMBER

Enum Constant	
STRING	public static final DAEConstants.Metadata.DataType STRING
TIME	public static final DAEConstants.Metadata.DataType TIME

Table 41: Method Summary

Modifier and Type	Method and Description
boolean	compareToNoCase(String text)
String	toString()
static DAEConstants.Metadata.DataType	valueOf(String name) Returns the enum constant of this type with the specified name.
static DAEConstants.Metadata.DataType[]	values() Returns an array containing the constants of this enum type, in the order they are declared.

Table 42: Methods inherited from class java.lang.Enum

Methods inherited from class java.lang.Enum
compareTo, equals, getDeclaringClass, hashCode, name, ordinal, valueOf

Table 43: Methods inherited from class java.lang.Object

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Method Detail

Table 44: values

values
public static DAEConstants.Metadata.DataType[] values()

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (DAEConstants.Metadata.DataType c : DAEConstants.Metadata.DataType.values())
    System.out.println(c);
```

Returns:

- an array containing the constants of this enum type, in the order they are declared

Table 45: valueOf

valueOf
`public static DAEConstants.Metadata.DataType valueOf(String name)`
Returns the enum constant of this type with the specified name. The string must match exactly an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)
Parameters: • `name` - the name of the enum constant to be returned.
Returns: • the enum constant with the specified name
Throws: • `IllegalArgumentException` - if this enum type has no constant with the specified name • `NullPointerException` - if the argument is null

Table 46: toString

toString
`public String toString()`
Overrides: • `toString in class Enum<DAEConstants.Metadata.DataType>`

Table 47: compareToNoCase

compareToNoCase
`public boolean compareToNoCase(String text)`

6.4.9.1 DAEConstants.Metadata.DataType Enum Reference

DAEConstants.Metadata.DataType Enums

Table 48: Public Member Functions

Type	Name and Description
string	toString ()
Boolean	compareToNoCase (String text)

Table 49: Public Attributes

Name and Description
BOOLEAN =("boolean")
INTEGER =("integer")
BIGINTEGER =("biginteger")
NUMBER =("number")

Name and Description
DATE =("date")
DATETIME =("datetime")
TIME =("time")
STRING =("string")

Table 50:

Member Functions
DAEConstants.Metadata.DataType.compareToNoCase (String text)
DAEConstants.Metadata.DataType.toString ( )

Table 51:

Member Data
DAEConstants.Metadata.DataType.BIGINTEGER =("biginteger")
DAEConstants.Metadata.DataType.BOOLEAN =("boolean")
DAEConstants.Metadata.DataType.DATE =("date")
DAEConstants.Metadata.DataType.DATETIME =("datetime")
DAEConstants.Metadata.DataType.INTEGER =("integer")
DAEConstants.Metadata.DataType.NUMBER =("number")
DAEConstants.Metadata.DataType.STRING =("string")
DAEConstants.Metadata.DataType.TIME =("time")

6.4.10 Enum DAEWorkflow

Enum reference for DAEWorkflow com.sap.bi.da.extension.sdk

java.lang.Object

- com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys

Enclosing class:

- DAEConstants.Metadata

```
public static final class DAEConstants.Metadata.Keys
extends Object
```

Table 52: Enum Summary

Enum Constant	Enum Constant Detail
CREATE	`public static final DAEWorkflow CREATE`
EDIT	`public static final DAEWorkflow EDIT`

Enum Constant	Enum Constant Detail
REFRESH	`public static final DAEWorkflow REFRESH`

Table 53: Method Summary

Modifier and Type	Method and Description
`String`	toString()
static `DAEWorkflow`	toString() Returns the enum constant of this type with the specified name.
static `DAEWorkflow[]`	values() Returns an array containing the constants of this enum type, in the order they are declared.

Table 54: Methods inherited from:

Name	Summary List
class java.lang.Enum	compareTo, equals, getDeclaringClass, hashCode, name, ordinal, valueOf
class java.lang.Object	getClass, notify, notifyAll, wait, wait, wait

Table 55: Method Detail

Name	

values	`public static DAEWorkflow[] values()`
	Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:
	`for (DAEWorkflow c : DAEWorkflow.values())`
	`    System.out.println(c);`
	Returns:
	- an array containing the constants of this enum type, in the order they are declared

valueOf	`public static DAEWorkflow valueOf(String name)` Returns the enum constant of this type with the specified name. The string must match exactly an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.) **Parameters:** - `name` - the name of the enum constant to be returned. **Returns**: - the enum constant with the specified name **Throws:** - `IllegalArgumentException` - if this enum type has no constant with the specified name - `NullPointerException` - if the argument is null

Name	
toString	`public String toString()` **Overrides:** • `toString` in class `Enum<DAEWorkflow>`

6.4.10.1 DAEWorkflow Enum Reference

DAEWorkflow Enum Reference

Table 56: Public Member Functions

Type	Name and Description
	DAEWorkflow (String text)
String	toString ()

Table 57:

Public Attributes
CREATE =("create")
EDIT =("edit")
REFRESH =("refresh")

Table 58:

Constructor & Destructor
DAEWorkflow (String text)

Table 59:

Member Functions
DAEWorkflow.toString ( )

Table 60:

Member Data
DAEWorkflow.CREATE =("create")
DAEWorkflow.EDIT =("edit")
DAEWorkflow.REFRESH =("refresh")

6.4.10.2 DAEWorkflow Enum Reference Enum Reference

DAEWorkflow Enum Reference

Table 61: Public Member Functions

Public Member Functions	
	DAEWorkflow (String text)
String	toString ()

Table 62: Public Member Functions

Public Attributes
CREATE =("create")
EDIT =("edit")
REFRESH =("refresh")

Table 63: Public Member Functions

Constructor & Destructor
DAEWorkflow.toString (String text)

6.4.11 Interface IDAEAcquisitionJobContext

Interface IDAEAcquisitionJobContext

6.4.12 Interface IDAEAquisitionState

IDAEAcquisitionState Interface description for com.sap.bi.da.extension.sdk

```
public interface IDAEAcquisitionState
```

Table 64: Field Summary

Modifier and Type	Field and Description (See Also: Constant Field Values)

`static String`	`ENV_PROPS_NAME`
	`static final String ENV_PROPS_NAME`
	This property holds information about the current runtime environment, such as productLocale and preferredViewingLocale

Modifier and Type	Field and Description (See Also: Constant Field Values)

`static String`	`INFO_JSON_PROP_NAME`
	`static final String INFO_JSON_PROP_NAME`
	The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend.

`static String`	`RUNTIME_INFO_JSON_PROP_NAME`
	`static final String RUNTIME_INFO_JSON_PROP_NAME`
	The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend. Use this name for information that you don't wish to be persist across restart. For example, authentication information or acquired data.

Table 65: Method Summary

Modifier and Type	Method and Description
`Map<String,String>`	`getEnvProps()` Owned by the framework. Returns:
`String`	`getInfo()` Owned by the extension. **Returns**: • `info`
`String`	`getRuntimeInfo()` Owned by the extension. Returns: • `runtimeInfo`

6.4.12.1 AcquisitionState - JSON Object

The structure of the JSON object is as follows:

```
{
    "info" : "",
    "runtimeInfo" : "",
    "envProps" : {}
}
```

See documentation for the interface IDAEAcquisitionState as that and this object are identical.

AcquisitionState will be passed to your UI through the UI workflows:

```
    doCreateWorkflow
    doRefreshWorkflow
    doEditWorkflow
```

The `doCreateWorkflow` should also return a `suggestedDatasetName` string.

It is your UI's responsibility to return a `AcquisitionState` object after each UI workflow. For example:

- Returning a query which represents your acquisition. This query is passed to your java extension at acquisition time.
- If you need some kind of authentication to perform your acquisition, returning valid credentials in the `runtimeInfo` field. Lumira will pass credentials that you supply, immediately to your java extension during acquisition time.

6.4.12.2 acquisitionState properties

The supported properties of `acquisitionState`.

Table 66:

Propties of acquisitionState	
acquisitionState.envProps.productLocale	The locale of the Lumira product. You should use this to make sure your extension UI also displays in the same language as the rest of the product.
acquisitionState.envProps.preferredViewingLocale	The locale that should be applied to your data. This may be required during the data acquisition.

6.4.13 Interface IDAEClientRequestJob

IDAEClientRequestJob Interface description.

All Superinterfaces:

- `IDAEJob<File>`

```
public interface IDAEDataAcquisitionJob
extends IDAEJob<File>
```

Table 67: Method Summary

Methods inherited from interface com.sap.bi.da.extension.sdk.IDAEJob
`cancel, cleanup, execute`

6.4.14 Interface IDAEEnvironment

IDAEEnvironmntt Interface description.

```
public interface IDAEEnvironment
```

Table 68: Method Summary

Modifier and Type	Method and Description
`boolean`	`allowTrustedDAExtensions()` `True` if the the environment allows trusted DA extensions, `false` otherwise. For example, on desktop, the returned value will be affected by the application keycode.
`java.lang.String`	`getEnvironmentVersion(String environmentId)` Get version string for some particular aspect of the enclosing environment.
`boolean`	`is32BitEnvironment()` True if the environment is 32-bit, false if not.
`boolean`	`isBackendOnClient()` `true` if the client and the back end are on the same machine (for example, *Desktop*). `false` if not (for example, *LIMA* or *Cloud*)

6.4.15 Interface IDAEJob

IDAEJob<T> Interface description.

Type Parameters:

- `T` – the type of the result returned by this job.

All Known Subinterfaces:

- `IDAEClientRequestJob`, `IDAEDataAcquisitionJob`, `IDAEMetadataAcquisitionJob`

```
public interface IDAEJob<T>
```

This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework

Table 69: Method Summary

Modifier and Type	Method and Description
`void`	`cancel()` Signals to this job to cancel the current operation as soon as possible.

Modifier and Type	Method and Description
void	`cleanup()` Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc.
T	`execute(IDAEProgress callback)`

Table 70: Method Detail

Name	
execute	`T execute(IDAEProgress callback) throws DAException` **Returns:** • the result of this job when done. This call will block until a result is returned or the job terminates unsuccessfully. **Throws:** • `DAException`
cancel	`void cancel()` Signals to this job to cancel the current operation as soon as possible. It is highly recommended that your extension support cancel for usability reasons. Should be non-blocking.
cleanup	`void cleanup()` Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc. Lumira will always call this method after - execute () is called and the result is no longer needed - any API throwing a DAException - cancel () was called

6.4.15.1 IDAEJob< T > Interface Reference

This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework

Table 71: Public Member Functions

Type	Name and Description
T	execute (IDAEProgress callback) throws DAException
void	cancel () Signals to this job to cancel the current operation as soon as possible. It is highly recommended that your extension support cancel for usability reasons. Should be non-blocking.

Type	Name and Description
void	cleanup () Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc. Lumira will always call this method after: ● execute () is called and the result is no longer needed ● any API throwing a DAException ● cancel () was called
returns	IDAEJob< T >.execute (IDAEProgress callback) throws DAException The result of this job when done. This call will block until a result is returned or the job terminates unsuccessfully.

6.4.16 Interface IDAELogger

This interface represents the logger passed to the extension.

The assembly (*Desktop* and *Edge* for example) should create its own implementation of this so as to unify logging with the extension. "Trace" calls are intended to be seen by the developer, whereas "Log" calls are actionable items to be seen by an administrator.

```
public interface IDAELogger
```

Table 72: Method List for IDAELogger

Method	Method Detail
traceDebug	
traceDebug	void traceDebug(java.lang.Object msg)
traceInfo	void traceInfo(java.lang.Object msg)
traceError	void traceError(java.lang.Object msg)
traceDebug	void traceDebug(java.lang.Throwable t, java.lang.Object msg)
traceInfo	void traceInfo(java.lang.Throwable t, java.lang.Object msg)
traceError	void traceError(java.lang.Throwable t, java.lang.Object msg)
traceDebug	void traceDebug(java.lang.String format, java.lang.Object... args)
traceInfo	void traceInfo(java.lang.String format, java.lang.Object... args)
traceError	void traceError(java.lang.String format, java.lang.Object... args)

Method	Method Detail
`traceDebug`	void traceDebug(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`traceInfo`	void traceInfo(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`traceError`	void traceError(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`traceDebug`	void traceDebug(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)
`traceInfo`	void traceInfo(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)
`traceError`	void traceError(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)
logInfo	
`logInfo`	void logInfo(java.lang.Object msg)
`traceInfo`	void logWarning(java.lang.Object msg)
`traceError`	void logError(java.lang.Object msg)
`logInfo`	void logInfo(java.lang.Throwable t, java.lang.Object msg)
`traceInfo`	void logWarning(java.lang.Throwable t, java.lang.Object msg)
`traceError`	void logError(java.lang.Throwable t, java.lang.Object msg)
`logInfo`	void logInfo(java.lang.String format, java.lang.Object... args)
`traceInfo`	void logWarning(java.lang.String format, java.lang.Object... args)
`traceError`	void logError(java.lang.String format, java.lang.Object... args)
`logInfo`	void logInfo(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`traceInfo`	void logWarning(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`traceError`	void logError(java.lang.Throwable t, java.lang.String format, java.lang.Object... args)
`logInfo`	void logInfo(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)
`traceInfo`	void logWarning(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)
`traceError`	void logError(java.lang.Class<?> c, java.lang.String format, java.lang.Object... args)

6.4.17 Interface IDAEMetadataAcquisitionJob

IDAEMetadaAcquisitionJob description

All Superinterfaces:

- IDAEJob<String>

```
public interface IDAEMetadataAcquisitionJob
extends IDAEJob<String>
```

Table 73: Method Summary

Methods inherited from interface com.sap.bi.da.extension.sdk.IDAEJob
`cancel, cleanup, execute`

6.4.18 Interface IDAEProgress

IDAEProgress Interface description.

```
public interface IDAEProgress
```

Callback interface for clients to be notified as job states are updated.

Table 74: Method Summary

Modifier and Type	Class and Description
`void`	`percentComplete(float percentComplete)` Notification that a job has completed a certain percentage of its execution..

Table 75: Method Detail

Name	Description
`percentComplete`	`void percentComplete(float percentComplete)` Notification that a job has completed a certain percentage of its execution. Parameters: • `percentComplete -`

6.4.19 Interface IDAExtension

IDAExtension Interface description.

```
public interface IDAExtension
```

This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.

Version:

- 1.0

Table 76: Method Summary

Modifier and Type	Class and Description
`IDAEClientRequestJob`	`getClientRequestJob(String request)` Asks your extension to handle requests that you sent from your client extension code.
`IDAEAcquisitionJobContext`	`getDataAcquisitionJobContext(IDAEAcquisitionState acquisitionState)` Get a context for data acquisition jobs.
`Set<DAEWorkflow>`	`getEnabledWorkflows(IDAEEnvironment environment, IDAEAcquisitionState acquisitionState)` Get the supported workflows in the provided environment (and optionally, with the provided acquisition state).
`String`	`getExtensionId()` Returns the extension id that uniquely identifies your extension.

Table 77: Method Detail

Name	Description
`getExtensionId`	`String getExtensionId()` Returns the extension id that uniquely identifies your extension. it is recommended to use some kind of hierarchical naming convention like com.mycompany.subcategory.subcategory2....name This id should be identical to the da-extension-id in the AcquisitionState that your UI extension returns to Lumira. **Returns**: • a string representing your extension id.
`getDataAcquisitionJobContext`	`IDAEAcquisitionJobContext getDataAcquisitionJobContext(IDAEAcquisitionState acquisitionState)` Get a context for data acquisition jobs. **Parameters**: • `acquisitionState` - - the acquisition state. **Returns**: • a container object to serve jobs which can provide DA info (metadata, data) based on the point in time at which this context was created. **Throws**: • `DAException`

Name	Description
`getClientRequestJob`	IDAEClientRequestJob getClientRequestJob(String request) Asks your extension to handle requests that you sent from your client extension code. If your client extension code made a request through the javascript API handleDAExtensionRequest, the request is passed through here. **Parameters**: ● `request` - **Returns**: ● a job to handle the client request
`getEnabledWorkflows`	`Set<DAEWorkflow> getEnabledWorkflows(IDAEEnvironment environment, IDAEAcquisitionState acquisitionState)` Get the supported workflows in the provided environment (and optionally, with the provided acquisition state). Typically, Lumira will ask your extension for these capabilities and the extension is expected to answer them based on the environment argument. Scenarios where you might disable your extension: 1) if your extension needs to run a 64-bit process and Lumira tells you this is a 32-bit environment 2) if you want your extension to be available only on certain versions of Lumira product Note that if acquisitionState is null and the returned list is empty, this extension will be disabled. **Parameters**: ● `information` - about the environment in which the DA extension is running ● `acquisitionState` - the applicable acquisitionState, or null to determine applicability of workflows based on environment only. This argument is used to determine refreshability and editability of an acquisition. This argument should be ignored for create-ability (i.e. just use the environment argument) **Returns**: ● the list of supported workflows.

6.4.19.1 IDAExtension

This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.

Version 1.0 (Lumira 1.23)

Table 78: Classes

Type	Name and description
class	DAEConstants - VERSION, COLUMNS, HIERARCHIES, LEVELS, TYPE, NAME, LENGTH, ID ANALYTICAL_TYPE, AGGREGATION_FUNCTION - BOOLEAN, INTEGER, BIGINTEGER, NUMBER, DATE, DATETIME, TIME, STRING
enum	DAEWorkflow
class	DAException Exception to be thrown when a problem is encountered during the execute() method of a DAE Job. Note that this message will be presented to the user, so should be localized using the locale provided by the DAE framework.
interface	DAEAcquisitionJobContext The context for a data acquisition. Jobs returned by this context are expected to return results which are consistent. e.g. always return the same metadata, data is always consistent with the metadata.
interface	IDAEAcquisitionState
interface	IDAEClientRequestJob
interface	IDAEDataAcquisitionJob
interface	IDAEEnvironment
interface	IDAEJob< T > This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework. Parameters: <T> the type of the result returned by this job.
interface	IDAEMetadataAcquisitionJob
interface	IDAEProgress Callback interface for clients to be notified as job states are updated.
interface	IDAExtension This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework. You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.

6.5 Message Logging and Error Handling

6.5.1 Error Handling

Java and Javascript error handling.

Error handling - Java

- From your job.execute() method, throw a DAException with your localized message (and optionally the parent throwable).
- This will be caught by the DA Framework and bubbled up to the user in the standard message dialog.

Error handling - Javascript

- No general error display framework will be supplied.
- Any errors detected within the flow of the client-side code should be handled in the extension client UI.

6.5.2 Message Logging

Message logging for auditing purposes, using Java or Javascript.

Java

- Use SAPLogger by importing com.sap.hilo.util.logging.SAPLogger; and so on.

Javascript

- Use the window.logger:
  - Source

    ```
    h5v2/sap.vi.desktoplog/sapLogger.js
    ```

  - Example

    ```
    window.logger.error("sap.vi.desktop.datasource.extension.ExtensionDataSourceEn
    try",
    errorThrown);
    ```

6.6 handleDAExtensionRequest Javascript

Backend plugin call from Javascript code that calls...

The backend extension point is com.sap.bi.da.extension.backend

```
<plugin>
  <extension point="com.sap.bi.da.extension.backend">
     // Java class that implements IDAExtension interface.
     <da_extension class="com.mycompany.myextension.MyExtensionClass">
     </da_extension>
  </extension>
</plugin>
```

6.7 Query Entries

Queries are to be entered as a continuous string, rather that separated over several lines.

The following incorrect example shows a typical SQL query spread out over several lines:

```
query;Select *
From Table
Where x=y
;true
```

Instead, use a single line for the entire argument as follows:

```
query;Select * From Table Where x=y ;true
```

6.8 CSV Data Format Reference

Comma Separated Value (CSV) specification for the SAP Lumira Data Acquisition Extension Framework.

SAP Lumira expects the CSV format with the following restrictions.:

Table 79: CSV Format Accepted by SAP Lumira

Item	Description and example
No/NULL Values	To specify no value, insert an unquoted NUL character '\u0000' (unicode) or '0x00' (hex) or the NUL character in ASCII. Note Do not use the literal, NULL. Example - "Jill","Fox", ,"33" (where , , contains the NUL character.) Note It is important to never quote the NULL values.
Quoting	Use the quotation mark (double-quote ") character for all data supplied to SAP Lumira. Use double quotes when including quote characters. Example: - "Jill","Fox","jfox",33,"She said ""howdy stranger!"" "
Field Delimiter	Use only the comma (`,`) to separate fields.
Row Delimiter	Use Windows carriage-return: (`\r\n`)
Headers	Column names are described in the metadata file only. See: Metada File Structure. Note SAP Lumira does not recognize the use of the first row as column names.
Trimming	Trim your data of leading or trailing spaces if needed.

Item	Description and example
Encoding	The only recognized encoding is UTF-8. Note Remove any byte-order mark (BOM) character at the beginning of CSV data before supplying it to Lumira.

Table 80: Supported data types

Type	
BOOLEAN	This ia a `1` for true, or `0` (number zero) for false.
STRING	The metadata must describe the string as follows:: - The encoding / character set ○ Example: "UTF-8" ○ Specify using the global property `string-encoding`
NUMBER	Limitations: - The decimal precision is 15 significant digits (for example, 1234.123456789012345) Note Use on currency values with repetative calculations is not recommended. - Omit the comma for the thousands separator (for example, use 1000, not 1,000) - Use a leading zero and a period to represent a decimal point, not a comma (for example, 0.333, not .333; 4.55, not 4,55) - Use negative sign (-). Do not use a positive sign (+) - Scientific notation is not supported (for example, 3.334E+15)
INTEGER	For 32-bit signed integer values. Values range from –2,147,483,648 through +2,147,483,647. Limitations: (the same as for `NUMBER`)
DATE	Date format is YYYY-MM-DD (for example, 1985-12-25)

6.9 Metadata File Structure

The SAP Lumira Data Access Extension metadata format. This file describes each of the columns in a data file it is to be paired with.

Top level header description format

Detailed definitions for "columns" and "hierarchies" appear separately.

```
{
    "version": "1.0",
    "columns":              // required
       <columnInfo>,
    "hierarchies":          // optional
        <hierarchyInfo>
}
```

"columns" description format

```
"columns":
[
    {
        "name": "<NAME>",
        "id": "<ID>",
        "type": "<TYPE>",
        "description": "<DESCRIPTION>",    // not yet used
        "analyticalType": "<TYPE>",
        "aggregationFunction" : "<FUNCTION>",
    }
    ,...
]
```

Example Header and one column described.

```
{
    "version": "1.0",
    "columns": [
        {
            "name": "Integer (32-bit signed)",
            "id": "id0",
            "type": "Integer",
            "analyticalType": "dimension"
        }
    ]
}
```

Table 81: The "columns" parameter descriptions

Parameter Name	Description
name	This is the display name for the column.
id	This is a unique identifier for this column. This should always map uniquely to a column in your data source. This is the identifier used in hierarchy definitions that will refer to this column.
type	The data type of this column of supported values as follows: • boolean • integer • biginteger • number • date • datetime • time • string
description	This is a description for the column name. Note This an optional field that is currently not read, as of version SAP Lumira 1.23. .

<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>analyticalType</td><td>This is choice is one of the following:
<ul><li>dimension</li><li>measure (see aggregationFunction)</li></ul></td></tr>
<tr><td>aggregationFunction</td><td>This is an optional field used when the analyticalType is "measure".
The default values for numeric or non-numeric columns are as follows:
<ul><li>sum (for data type not equal to string)</li><li>count_all (for data type = string)</li></ul>
The available aggregation functions depend on the data type are as follows:
Table 82:
<table>
<tr><th>Data type</th><th>boolean</th><th>bigInteger, integer, number</th><th>string</th><th>date</th><th>Description</th></tr>
<tr><td>AVERAGE</td><td></td><td>x</td><td></td><td></td><td>This is the average (SUM / COUNT_ALL).</td></tr>
<tr><td>COUNT_ALL</td><td>x</td><td>x</td><td>x</td><td>x</td><td>Count of all items in the column.</td></tr>
<tr><td>COUNT_DISTINCT</td><td>x</td><td>x</td><td>x</td><td>x</td><td>Count of unique occurences in the column.</td></tr>
<tr><td>MAX</td><td>x</td><td>x</td><td></td><td>x</td><td>The highest value found in the column.</td></tr>
<tr><td>MIN</td><td>x</td><td>x</td><td></td><td>x</td><td>The lowest value found in the column.</td></tr>
<tr><td>NONE</td><td></td><td>x</td><td></td><td></td><td>No entry.</td></tr>
<tr><td>SUM</td><td></td><td>x</td><td></td><td></td><td>A total of all values in the column.</td></tr>
</table></td></tr>
</table>

"hierarchies"

i Note

These are level-based hierarchies only.

```
"hierarchies":
[
    {
        "name":         "<NAME>",
        "description": "<DESCRIPTION>",
        "levels":
          [ <colId1>, <colId2>, <colId3>, ... ]
    }
]
```

Table 83: The "hierarchy" parameter descriptions

Parameter Name	Description
`name`	A unique display name for this hierarchy. Example: `"name": "Country",`
`description`	This is a description for the hierarchy name. Note This an optional field that is currently not read, as of version SAP Lumira 1.23. Example: `"description": "This is the three-digit A3 (UN) country code, ie. DEU = Germany.",`
`levels`	A list of column IDs that make up the hierarchy, ordered by the outermost (parent) to innermost (child). Example: `"levels":[<Region>, <Country>, <City>]`

6.10 Index

Index of classes, constructors, interfaces, methods, variables, enums...

Table 84: A

Name - "A"	Description
AGGREGATION_FUNCTION	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
allowTrustedDAExtensions()	Method in interface com.sap.bi.da.extension.sdk.IDAEEnvironment True if the the environment allows trusted DA extensions, false otherwise.
ANALYTICAL_TYPE	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys

Table 85: C

Name - "C"	Description
cancel()	Method in interface com.sap.bi.da.extension.sdk.IDAEJob Signals to this job to cancel the current operation as soon as possible.
cleanup()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionJobContext Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc.

Name - "C"	Description
cleanup()	Method in interface com.sap.bi.da.extension.sdk.IDAEJob Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc.
COLUMNS	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
com.sap.bi.da.extension.sdk	package com.sap.bi.da.extension.sdk
compareToNoCase(String)	Method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction
compareToNoCase(String)	Method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType

Table 86: D

Name - "D"	Description
DAEConstants	Class in com.sap.bi.da.extension.sdk
DAEConstants()	Constructor for class com.sap.bi.da.extension.sdk.DAEConstants
DAEConstants.Metadata	Constructor for class com.sap.bi.da.extension.sdk.DAEConstants
DAEConstants.Metadata	Class in com.sap.bi.da.extension.sdk
DAEConstants.Metadata()	Constructor for class com.sap.bi.da.extension.sdk.DAEConstants.Metadata
DAEConstants.Metadata.AggregationFunction	Enum in com.sap.bi.da.extension.sdk
DAEConstants.Metadata.DataType	Enum in com.sap.bi.da.extension.sdk
DAEConstants.Metadata.Keys	Class in com.sap.bi.da.extension.sdk
DAEConstants.Metadata.Keys()	Constructor for class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
DAEWorkflow	Enum in com.sap.bi.da.extension.sdk
DAException	Enum in com.sap.bi.da.extension.sdk Exception to be thrown when a problem is encountered during the execute() method of a DAE Job.
DAException(String, Throwable)	Constructor for exception com.sap.bi.da.extension.sdk.DAException
DAException(String)	Constructor for exception com.sap.bi.da.extension.sdk.DAException
DAException(Throwable)	Constructor for exception com.sap.bi.da.extension.sdk.DAException
ENV_PROPS_NAME	Static variable in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState This property holds information about the current runtime environment, such as productLocale and preferredViewingLocale
execute(IDAEProgress)	Method in interface com.sap.bi.da.extension.sdk.IDAEJob

Table 87: G

getClientRequestJob(String)	Method in interface com.sap.bi.da.extension.sdk.IDAExtension Asks your extension to handle requests that you sent from your client extension code.
getDataAcquisitionJob()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionJobContext Get a job to return the data corresponding to the point in time at which this object was created.
getDataAcquisitionJobContext(IDAEAcquisitionState)	Method in interface com.sap.bi.da.extension.sdk.IDAExtension Get a context for data acquisition jobs.
getEnabledWorkflows(IDAEEnvironment, IDAEAcquisitionState)	Method in interface com.sap.bi.da.extension.sdk.IDAExtension Get the supported workflows in the provided environment (and optionally, with the provided acquisition state).
getEnvironmentVersion(String)	Method in interface com.sap.bi.da.extension.sdk.IDAEEnvironment Get version string for some particular aspect of the enclosing environment.
getEnvProps()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState Owned by the framework.
getExtensionId()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState Returns the extension id that uniquely identifies your extension.
getInfo()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState Owned by the extension.
getMetadataAcquisitionJob()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionJobContext Get a job to return the metadata corresponding to the point in time at which this object was created.
getRuntimeInfo()	Method in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState Owned by the extension.
HIERARCHIES	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys

Table 88: I

ID	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
IDAEAcquisitionJobContext	Interface in com.sap.bi.da.extension.sdk The context for a data acquisition.
IDAEAcquisitionState	Interface in com.sap.bi.da.extension.sdk

ID	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
IDAEClientRequestJob	Interface in com.sap.bi.da.extension.sdk
IDAEDataAcquisitionJob	Interface in com.sap.bi.da.extension.sdk
IDAEEnvironment	Interface in com.sap.bi.da.extension.sdk
IDAEJob<T>	Interface in com.sap.bi.da.extension.sdk This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework
IDAEMetadataAcquisitionJob	Interface in com.sap.bi.da.extension.sdk
IDAEProgress	Interface in com.sap.bi.da.extension.sdk Callback interface for clients to be notified as job states are updated.
IDAExtension	Interface in com.sap.bi.da.extension.sdk This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.
INFO_JSON_PROP_NAME	Static variable in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend.
is32BitEnvironment()	Method in interface com.sap.bi.da.extension.sdk.IDAEEnvironment True if the environment is 32-bit, false if not.
isBackendOnClient()	Method in interface com.sap.bi.da.extension.sdk.IDAEEnvironment True if the client and the back end are on the same machine (e.g. Desktop). False if not (e.g. LIMA, Cloud)

Table 89: L, N, P, R

L, N, P, R	
LENGTH	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keyss
LEVELS	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
NAME	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
percentComplete(float)	Method in interface com.sap.bi.da.extension.sdk.IDAEProgress Notification that a job has completed a certain percentage of its execution.
RUNTIME_INFO_JSON_PROP_NAME	Static variable in interface com.sap.bi.da.extension.sdk.IDAEAcquisitionState The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend.

Table 90: T

toString()	Method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction
toString()	Method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType
toString()	Method in enum com.sap.bi.da.extension.sdk.DAEWorkflow
TYPE	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys

Table 91: V

valueOf(String)	Static method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction - Returns the enum constant of this type with the specified name.
valueOf(String)	Static method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType Returns the enum constant of this type with the specified name
valueOf(String)	Static method in enum com.sap.bi.da.extension.sdk.DAEWorkflow Returns the enum constant of this type with the specified name.
values()	Static method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.AggregationFunction Returns an array containing the constants of this enum type, in the order they are declared.
values()	Static method in enum com.sap.bi.da.extension.sdk.DAEConstants.Metadata.DataType Returns an array containing the constants of this enum type, in the order they are declared.
values()	Static method in enum com.sap.bi.da.extension.sdk.DAEWorkflow Returns an array containing the constants of this enum type, in the order they are declared.
VERSION	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
VERSION	Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata

6.11 Java JAR file structure

This topic describes the location and contents of the Java JAR archive inside a Data Access Extension Transport Unit ZIP file. This is compiled by Eclipse via the `export.xml` file.

This Jar file is located inside the eclipse folder within the Transport Unit, `<extensionname>_<versionnumber>.ZIP`, at:

- `> eclipse > plugins > <domain.name>.<extensionname>_<versionnumber>.jar`

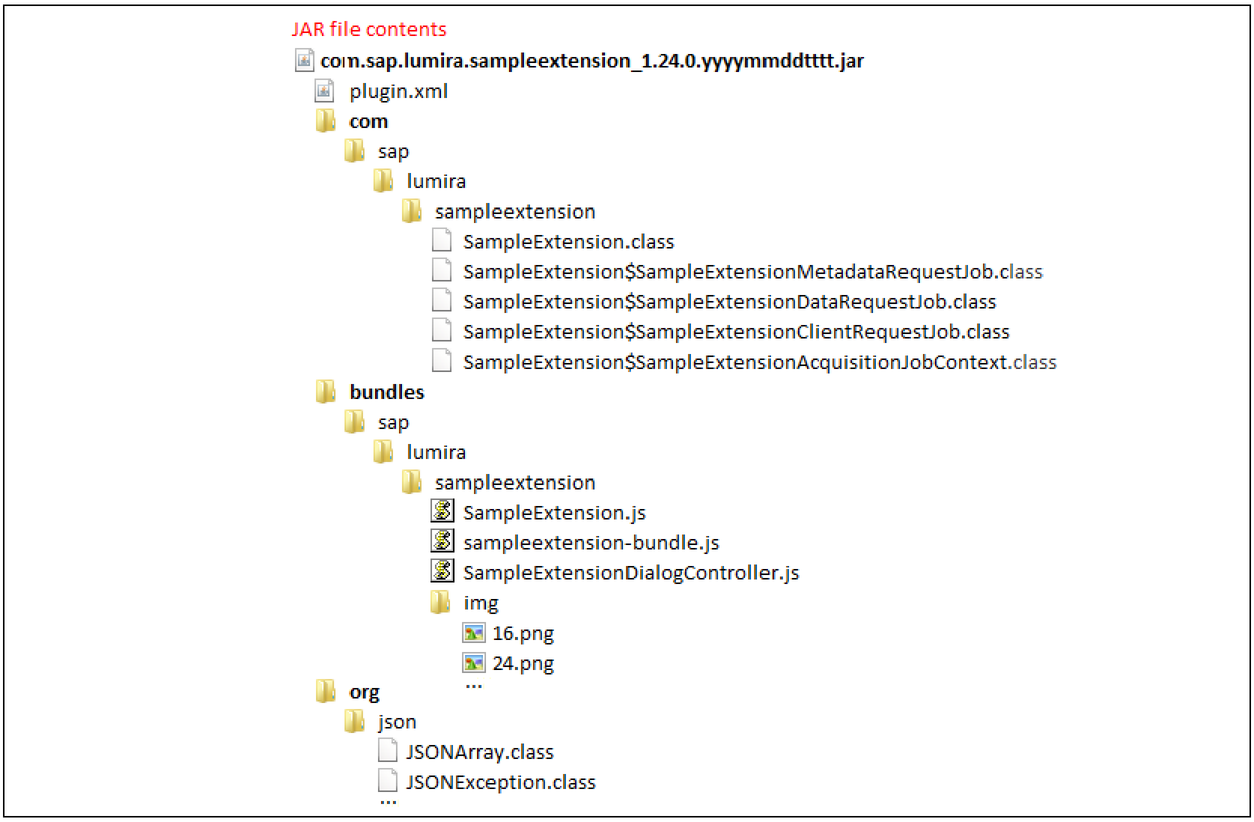
Table 92: Files within the JAR archive

Folder inside the JAR archive	Example	Description
<domain.name>.<extensionname>_<versionnumber>.jar `com.sap.lumira.sampleextension_1.24.0.201503011200.jar`		

Folder inside the JAR archive	Example	Description
<domain>	com	Location of Java *.class files for the main Data Acces Extesion project. ● com > plugins > lumira > <extensionname> > *.class SampleExtension.class
	org	Example of dependencies, *.class files the DAE project requires ● org > json > <JSON*>.class JSONArray.class JSONException.class
META-INF	META-INF	A listing of files included inside the JAR archive. Manifest-Version: 1.0 Ant-Version: Apache Ant 1.9.2 Created-By: 1.7.0_75-b13 (Oracle Corporation) Bundle-ManifestVersion: 2 Bundle-Name: SAP Lumira Sample Extension Bundle-SymbolicName: com.sap.lumira.sampleextension;singleton:=true Bundle-Version: 1.24.0.201503101453 Bundle-Vendor: SAP Bundle-RequiredExecutionEnvironment: JavaSE-1.7 Bundle-ActivationPolicy: lazy Require-Bundle: com.sap.bi.da.extension.sdk;bundle- version="1.24.0"

plugin.xml	plugin.xml	Unique attributes of the Data Access Exension in XML format.
		<?xml version="1.0" encoding="UTF-8"?>
		<?eclipse version="3.4"?>
		<plugin>
		   <extension
		         point="com.sap.bi.da.extension.backend">
		      <da_extension
		
		class="com.sap.lumira.sampleextension.SampleExtensi
		on"
		            id="sap.lumira.sampleextension">
		      </da_extension>
		   </extension>
		</plugin>

This is an example of what the inside of a typical JAR file would look like if it were expanded. In this example, files from the sampleextension are listed. (A few PNG and JSON class files are omitted.):



JAR file contents

```
com.sap.lumira.sampleextension_1.24.0.yyyymmddtttt.jar
  plugin.xml
  com
    sap
      lumira
        sampleextension
          SampleExtension.class
          SampleExtension$SampleExtensionMetadataRequestJob.class
          SampleExtension$SampleExtensionDataRequestJob.class
          SampleExtension$SampleExtensionClientRequestJob.class
          SampleExtension$SampleExtensionAcquisitionJobContext.class
  bundles
    sap
      lumira
        sampleextension
          SampleExtension.js
          sampleextension-bundle.js
          SampleExtensionDialogController.js
          img
            16.png
            24.png
            ...
  org
    json
      JSONArray.class
      JSONException.class
      ...
```

7 JAVA source file listings

7.1 Java File List

Here is a list of all Data Access Extension public Java files with brief descriptions:

Table 93: Classes

File name	Type	Name and Description
`DAEConstants.java`		
	class	DAEConstants
	class	DAEConstants.Metadata
	enum	DAEConstants.Metadata.Keys
	enum	Metadata.DataType
	enum	Metadata.AggregationFunction
`DAEWorkflow.java`		
	enum	DAEWorkflow
`DAException.java`		
	class	DAException Exception to be thrown when a problem is encountered during the execute() method of a DAE Job.
`IDAEAcquisitionJobContext.java`		
	interface	IDAEAcquisitionJobContext The context for a data acquisition.
`IDAEAcquisitionState.java`		
	interface	IDAEAcquisitionState
`IDAEClientRequestJob.java`		
	interface	IDAEClientRequestJob
`IDAEDataAcquisitionJob.java`		
	interface	IDAEDataAcquisitionJob
`IDAEEnvironment.java`		
	interface	IDAEEnvironment
`IDAEJob.java`		

File name	Type	Name and Description
	interface	IDAEJob< T > This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework.
IDAELogger.java		
	interface	This interface represents the logger passed to the extension. The assembly (Desktop, Edge, etc.) should create its own implementation of this so as to unify logging with the extension. "Trace" calls are intended to be seen by the developer, whereas "Log" are actionable items to be seen by an administrator.
IDAEMetadataAcquisitionJob.java		
	interface	IDAEMetadataAcquisitionJob
IDAEProgress.java		
	interface	IDAEProgress
		Callback interface for clients to be notified as job states are updated.
	parameter	percentComplete (float percentComplete) Notification that a job has completed a certain percentage of its execution.
IDAExtension.java		
	interface	This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework.

7.2 IDAExtension.java

```
package com.sap.bi.da.extension.sdk;
import java.util.Set;
/**
 * This interface represents a data acquisition extension to be used with the
<b>Lumira Data Acquisition framework</b>
 * <p>
 * You are expected to contribute ONE implementation of this interface which Lumira
will look for in your extension bundle.
 *
 * @version 1.0
 *
 */
public interface IDAExtension
{
    /**
     * Called to allow the extension to initialize itself.
     * <p>
     * <h3>Method life time</h3>
     * <ul>
     *   <li>This method is guaranteed to be called once</li>
     *   <li>This method is guaranteed to be called before the other methods.</li>
     * </ul>
     *
     * @param environment information about the environment in which the DA
extension is running
     */
    void initialize(IDAEEnvironment environment);
     /**
     * Get a context for data acquisition jobs.
     * @param acquisitionState the acquisition state.
     *
     * @return a container object to serve jobs which can provide DA info
(metadata, data) based on the point in time at which this context was created.
     */
    IDAEAcquisitionJobContext getDataAcquisitionJobContext(IDAEAcquisitionState
acquisitionState);
    /**
     * Asks your extension to handle requests that you sent from your client
extension code.
     *
     * If your client extension code made a request through the JavaScript API
<b>handleDAExtensionRequest</b>, the request is passed through here.
     * @param request
     *
     * @return a job to handle the client request
     */
    IDAEClientRequestJob getClientRequestJob(String request);
    /**
     * Get the supported workflows in the provided environment (and optionally,
with the provided acquisition state).
     * Typically, Lumira will ask your extension for these capabilities and the
extension is expected to answer them based on the environment argument.
     *
     * Scenarios where you might disable your extension:
     * <ul>
     *   <li>if your extension needs to run a 64-bit process and Lumira tells you
this is a 32-bit environment</li>
     *   <li>if you want your extension to be available only on certain versions of
Lumira product</li>
     * </ul>
     *
     * Note that if acquisitionState is null and the returned list is empty, this
extension will be disabled.
     * @param acquisitionState the applicable acquisitionState, or null to
determine applicability of workflows based on environment only.
     *        This argument is used to determine refreshability and editability of
an acquisition.
     *        This argument should be ignored for create-ability (i.e. just use the
environment argument)
     *
     * @return the list of supported workflows.
     */
    Set<DAEWorkflow> getEnabledWorkflows(IDAEAcquisitionState acquisitionState);
}
```

7.3 DAEWorkflow.java

```
package com.sap.bi.da.extension.sdk;
public enum DAEWorkflow
{
    CREATE  ("create"),
    EDIT    ("edit"),
    REFRESH ("refresh");
    private final String text;
    DAEWorkflow(String text) {
        this.text = text;
    }
    public String toString() {
        return this.text;
    }
}
```

7.4 IDAEAcquisitionState.java

```
package com.sap.bi.da.extension.sdk;
import java.util.Map;
public interface IDAEAcquisitionState {
    /*
     * Because the JSON AcquisitionState is merged directly into Java
DataSourceInfo,
     * this same property name is also directly used in there. The name space is
     * shared. Be aware of name clashes.
     */
    /**
     * The property name that extension client should use in the JSON Acquisition
     * State for information sent from extension client to extension backend.
     */
    String INFO_JSON_PROP_NAME = "info";
    /*
     * Because the JSON AcquisitionState is merged directly into Java
DataSourceInfo,
     * this same property name is also directly used in there. The name space is
     * shared. Be aware of name clashes.
     */
    /**
     * The property name that extension client should use in the JSON Acquisition
     * State for information sent from extension client to extension backend.
     *
     * Use this name for information that you don't wish to be persist across
restart.
     * For example, authentication information or acquired data.
     */
    String RUNTIME_INFO_JSON_PROP_NAME = "runtimeInfo";

    /**
     * This property holds information about the current runtime environment, such
as
     * productLocale and preferredViewingLocale
     */
    String ENV_PROPS_NAME = "envProps";
    /**
     * Owned by the extension.
     * @return info
     */
    String getInfo();
    /**
     * Owned by the extension. Not persisted beyond application restart.
     * @return runtimeInfo
     */
    String getRuntimeInfo();
    /**
     * Owned by the framework.
     * @return
     */
    Map<String, String> getEnvProps();
}
```

7.5 IDAEEnvironment.java

```
package com.sap.bi.da.extension.sdk;
import java.io.File;
public interface IDAEEnvironment
{
    /**
     * {@code true} if the environment allows trusted DA extensions, {@code false}
otherwise.
     * For example, on desktop, the returned value will be affected by the
application keycode.
     */
    boolean allowTrustedDAExtensions();
    /**
     * Get version string for some particular aspect of the enclosing environment.
     */
    String getEnvironmentVersion(String environmentId);  // e.g. "BOE", "HANA"

    /**
     * {@code true} if the client and the back end are on the same machine (e.g.
Desktop).  {@code false} if not (e.g.
     * LIMA, Cloud)
     */
    boolean isBackendOnClient();

    /**
     * {@code true} if the environment is 32-bit, {@code false} if not.
     */
    boolean is32BitEnvironment();

    /**
     * Get the temporary directory to which you may write temporary files.  This
directory may be shared among
     * extensions, so make sure to use file names that are unique.
     *
     * @return a directory to which the extension may store temporary files
     * @see File#createTempFile(String, String, File)
     */
    File getTemporaryDirectory();

    /**
     * Gets the logger for the enclosing environment.
     */
    IDAELogger getLogger();
}
```

7.6 IDAEAcquisitionJobContext.java

```
package com.sap.bi.da.extension.sdk;
```

```
/**
 * The context for a data acquisition.
 * Jobs returned by this context are expected to return results which are
consistent.
 * e.g. always return the same metadata, data is always consistent with the
metadata.
 */
public interface IDAEAcquisitionJobContext
{
    /**
     * Get a job to return the metadata corresponding to the point in time at which
this object was created.
     */
    IDAEMetadataAcquisitionJob getMetadataAcquisitionJob();
    /**
     * Get a job to return the data corresponding to the point in time at which
this object was created.
     */
    IDAEDataAcquisitionJob getDataAcquisitionJob();
    /**
     * Asks the extension to do any necessary clean up of associated resources such
as closing connections, freeing resources, etc.
     * This will be called when the extension framework is done with this context.
     */
    void cleanup();
}
```

7.7 IDAEMetadataAcquisitionJob.java

```
package com.sap.bi.da.extension.sdk;
public interface IDAEMetadataAcquisitionJob extends IDAEJob<String>
{
}
```

7.8 IDAEDataAcquisitionJob.java

```
package com.sap.bi.da.extension.sdk;
import java.io.File;
public interface IDAEDataAcquisitionJob extends IDAEJob<File>
{
}
```

7.9 IDAEClientRequestJob.java

```
package com.sap.bi.da.extension.sdk;
```

```
public interface IDAEClientRequestJob extends IDAEJob<String>
{
}
```

7.10 IDAEJob.java

```
package com.sap.bi.da.extension.sdk;
/**
 * This interface represents a data acquisition extension job to be used with the 
<b>Lumira Data Acquisition framework</b>
 *
 * @param <T> the type of the result returned by this job.
 */
public interface IDAEJob<T>
{
    /**
     * @return the result of this job when done. This call will block until a 
result is returned or the job terminates unsuccessfully.
     *
     */
    T execute (IDAEProgress callback) throws DAException;
    /**
     * Signals to this job to cancel the current operation as soon as possible.
     * It is highly recommended that your extension support cancel for usability 
reasons.
     *
     * Should be non-blocking.
     */
    void cancel ();
    /**
     * Asks the extension to do any necessary clean up of associated resources such 
as closing connections, freeing resources, etc.
     *
     * Lumira will always call this method after
     * - execute () is called and the result is no longer needed
     * - any API throwing a DAException
     * - cancel () was called
     */
    void cleanup ();
}
```

7.11 IDAELogger.java

Trace and logging provisions for debugging the extension and tracking errors.

```
package com.sap.bi.da.extension.sdk;
/**
 * This interface represents the logger passed to the extension.
 *
 * The assembly (Desktop, Edge, etc.) should create its own implementation of this 
so as to unify logging with
 * the extension.
 *
 * "Trace" calls are intended to be seen by the developer, whereas "Log" are
actionable items to be seen by an administrator.
 */
public interface IDAELogger {
    public void traceDebug(Object msg);
    public void traceInfo(Object msg);
    public void traceError(Object msg);
    public void traceDebug(Throwable t, Object msg);
    public void traceInfo(Throwable t, Object msg);
    public void traceError(Throwable t, Object msg);
    public void traceDebug(String format, Object... args);
    public void traceInfo(String format, Object... args);
    public void traceError(String format, Object... args);
    public void traceDebug(Throwable t, String format, Object... args);
    public void traceInfo(Throwable t, String format, Object... args);
    public void traceError(Throwable t, String format, Object... args);

    public void traceDebug(Class<?> c, String format, Object... args);
    public void traceInfo(Class<?> c, String format, Object... args);
    public void traceError(Class<?> c, String format, Object... args);

    public void logInfo(Object msg);
    public void logWarning(Object msg);
    public void logError(Object msg);
    public void logInfo(Throwable t, Object msg);
    public void logWarning(Throwable t, Object msg);
    public void logError(Throwable t, Object msg);
    public void logInfo(String format, Object... args);
    public void logWarning(String format, Object... args);
    public void logError(String format, Object... args);
    public void logInfo(Throwable t, String format, Object... args);
    public void logWarning(Throwable t, String format, Object... args);
    public void logError(Throwable t, String format, Object... args);

    public void logInfo(Class<?> c, String format, Object... args);
    public void logWarning(Class<?> c, String format, Object... args);
    public void logError(Class<?> c, String format, Object... args);
}
```

7.12 IDAEProgress.java

```
package com.sap.bi.da.extension.sdk;
/**
 * Callback interface for clients to be notified as job states are updated.
 */
public interface IDAEProgress
{

    /**
     * Notification that a job has completed a certain percentage of its execution.
     * @param percentComplete
     */
    void percentComplete (float percentComplete);
}
```

7.13 DAEConstants.java

```
package com.sap.bi.da.extension.sdk;
public final class DAEConstants {
    public static final class Metadata {
        public static final String VERSION = "1.0";
        public static final class Keys {
            public static final String VERSION = "version";
            public static final String COLUMNS = "columns";
            public static final String HIERARCHIES = "hierarchies";
            public static final String LEVELS = "levels";
            public static final String TYPE = "type";
            public static final String NAME = "name";
            public static final String ID = "id";
            public static final String ANALYTICAL_TYPE = "analyticalType";
            public static final String AGGREGATION_FUNCTION = "aggregationFunction";
        }
        public enum DataType {
            BOOLEAN("boolean"),
            INTEGER("integer"),
            BIGINTEGER("biginteger"),
            NUMBER("number"),
            DATE("date"),
            DATETIME("datetime"),
            TIME("time"),
            STRING("string");
            private final String text;
            private DataType(final String text) {
                this.text = text;
            }
            @Override
            public String toString() {
                return text;
            }
            public boolean compareToNoCase(String text) {
                return this.text.equals(text.toLowerCase());
            }
        }
        public enum AggregationFunction {
            NONE("none"),
            SUM("sum"),
            MIN("min"),
            MAX("max"),
            AVERAGE("average"),
            COUNT_ALL("count_all"),
            COUNT_DISTINCT("count_distinct");
            private final String text;
            private AggregationFunction(final String text) {
                this.text = text;
            }
            @Override
            public String toString() {
                return text;
            }
            public boolean compareToNoCase(String text) {
                return this.text.equals(text.toLowerCase());
            }
        }
    }
}
```

7.14 DAException.java

```
package com.sap.bi.da.extension.sdk;
/**
 * Exception to be thrown when a problem is encountered during the execute() method
of a DAE Job.
 * Note that this message will be presented to the user, so should be localized
using the locale provided by the DAE framework.
 */
public class DAException extends Exception
{
    private static final long serialVersionUID = 1L;
    /**
     * @param localizedMessage the localized message to be presented to the user
     * @param cause the parent Throwable if any
     */
    public DAException(String localizedMessage, Throwable cause) {
        super(localizedMessage, cause);
    }
    /**
     * @param localizedMessage the localized message to be presented to the user
     */
    public DAException(String localizedMessage) {
        super(localizedMessage);
    }
    public DAException(Throwable cause) {
        super(cause.getLocalizedMessage(), cause);
    }
}
```

7.15 Index

Alphabetical index of variables, methods. enums,

A

- `AGGREGATION_FUNCTION` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
- `allowTrustedDAExtensions()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEEnvironment`. True if the the environment allows trusted DA extensions, false otherwise.
- `ANALYTICAL_TYPE` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`

C

- `cancel()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEJob`. Signals to this job to cancel the current operation as soon as possible.
- `cleanup()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEAcquisitionJobContext`. Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc.
- `cleanup()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEJob` Asks the extension to do any necessary clean up of associated resources such as closing connections, freeing resources, etc.

- `COLUMNS` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
- `com.sap.bi.da.extension.sdk` - package com.sap.bi.da.extension.sdk
- `compareToNoCase(String)` - Method in enum com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.AggregationFunction`
- `compareToNoCase(String)` - Method in enum com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.DataType`

D

- `DAEConstants` - Class in com.sap.bi.da.extension.sdk
- `DAEConstants()` - Constructor for class com.sap.bi.da.extension.sdk.`DAEConstants`
- `DAEConstants.Metadata` - Class in com.sap.bi.da.extension.sdk
- `DAEConstants.Metadata()` - Constructor for class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata`
- `DAEConstants.Metadata.AggregationFunction` - Enum in com.sap.bi.da.extension.sdk
- `DAEConstants.Metadata.DataType` - Enum in com.sap.bi.da.extension.sdk
- `DAEConstants.Metadata.Keys` - Class in com.sap.bi.da.extension.sdk
- `DAEConstants.Metadata.Keys()` - Constructor for class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
- `DAEWorkflow` - Enum in com.sap.bi.da.extension.sdk
- `DAException` - Exception in com.sap.bi.da.extension.sdk. Exception to be thrown when a problem is encountered during the `execute()` method of a DAE Job.
- `DAException(String, Throwable)` - Constructor for exception com.sap.bi.da.extension.sdk.`DAException`
- `DAException(String)` - Constructor for exception com.sap.bi.da.extension.sdk.`DAException`
- `DAException(Throwable)` - Constructor for exception com.sap.bi.da.extension.sdk.`DAException`

E

- `ENV_PROPS_NAME` : IDAEAcquisitionState - Static variable. This property holds information about the current runtime environment, such as `productLocale` and `preferredViewingLocale`.
- `execute(IDAEProgress)` - Method in interface com.sap.bi.da.extension.sdk.`IDAEJob`

G

- `getClientRequestJob()` : IDAExtension - Method that asks your extension to handle requests that you sent from your client extension code.
- `getDataAcquisitionJob()` : IDAEAcquisitionJobContext - Method to get a job to return the data corresponding to the point in time at which this object was created.
- `getDataAcquisitionJobContext(IDAEAcquisitionState)` : IDAExtension - Method to get context for data acquisition jobs.
- `getEnabledWorkflows()` : IDAExtension - Method to get the supported workflows in the provided environment (and optionally, with the provided acquisition state).
- `getEnvironmentVersion()` : IDAEEnvironment - Method to get the version string for some particular aspect of the enclosing environment.
- `getEnvProps()` : IDAEAcquisitionState - Method owned by the framework.
- `getExtensionId()` : IDAExtension - Method returns the extension id that uniquely identifies your extension.
- `getInfo()` : IDAEAcquisitionState - Method owned by the extension.

- `getMetadataAcquisitionJob()` : IDAEAcquisitionJobContext - Method to get a job to return the metadata corresponding to the point in time at which this object was created.
- `getRuntimeInfo()` : IDAEAcquisitionState - Method in IDAEAcquisitionState, owned by the extension.

H

- `HIERARCHIES` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`

I

- `ID` - Static variable in class com.sap.bi.da.extension.sdk.DAEConstants.Metadata.Keys
- Interfaces in `com.sap.bi.da.extension.sdk`
  - `IDAEAcquisitionJobContext` - The context for a data acquisition.
  - `IDAEAcquisitionState`
  - `IDAEClientRequestJob`
  - `IDAEDataAcquisitionJob`
  - `IDAEEnvironment`
  - `IDAEJob<T>` - This interface represents a data acquisition extension job to be used with the Lumira Data Acquisition framework
  - `IDAELogger` - This interface represents the logger passed to the extension.
  - `IDAEMetadataAcquisitionJob`
  - `IDAEProgress` - Callback interface for clients to be notified as job states are updated.
  - `IDAExtension` - This interface represents a data acquisition extension to be used with the Lumira Data Acquisition framework You are expected to contribute ONE implementation of this interface which Lumira will look for in your extension bundle.
- `INFO_JSON_PROP_NAME` - Static variable in interface com.sap.bi.da.extension.sdk.`IDAEAcquisitionState`
  The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend.
- `is32BitEnvironment()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEEnvironment`
  True if the environment is 32-bit, false if not.
- `isBackendOnClient()` - Method in interface com.sap.bi.da.extension.sdk.`IDAEEnvironment`
  True if the client and the back end are on the same machine (example: Desktop). False if not (example: LIMA, Cloud)

L

- `LENGTH` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
- `LEVELS` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
- `logError`, `logInfo`, `logWarning` - Methods in interface com.sap.bi.da.extension.sdk.`IDAELogger`
  - logError(Object)
  - logError(Throwable, Object)
  - logError(String, Object...)
  - logError(Throwable, String, Object...)
  - logError(Class<?>, String, Object...)
  - logInfo(Object)
  - logInfo(Throwable, Object)
  - logInfo(String, Object...)
  - logInfo(Throwable, String, Object...)
  - logInfo(Class<?>, String, Object...)

  - logWarning(Object)
  - logWarning(Throwable, Object)
  - logWarning(String, Object...)
  - logWarning(Throwable, String, Object...)
  - logWarning(Class<?>, String, Object...)

N

- `NAME` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`

P

- `percentComplete(float)` - Method in interface com.sap.bi.da.extension.sdk.`IDAEProgress` Notification that a job has completed a certain percentage of its execution.

R

- `RUNTIME_INFO_JSON_PROP_NAME` - Static variable in interface com.sap.bi.da.extension.sdk.`IDAEAcquisitionState`. The property name that extension client should use in the JSON Acquisition State for information sent from extension client to extension backend.

T

- `toString()` - Method in enum
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.AggregationFunction`
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.DataType`
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.AggregationFunction`
- `traceDebug`, `traceError`, `traceInfo` ... Methods in interface com.sap.bi.da.extension.sdk.`IDAELogger`
  - traceDebug(Object)
  - traceDebug(Throwable, Object)
  - traceDebug(String, Object...)
  - traceDebug(Throwable, String, Object...)
  - traceDebug(Class<?>, String, Object...)
  - traceError(Object)
  - traceError(Throwable, Object)
  - traceError(String, Object...)
  - traceError(Throwable, String, Object...)
  - traceError(Class<?>, String, Object...)
  - traceInfo(ObjecttraceInfo)
  - traceInfo(Throwable, Object)
  - traceInfo(String, Object...)
  - traceInfo(Throwable, String, Object...)
  - traceInfo(Class<?>, String, Object...)
- `TYPE` - Static variable in class com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`

V

- valueOf(String) - Static method in enum. Returns the enum constant of this type with the specified name:
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.AggregationFunction`
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.DataType`
  - com.sap.bi.da.extension.sdk.`DAEWorkflow`

Returns the enum constant of this type with the specified name.

- `values()` - Static method in enum, Returns an array containing the constants of this enum type, in the order they are declared:
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.DataType`
  - com.sap.bi.da.extension.sdk.`DAEWorkflow`
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.AggregationFunction`
- `VERSION` - Static variable in classes:
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata.Keys`
  - com.sap.bi.da.extension.sdk.`DAEConstants.Metadata`

8 Example "sampleextension"

8.1 Instructions for creating the sampleextension

This code example for a Data Access Extension shows how to use Eclipse to build a plugin, install it using the SAP Lumira Extension Manager, and test the extension by importing sample data.

The following points summarize the steps involved in creating and installing the `sampleextension` project:

- Install and configure the required software as described in the Hardware and Software Requirements.
- Create all the project files for Eclipse as described in the following Eclipse Project for the "exsampleextension"
- Start Eclipse using the `eclipse.bat` batch file as described in Starting Eclipse with the `eclipse.bat` Batch File
- Import the project from the files you created
- Build the `sampleextension` in Eclipse, right-click `export.xml > Run As > Ant Build`
- Install the `sampleextension` into SAP Lumira using the *Extension Manager*
- Import the test data to verify that the extension works

The following detailed tutorial describes how to build, install, and test the `sampleextension` on SAP Lumira.

Building the sample extension using Eclipse

1. Run the batch file `\workspace_SampleExtension\eclipse.bat` to open Eclipse in a new window.
2. Double-click *platform.target* from the Eclipse *Project Explorer*.
3. In the *platform.target* window at the top right, click the link *Set as Target Platform*.
4. Right-click on `export.xml > Run As > Ant Build`.

> ℹ Note
>
> The Eclipse *Console* pane, located at the bottom right, displays the build progress. The *Total time:* will display to let you know the build is complete.

5. In Windows Explorer, verify the presence of the compiled extension at `...`
   `\com.sap.lumira.sampleextension > target >`
   `com.sap.lumira.sampleextension_1.24...zip`

Installing the extension into SAP Lumira

1. In SAP Lumira, open the *Extension Manager* from the *File > Extensions* menu.
2. In the *Extension Manager* dialog box, lower right, click the *Install Extension* button.
3. In the *Open* dialog box, navigate to and select the sample extension ZIP file, then click the *Open* button.
4. Close the *Extension Manager* and restart SAP Lumira.

Testing the sample extension in SAP Lumira

1. Start SAP Lumira and go to ▶ *File > New Dataset* ◀, under *Select a Source*, select *SAP Lumira Sample Extension*, (blue square icon), then select *Next*.

2. Enter the *Dataset Name*, the path to the data text CSV File (for example, `c:\Data\test.csv`), and the path to the *Metadata File* (for example, `C:\Data\test.txt`, and click *OK*.

   > Note
   >
   > The *Ping* button is an example of a simple user interface component added to the Javascript code. Click the *Ping* button, and a *pong* dialog box appears.

3. To display the data, from the left-most panel *MEASURES* section, drag *Integer* to the *X Axis* under the *MEASURES* label, and from the left-most panel*DIMENSIONS* section, drag *Number...* to the *Y Axis* under the *DIMENSIONS* label, and drag *String...* or *Date...* to *Legend Color DIMENSIONS* area.
4. Test the *Edit data source* and *Refresh document* Data menu items by swapping the data files. Make sure the metadata text file applies to each CSV file you swap. Do not have the CSV file locked open by another program such as Microsoft Excel.

8.2 Starting Eclipse with the eclipse.bat Batch File

The eclipse.bat file is suggested as a convenience to start Eclipse cleanly with the required Path environment variables set.

The file `eclipse.bat` is used to start Eclipse, and set the Path environment variables to declare the location of Eclipse, and the path to the Java Development Kit installation.

Eclipse Project Path: \

> Note
>
> This file is outside the project folder, `sap.lumira.sampleextension`.

The text of the file `eclipse.bat` is as follows:

```
set ECLIPSE_HOME=C:\Program Files\eclipse
set JAVA_HOME=C:\Program Files\Java\jdk1.7.0_75
start "eclipse" "%ECLIPSE_HOME%\eclipse.exe" -vm "%JAVA_HOME%\bin\javaw.exe" -data .
```

Two Path variables need to be set to reflect your current environment.

1. `ECLIPSE_HOME` is set to the Eclipse install location, for example:

   ```
   set ECLIPSE_HOME=C:\Program Files\eclipse
   ```

2. `JAVA_HOME` is set to the JDK (not the JRE Java runtime) install location, for example:

   ```
   set JAVA_HOME=C:\Program Files\Java\jdk1.7.0_75
   ```

3. To start Eclipse, run the batch file `eclipse.bat` to open Eclipse into a new window.

8.3 Eclipse Project for the "sampleextension"

All the project files needed to produce the sample extension data access extension plugin to be used with Eclipse.

This section lists all the files and code listings needed to create a buildable Eclipse project for the exampleextension.

Paths listed are relative to the Eclipse project folder. All project files are contained within the folder `sap.lumira.sampleextension`.

> **Note**
>
> The sizes in bytes apply to the project file if it is obtained online. Copy and pasting the text from this document to recreated the files may result in slightly different sizes depending on whether or not line returns are included.

The high level structure of the Eclipse project is as follows:

Table 94: Eclipse Project "sampleextension" folder and file listing

Folder or File	File (inside folder)	Size (bytes)	Description
src\ Note Download this library from https://github.com/douglascrockford/JSON-java	com\sap\lumira\sampleextension\SampleExtension.java	4,611	Main extension code
	src\org\json\JSONArray.java	32,193	JSON library file - Arrays
	src\org\json\JSONException.java	1,066	JSON library file - Exception
	src\org\json\JSONObject.java	57,539	JSON library file - Object
	src\org\json\JSONString.java	708	JSON library file - String
	src\org\json\JSONStringer.java	3,266	JSON library file - Stringer
	src\org\json\JSONTokener.java	13,002	JSON library file - Tokener
	src\org\json\JSONWriter.java	10,678	JSON library file - Writer

Folder or File	File (inside folder)	Size (bytes)	Description
WebContent\	sap\lumira\sampleextension\SampleExtension.js sap\lumira\sampleextension\SampleExtension-DialogController.js sap\lumira\sampleextension\sampleextension-bundle.js sap\lumira\sampleextension\img\16.png sap\lumira\sampleextension\img\24.png sap\lumira\sampleextension\img\32.png sap\lumira\sampleextension\img\32_w.png sap\lumira\sampleextension\img\48.png	2,456 4,405 618 140 151 160 162 193	Javascript support file Javascript support file Javascript support file 16-pixel square blue icon 24-pixel square blue icon 32-pixel square blue icon 32-pixel square light blue icon 48-pixel square blue icon
features\	sap\lumira\sampleextension\sampleextension-feature.json	655	JSON extension feature metadata
META-INF\	MANIFEST.MF	340	Manifest for the JAR extension plugin file
lib\	com.sap.bi.da.extension.sdk_1.24...jar	12,224	DAE SDK library included in SAP Lumira
.classpath	(file at the root of the project folder)	356	Eclipse IDE (XML) path to classes
.project	(file at the root of the project folder)	1,392	Eclipse IDE (XML) project description
build.properties	(file at the root of the project folder)	135	Eclipse IDE build properties
export.xml	(file at the root of the project folder)	4,298	Eclipse right-click: *Run As > Ant Build* (XML)
platform.target	(file at the root of the project folder)	303	Eclipse target platform plugin id (XML)
plugin.xml	(file at the root of the project folder)	333	Eclipse plugins to use (XML)

8.3.1 Eclipse IDE Project Structure

Eclipse is a typical integrated development environment (IDE) for Data Access Extension development.

The following project structure illustrates a typical Eclipse setup as viewed from the top level of the Project Explorer:

Table 95: Eclipse Project Explorer

Folder (F) or File (f) Name		Description
com.sap.lumira.<extensionName>	Library (Eclipse Env.)	Extension project name

Folder (F) or File (f) Name			Description
F	Plug-in Dependencies	Library (Eclipse Env.)	Data Access Extension jar library
F	JRE System Library	Library (Eclipse Env.)	Java Development Kit Library
F	JavaScript Resources	Library (Eclipse Env.)	Extension dependency, JavaScript library.
F	META-INF	Library (Eclipse Env.)	Auto-generated
F	lib	Library for SDK	JAR Library file of the Data Access Extension SDK.
F	src	Source Code Java	Location of the Java source code.
F	WebContent	Source Code Javascript	Location of the JavaScript files and icon image files.
F	target	Extension build	Output location of the ZIP file of the Data Access Extension (Transport Unit) that is created here as a result of the build from `export.xml`
f	plugin.xml	Plugin metadata	Eclipse project import / export. Contains the class and id name of the extension.
f	export.xml	Build instructions ANT	The project is built from this file with a right-click > Run As > Ant Build
f	build.properties	Eclipse project file	Eclipse project import / export
f	platform.target	Eclipse project file	Eclipse project import / export. Set the platform target when importing your project.
f	.classpath	Eclipse project file	Eclipse project import / export
f	.project	Eclipse project file	Eclipse project import / export

8.3.2 src folder

This folder contains the Java source code for the sampleextension project, as well as the JSON library files.

Eclipse Project Path: `\sap.lumira.sampleextension\src\`

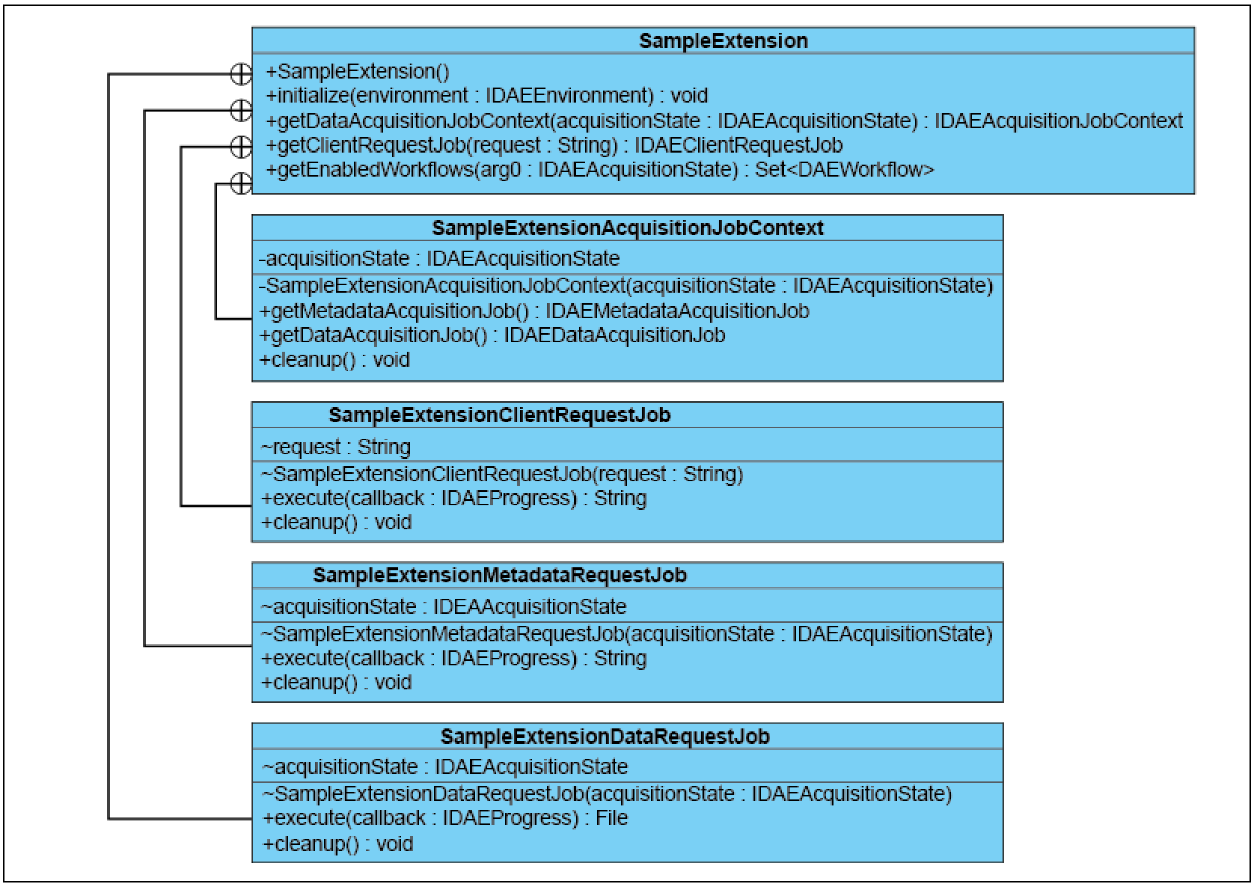
8.3.2.1 SampleExtension.java Code Example

This is the source code for the Sample Extension Eclipse project for the SAP Lumira Data Access Extension.

Eclipse Project Path: `\sap.lumira.sampleextension\src\com\sap\lumira\sampleextension\SampleExtension.java`

```
package com.sap.lumira.sampleextension;
import java.io.File;
import java.nio.file.Files;
import java.util.EnumSet;
import java.util.Set;
import com.sap.bi.da.extension.sdk.DAEWorkflow;
import com.sap.bi.da.extension.sdk.DAException;
import com.sap.bi.da.extension.sdk.IDAEAcquisitionJobContext;
import com.sap.bi.da.extension.sdk.IDAEAcquisitionState;
import com.sap.bi.da.extension.sdk.IDAEClientRequestJob;
import com.sap.bi.da.extension.sdk.IDAEDataAcquisitionJob;
import com.sap.bi.da.extension.sdk.IDAEEnvironment;
import com.sap.bi.da.extension.sdk.IDAEMetadataAcquisitionJob;
import com.sap.bi.da.extension.sdk.IDAEProgress;
import com.sap.bi.da.extension.sdk.IDAExtension;
import org.json.JSONObject;
public class SampleExtension implements IDAExtension {
    public SampleExtension() {
    }
    @Override
    public void initialize(IDAEEnvironment environment) {
        // This function will be called when the extension is initially loaded
        // This gives the extension to perform initialization steps, according to
the provided environment
    }
    @Override
    public IDAEAcquisitionJobContext getDataAcquisitionJobContext
(IDAEAcquisitionState acquisitionState) {
        return new SampleExtensionAcquisitionJobContext(acquisitionState);
    }
    @Override
    public IDAEClientRequestJob getClientRequestJob(String request) {
        return new SampleExtensionClientRequestJob(request);
    }
    private static class SampleExtensionAcquisitionJobContext implements
IDAEAcquisitionJobContext {
        private IDAEAcquisitionState acquisitionState;
        SampleExtensionAcquisitionJobContext(IDAEAcquisitionState acquisitionState)
{
            this.acquisitionState = acquisitionState;
        }
        @Override
        public IDAEMetadataAcquisitionJob getMetadataAcquisitionJob() {
            return new SampleExtensionMetadataRequestJob(acquisitionState);
        }
        @Override
        public IDAEDataAcquisitionJob getDataAcquisitionJob() {
            return new SampleExtensionDataRequestJob(acquisitionState);
        }
        @Override
        public void cleanup() {
            // Called once acquisition is complete
            // Provides the job the opportunity to perform cleanup, if needed
            // Will be called after both job.cleanup()'s are called
        }
    }
    private static class SampleExtensionDataRequestJob implements
IDAEDataAcquisitionJob
    {
        IDAEAcquisitionState acquisitionState;
        SampleExtensionDataRequestJob (IDAEAcquisitionState acquisitionState) {
            this.acquisitionState = acquisitionState;
        }
        @Override
        public File execute(IDAEProgress callback) throws DAException {
            try {
                JSONObject infoJSON = new JSONObject(acquisitionState.getInfo());
                File csv = new File(infoJSON.getString("csv"));
                return csv;
            } catch (Exception e) {
                throw new DAException("Sample Extension acquisition failed", e);
            }
        }
        @Override
        public void cancel() {
            // Cancel is currently not supported
        }
        @Override
        public void cleanup() {
            // Called once acquisition is complete
        }
    }
    private static class SampleExtensionMetadataRequestJob implements 
IDAEMetadataAcquisitionJob {
        IDAEAcquisitionState acquisitionState;
        SampleExtensionMetadataRequestJob (IDAEAcquisitionState acquisitionState) {
            this.acquisitionState = acquisitionState;
        }
        @Override
        public String execute(IDAEProgress callback) throws DAException {
            try {
                JSONObject infoJSON = new JSONObject(acquisitionState.getInfo());
                File metadataFile = new File(infoJSON.getString("metadata"));
                String metadata = new 
String(Files.readAllBytes(metadataFile.toPath()));
                return metadata;
            } catch (Exception e) {
                throw new DAException("Sample Extension acquisition failed", e);
            }
        }
        @Override
        public void cancel() {
            // Cancel is currently not supported
        }
        @Override
        public void cleanup() {
            // Called once acquisition is complete
        }
    }
    private class SampleExtensionClientRequestJob implements IDAEClientRequestJob {
        String request;
        SampleExtensionClientRequestJob(String request) {
            this.request = request;
        }
        @Override
        public String execute(IDAEProgress callback) throws DAException {
            if ("ping".equals(request)) {
                return "pong";
            }
            return null;
        }
        @Override
        public void cancel() {
            // Cancel is currently not supported
        }
        @Override
        public void cleanup() {
            // This function is NOT called
        }
    }
    @Override
    public Set<DAEWorkflow> getEnabledWorkflows(IDAEAcquisitionState 
acquisitionState) {
        // If the extension is incompatible with the current environment, it may 
disable itself using this function
        // return EnumSet.allOf(DAEWorkflow.class) to enable the extension
        // return EnumSet.noneOf(DAEWorkflow.class) to disable the extension
        // Partial enabling is not currently supported
        return EnumSet.allOf(DAEWorkflow.class);
    }
}
```

The following class lists diagramatically what is in the preceeding code.



SampleExtension

+SampleExtension()
+initialize(environment : IDAEEnvironment) : void
+getDataAcquisitionJobContext(acquisitionState : IDAEAcquisitionState) : IDAEAcquisitionJobContext
+getClientRequestJob(request : String) : IDAEClientRequestJob
+getEnabledWorkflows(arg0 : IDAEAcquisitionState) : Set<DAEWorkflow>

SampleExtensionAcquisitionJobContext

-acquisitionState : IDAEAcquisitionState

-SampleExtensionAcquisitionJobContext(acquisitionState : IDAEAcquisitionState)
+getMetadataAcquisitionJob() : IDAEMetadataAcquisitionJob
+getDataAcquisitionJob() : IDAEDataAcquisitionJob
+cleanup() : void

SampleExtensionClientRequestJob

~request : String

~SampleExtensionClientRequestJob(request : String)
+execute(callback : IDAEProgress) : String
+cleanup() : void

SampleExtensionMetadataRequestJob

~acquisitionState : IDEAAcquisitionState

~SampleExtensionMetadataRequestJob(acquisitionState : IDAEAcquisitionState)
+execute(callback : IDAEProgress) : String
+cleanup() : void

SampleExtensionDataRequestJob

~acquisitionState : IDAEAcquisitionState

~SampleExtensionDataRequestJob(acquisitionState : IDAEAcquisitionState)
+execute(callback : IDAEProgress) : File
+cleanup() : void

8.3.2.2 *.json Library Java Code

JSON library files used in the Sample Extension Eclipse project for SAP Lumira Data Access Extension.

The following is a list of Java source files that are used in this project.

Path example: `\sap.lumira.sampleextension\src\org\json\JSONArray.java`

These (7) files used in this project are available from the JSON library from http://www.json.org/java/, and may be downloaded from https://github.com/douglascrockford/JSON-java

- JSONArray.java (Raw code example path: https://github.com/douglascrockford/JSON-java/blob/master/JSONArray.java
- JSONException.java
- JSONObject.java
- JSONString.java
- JSONStringer.java
- JSONTokener.java
- JSONWriter.java

8.3.3 WebContent folder

Listing of Javascript code files and image files used in the Data Access Extension sample extension project.

Eclipse Project Path:

- `\sap.lumira.sampleextension\WebContent\com\sap\lumira\sampleextension\*.js`
- `\sap.lumira.sampleextension\WebContent\com\sap\lumira\sampleextension\img\*.png`

The PNG files can be created for this project using any screen capture or paint program that can specify pixel dimensions and save the file as RGB PNG. The following PNG and Javascript files are used in the WebContent folder:

Table 96: WebContent files

Filename	Description
SampleExtension.js	This JavaScript file handles the workflow, CREATE, EDIT and REFRESH.
sampleextension-bundle.js	This is the metadata that describes the extension, the name, version number and so on.
SampleExtensionDialog-Controller.js	Code for the dialog box that displays after selecting this extension module in the Add new data-set dialog box.
img\16.png	16 pixel icon in the PNG grapics file format. Blue color.
img\24.png	24 pixel icon in the PNG grapics file format. Blue color.
img\32.png	32 pixel icon in the PNG grapics file format. Blue color.
img\32_w.png	32 pixel icon in the PNG grapics file format. W = White, which refers to the icon when selected in the interface. Light blue color.
img\48.png	48 pixel icon in the PNG grapics file format. Blue color.

8.3.3.1 SampleExtension.js JavaScript File

This JavaScript file handles the workflow, CREATE, EDIT and REFRESH.

Path: `\sap.lumira.sampleextension\WebContent\sap\lumira\sampleextension\SampleExtension.js`

```
define(["service!sap.bi.da.extension.sdk.clientRequestService",
"SampleExtensionDialogController"], function (ClientRequestService,
SampleExtensionDialogController) {
    "use strict";
    function SampleExtension() {
        var EXTENSION_ID = "sap.lumira.sampleextension";
        var fServiceCall = function(request, fSuccess, fFailure) {
            // The ClientRequestService is a way for the extension to communicate
to its Java backend
            // request will be passed to getClientRequestJob()
            // The value returned by clientRequestJob.execute() will be passed to
the fSucess callback
            // If an error occurs, the fFailure callback will be called
            ClientRequestService.callClientRequestService(EXTENSION_ID, request,
fSuccess, fFailure);
        };
        var createSampleExtensionDialog = function(acquisitionState, workflow) {
            var oDeferred = new jQuery.Deferred();
            var controller = new SampleExtensionDialogController(acquisitionState,
oDeferred, fServiceCall, workflow);
            controller.showDialog();
            return oDeferred.promise();
        };
        // This function will be called during a create dataset workflow
        // This function must immediately return a promise object
        // When the extension is finished performing UI tasks, resolve the promise
with the acquisitionState and dataset name
        // Other workflows do not need the dataset name
        // The resolved acquisitionState will be passed to the extension Java
backend getDataAcquisitionJobContext()
        this.doCreateWorkflow = function(acquisitionState) {
            return createSampleExtensionDialog(acquisitionState, "CREATE");
        };
        // This function will be called during an edit dataset workflow
        this.doEditWorkflow = function(acquisitionState) {
            return createSampleExtensionDialog(acquisitionState, "EDIT");
        };
        // This function will be called during a refresh workflow
        // This function should refresh the dataset with existing parameters
        // Minimal UI should be shown, if any
        this.doRefreshWorkflow = function(acquisitionState) {
            var oDeferred = new jQuery.Deferred();
            oDeferred.resolve(acquisitionState);
            return oDeferred.promise();
        };
    }
    // Functions that do not need to access private variables can be declared as
part of the prototype
    // This function must return an Object with properties Title and SubTitle,
determined by the provided acquisitionState
    // This will be displayed as an entry in the Most Recently Used pane
    SampleExtension.prototype.getConnectionDescription = function(acquisitionState)
{
        var info = JSON.parse(acquisitionState.info);
        return {
            Title: info.datasetName,
            SubTitle: info.csv
        };
    };
    // getIcon## must return a path to an image with size 48px*48px
    SampleExtension.prototype.getIcon48 = function() {
        return "/img/48.png";
    };
    SampleExtension.prototype.getIcon32 = function() {
        return "/img/32.png";
    };
    // The white version of the icon will be displayed when the extension is 
highlighted in the New Dataset dialog
    SampleExtension.prototype.getIcon32_white = function() {
        return "/img/32_w.png";
    };
    SampleExtension.prototype.getIcon24 = function() {
        return "/img/24.png";
    };
    SampleExtension.prototype.getIcon16 = function() {
        return "/img/16.png";
    };
    return SampleExtension;
});
```

8.3.3.2 SampleExtensionDialogController.js JavaScript File

Javascript that draws the dialog box and handles the user interface for the sampleextension Data Access Extension.

Eclipse Project Path: `\sap.lumira.sampleextension\WebContent\sap\lumira\sampleextension\SampleExtensionDialogController.js`

```
define(function() {
    "use strict";
    var SampleExtensionDialogController = function(acquisitionState, oDeferred, 
fServiceCall, workflow) {
        /*
        Create dialog controls
        */
        var dLayout = new sap.ui.commons.layout.MatrixLayout({
            layoutFixed : true,
            columns : 2,
            width : "570px",
            widths : [ "20%", "80%" ]
        });
        var datasetNameTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : "",
            enabled : workflow === "CREATE"
        });
        var datasetNameLbl = new sap.ui.commons.Label({
            text : "Dataset Name:",
            labelFor : datasetNameTxt
        });
        dLayout.createRow({
            height : "30px"
        }, datasetNameLbl, datasetNameTxt);
        // These paths correspond to the included sample data if the workspace was 
unzipped to the C drive
        var datasetTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : 'C:\\workspace_SampleExtension\\Sample Data\\data.csv'
        });
        var datasetLbl = new sap.ui.commons.Label({
            text : "CSV File:",
            labelFor : datasetTxt
        });
        dLayout.createRow({
            height : "30px"
        }, datasetLbl, datasetTxt);
        var metadataTxt = new sap.ui.commons.TextField({
            width : '100%',
            value : 'C:\\workspace_SampleExtension\\Sample Data\\metadata.txt'
        });
        var metadataLbl = new sap.ui.commons.Label({
            text : "Metadata File:",
            labelFor : metadataTxt
        });
        dLayout.createRow({
            height : "30px"
        }, metadataLbl, metadataTxt);
        // Client request call example
        var pingBtn = new sap.ui.commons.Button({
            press : [ function() {
                fServiceCall("ping", function(response) {
                    sap.ui.commons.MessageBox.alert(response);
                }, function(actionRequired, errorMessage, fullErrorObject) {
                    sap.ui.commons.MessageBox.alert(errorMessage);
                });
            }, this ],
            text : "Ping",
            enabled : true
        }).addStyleClass("button_ellipsis");
        dLayout.createRow({
            height : "30px"
        }, pingBtn);
        /*
        Button press events
        */
        var buttonCancelPressed = function() {
            dialog.close(); // dialog is hoisted from below
        };
        var buttonOKPressed = function() {
            var info = {};
            info.csv = datasetTxt.getValue();
            info.metadata = metadataTxt.getValue();
            info.datasetName =  datasetNameTxt.getValue();
            acquisitionState.info = JSON.stringify(info);
            oDeferred.resolve(acquisitionState, datasetNameTxt.getValue());
            dialog.close();
        };
        var okButton = new sap.ui.commons.Button({
            press : [ buttonOKPressed, this ],
            text : "OK",
            tooltip : "OK"
        }).setStyle(sap.ui.commons.ButtonStyle.Accept);
        var cancelButton = new sap.ui.commons.Button({
            press : [ buttonCancelPressed, this ],
            text : "Cancel",
            tooltip : "Cancel"
        }).addStyleClass(sap.ui.commons.ButtonStyle.Default);
        var onClosed = function() {
            if (oDeferred.state() === "pending") {
                oDeferred.reject();
            }
        };
        /*
        Modify controls based on acquisitionState
        */
        var envProperties = acquisitionState.envProps;
        if (acquisitionState.info) {
            var info = JSON.parse(acquisitionState.info);
            datasetTxt.setValue(info.csv);
            metadataTxt.setValue(info.metadata);
            envProperties.datasetName = info.datasetName;
        }
        datasetNameTxt.setValue(envProperties.datasetName);
        /*
        Create the dialog
        */
        var dialog = new sap.ui.commons.Dialog({
            width : "720px",
            height : "480px",
            modal : true,
            resizable : false,
            closed : onClosed,
            content: [dLayout],
            buttons : [okButton, cancelButton]
        });
        dialog.setTitle("Sample Extension: " + envProperties.datasetName);
        this.showDialog = function() {
            dialog.open();
        };
    };
    return SampleExtensionDialogController;
});
```

8.3.3.3 sampleextension-bundle.js JavaScript File

Metadata information for the sampleextension, version, name, provider and so on.

Eclipse Project Path: `\sap.lumira.sampleextension\WebContent\sap\lumira\sampleextension \sampleextension-bundle.js`

```
define([], function() {
    return sap.bi.framework.declareBundle({
        "id" : "sap.lumira.sampleextension",
        "version" : "REPLACE_VERSION",
         "components" : [{
            "id" : "sap.lumira.sampleextension",
            "provide" : "sap.bi.da.extension.client",
            "module" : "SampleExtension",
            "customProperties" : {
                "displayName" : "SAP Lumira Sample Extension",
                "description" : "for SAP Lumira Data Acquisition Framework"
            }
        }],
        "dependencies": ["sap.bi.da.extension.sdk.clientRequestService"]
    });
});
```

8.3.4 features Folder

This folder contains the `sampleextension-feature.json` file.

Eclipse Project Path: `\sap.lumira.sampleextension\features\sap\lumira\sampleextension\`

8.3.4.1 sampleextension-feature.json

Eclipse Project Path: `\sap.lumira.sampleextension\features\sap\lumira\sampleextension\sampleextension-feature.json`

```
{
  "metadataVersion": "1.0",
  "id": "sap.lumira.sampleextension",
  "name": "SAP Lumira Sample Extension",
  "description": "for SAP Lumira Data Acquisition Framework",
  "version": "REPLACE_VERSION",
  "vendor" : {
    "name": "SAP",
    "url": "www.sap.com"
  },
  "requires": [
    {
      "id": "sap.bi.da.extension.sdk",
      "version": "1.24.0",
      "match": "greaterOrEqual"
    }
  ],
  "eclipse": {
    "plugins": [
      {
        "id": "com.sap.lumira.sampleextension",
        "version": "REPLACE_VERSION"
      }
    ]
  },
  "bundles": [
    {
      "id": "sap.lumira.sampleextension",
      "version": "REPLACE_VERSION"
    }
  ]
}
```

8.3.5 lib Folder

This is the folder that contains the Data Access Extension SDK JAR file.

Eclipse Project Path: `\sap.lumira.sampleextension\lib\`

The SDK JAR file is located within SAP Lumira at the following location:

```
<installation directory>SAP Lumira\Desktop\plugins\
```

8.3.5.1 com.sap.bi.da.extension.sdk*.jar

The Data Access Extension SDK library JAR file that is included in SAP Lumira 1.24 and later.

Eclipse Project Path: `\sap.lumira.sampleextension\lib\com.sap.bi.da.extension.sdk_1.24<version number>.jar`

The SDK JAR file is located within SAP Lumira at the following location:

```
<installation directory>SAP Lumira\Desktop\plugins
\com.sap.bi.da.extension.sdk_1.24<version number>.jar
```

8.3.6 export.xml Build File

This is the file used to build the Eclipse exampleextension project.

Eclipse Project Path: `\sap.lumira.sampleextension\export.xml`

In Eclipse, build the extension by right-clicking on `export.xml`, > Run As > Ant Build.

This will create the SAP Lumira Data Access Extension ZIP file in the `target` folder in the Eclipse workspace directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<project default="plugin_export" name="build">
    <!-- Set to the directory defined in targetplatform.target -->
    <property name="targetplatform.dir" location="lib" />

    <target name="plugin_export">
        <!-- Define build directories -->
        <property name="build.root" location="target" />
        <property name="build.temp" location="${build.root}/temp" />
        <property name="unpack.dir" location="${build.temp}/unpack" />
        <!-- Create build directories -->
        <delete dir="${build.temp}" />
        <mkdir dir="${build.temp}" />
        <!-- Read the MANIFEST.MF -->
        <copy file="META-INF/MANIFEST.MF" tofile="${build.temp}/
MANIFEST.properties" />
        <replace file="${build.temp}/MANIFEST.properties">
            <replacefilter token=":=" value="=" />
            <replacefilter token=":" value="=" />
            <replacetoken>;</replacetoken>
            <replacevalue>
            </replacevalue>
        </replace>
        <property file="${build.temp}/MANIFEST.properties" />
        <property name="plugin.name" value="${Bundle-SymbolicName}" />
        <!-- Compute plugin version -->
        <tstamp>
              <format property="version.qualifier" pattern="yyyyMMddHHmm"
unit="minute"/>
        </tstamp>
        <loadresource property="version.full">
          <propertyresource name="Bundle-Version"/>
          <filterchain>
            <tokenfilter>
              <filetokenizer/>
              <replacestring from="qualifier" to="${version.qualifier}"/>
            </tokenfilter>
          </filterchain>
        </loadresource>
        <property name="plugin.jarname" value="${plugin.name}_${version.full}" />
        <!-- Set plugin version in MANIFEST.MF-->
        <copy file="META-INF/MANIFEST.MF" tofile="${build.temp}/MANIFEST.MF" />
        <replace file="${build.temp}/MANIFEST.MF" token="qualifier" value="$
{version.qualifier}"/>
        <!-- Plugin locations -->
        <property name="plugin.dir" location="${unpack.dir}/eclipse/plugins" />
        <property name="plugin.jar" location="${plugin.dir}/$
{plugin.jarname}.jar" />
        <property name="features.dir" location="${unpack.dir}/features" />
        <property name="product.zip" location="${build.root}/$
{plugin.jarname}.zip" />
        <property name="bundles.dir" location="${unpack.dir}/bundles" />

        <!-- Copy the WebContent folder -->
        <delete dir="${bundles.dir}" />
        <copy todir="${bundles.dir}">
            <fileset dir="WebContent/" />
        </copy>
        <!-- Replace version in bundle.js file -->
        <replace dir="${bundles.dir}" value="${version.full}">
          <include name="**/*-bundle.js"/>
          <replacetoken>REPLACE_VERSION</replacetoken>
        </replace>
        <!-- Copy the features folder -->
        <delete dir="${features.dir}" />
        <copy todir="${features.dir}">
            <fileset dir="features" />
        </copy>
        <!-- Replace version in feature json file -->
        <replace dir="${features.dir}" value="${version.full}">
          <include name="**/*.json"/>
          <replacetoken>REPLACE_VERSION</replacetoken>
        </replace>
        <!-- Assemble plug-in JAR -->
        <property name="build.result.folder" location="bin" />
        <antcall target="build.jar" />
        <mkdir dir="${plugin.dir}" />
        <jar destfile="${plugin.jar}" manifest="${build.temp}/MANIFEST.MF"
zip64Mode="never">
            <zipfileset dir="${build.result.folder}" />
            <zipfileset dir="." includes="plugin.xml" />
        </jar>
        <!-- If the extension has any external jar dependencies, they must be
included at this stage -->
        <!-- Copy any required jars to the ${plugin.dir} folder and entries must be
added to the feature.json file under eclipse.plugins -->
        <!-- Assemble the product zip -->
        <zip destfile="${product.zip}" level="9" zip64Mode="never">
            <fileset dir="${build.temp}/unpack" />
        </zip>
    </target>
    <target name="build.jar" unless="build.jar" description="build classes">
        <delete dir="${build.result.folder}" />
        <mkdir dir="${build.result.folder}" />
        <path id="build.classpath">
            <fileset dir="${targetplatform.dir}">
                <include name="**/*.jar" />
            </fileset>
        </path>
        <!-- compile the source code -->
        <property name="bundleJavacSource" value="1.7" />
        <property name="bundleJavacTarget" value="1.7" />
        <javac destdir="${build.result.folder}" failonerror="false" verbose="false"
debug="on" includeAntRuntime="no" source="${bundleJavacSource}" target="$
{bundleJavacTarget}">
            <classpath refid="build.classpath" />
            <src path="src/" />
            <src path="WebContent/" />
            <compilerarg value="@${build.temp}/javaCompiler...args"
compiler="org.eclipse.jdt.core.JDTCompilerAdapter" />
            <compilerarg line="-log &apos;${build.temp}/build.log&apos;"
compiler="org.eclipse.jdt.core.JDTCompilerAdapter" />
        </javac>
    </target>
</project>
```

8.3.7 plugin.xml

This is the metadata description information for the plugin the lists the `class` and `id`.

Eclipse Project Path: `\sap.lumira.sampleextension\plugin.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="com.sap.bi.da.extension.backend">
      <da_extension
            class="com.sap.lumira.sampleextension.SampleExtension"
            id="sap.lumira.sampleextension">
      </da_extension>
   </extension>
</plugin>
```

8.3.8 platform.target

This describes the Data Access Extension SDK name.

Double-click the `platform.target` file in the Eclipse *Project Explorer* pane, then click the link in the *Target Definition* window, *Set as Target Platform*.

Eclipse Project Path: `\sap.lumira.sampleextension\platform.target`

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?pde version="3.8"?><target name="targetplatform" sequenceNumber="7">
<locations>
<location path="${project_loc}\lib" type="Directory"/>
</locations>
<includeBundles>
<plugin id="com.sap.bi.da.extension.sdk"/>
</includeBundles>
</target>
```

8.3.9 build.properties

Eclipse build properties for the sampleextension project.

Eclipse Project Path: `\sap.lumira.sampleextension\build.properties`

```
source.. = src/,\
           WebContent/
output.. = bin/
bin.includes = META-INF/,\
               .,\
               plugin.xml
```

8.3.10 META-INF Folder

This folder contains the Eclipse auto-generated `MANIFEST.MF` file. The content is listed here for reference purposes.

Eclipse Project Path: `\sap.lumira.sampleextension\META-INF\`

8.3.10.1 MANIFEST.MF

Eclipse manifest file `MANIFEST.MF` for the project sampleextension.

Eclipse Project Path: `\sap.lumira.sampleextension\META-INF\MANIFEST.MF`

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: SAP Lumira Sample Extension
Bundle-SymbolicName: com.sap.lumira.sampleextension;singleton:=true
Bundle-Version: 1.24.0.qualifier
Bundle-Vendor: SAP
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Bundle-ActivationPolicy: lazy
Require-Bundle: com.sap.bi.da.extension.sdk;bundle-version="1.24.0"
```

8.3.11 .project File

Eclipse project environment description file. This file allows Eclipse to import the sampleextension project.

Eclipse Project Path: `\sap.lumira.sampleextension\.project`

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>sap.lumira.sampleextension</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>org.eclipse.wst.jsdt.core.javascriptValidator</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.eclipse.jdt.core.javabuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.eclipse.pde.ManifestBuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.eclipse.pde.SchemaBuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
    </buildSpec>
    <natures>
        <nature>org.eclipse.jdt.core.javanature</nature>
        <nature>org.eclipse.wst.jsdt.core.jsNature</nature>
        <nature>org.eclipse.pde.PluginNature</nature>
    </natures>
</projectDescription>
```

8.3.12 .classpath File

Eclipse classpath definitions file.

Eclipse Project Path: `\sap.lumira.sampleextension\.classpath`

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
    <classpathentry kind="con" path="org.eclipse.pde.core.requiredPlugins"/>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="src" path="WebContent"/>
    <classpathentry kind="output" path="bin"/>
</classpath>
```

8.3.13 target Folder

The target folder is where the `sampleextension.zip` file is located. The content is listed here for reference purposes, as it does need to be created by the programmer..

Eclipse Project Path: `\sap.lumira.sampleextension\.classpath`

The target folder contains the `sampleextension...zip` file, which is ready to be installed into SAP Lumira using the Extension Manager.

High level structure of the project is as follows:

Table 97: 'target' Folder and File Listing of the sampleextension

Folder or File	File (inside folder)	Size (bytes)	Description
com.sap.lumira.sampleextension_1.24...zip		30,037	The installable Data Access Extension ZIP file. This is the file that is installed into SAP Lumira using the Extension Manager.

Folder or File	File (inside folder)	Size (bytes)	Description
temp\unpack\	**bundles**\sap\sumira\sampleextension\ - img ○ 16.png ○ 24.png ○ 32.png ○ 32_2.png ○ 48.png - SampleExtension.js - sampleextension-bundle.js - SampleExtensionDialogController.js	70,545	temporary folder containing all the files inside the ZIP archive
	eclipseplugins\ - com.sap.lumira.sampleextension_1.24...jar		
	features\sap\lumira\sampleextension - sampleextension-feature.json		

8.4 Sample Data

8.4.1 Example Data

This is a CSV data file with 4 columns and 22 rows.

The following data is used for the example Data Access Extension. Note that all items are in double quotes. The third line has NUL bytes that cannot be displayed in text.

Save the file as `test.csv`.

```
"0","zero","0.0","0","1999-12-31"
"1","one","1.0","1","2000-01-01"
, , , , ,
"-1","quote escape:""","-0.01","1","2014-02-20"
```

Here is an alternate data set that will work with the same metadata file:

```
"1","First Row","0","12/31/1999"
"2","Second Row (CHECK IF FIRST ROW APPEARS)","1","1/1/2000"
"3","Third Row","0","2/20/2014"
"4","Fourth Row","1","1/20/1960"
"5","Fifth Row","0","3/15/1970"
"6","Sixth Row","1","2/18/1900"
```

```
"7","Seventh Row","0","12/31/1999"
"8,"Eighth,Row",1","1/1/2000"
"9","100000000000000","0","2/20/2014"
"10","1010.101","1","1/20/1960"
"11","κόσμε","0","3/15/1970"
"12","«ταБЬℓσ»","1","2/18/1900"
"13","٩(●̮̮̃•̃)۶","0","12/31/1999"
"14,"\t\r\n.","0","1/1/2000"
"15,"k6/°",,"2/20/2014"
"16,"3g╥","1","1/20/1960"
"17","WÉïqo¡","0","3/15/1970"
"18,"NEXT ROW IS BLANK","1","2/18/1900"
,,,
"20","EXTENSION 1!@#$%^&*()é 拡張 расширение Erweiterung","1","1/1/2000"
"21","SECOND LAST ROW (CHECK IF LAST ROW APPEARS)","0","2/20/2014"
"22","LAST ROW","1","1/1/2000"
```

8.4.2 Example Metadata

This is sample metadata for the example application. This file describes the column characteristics.

The following example is a matching file for the CSV data immediately preceeding. There are four columns described, each of which have attributes such as the `name`, `id`, `type`, `description` and `analyticalType` and `aggregationFunction` specified. The `dimensions` and `hierarchies` are not qualified, so they are left blank.

Save the file as `test.txt`.

Table 98: Column description metadata file

Parameter name	Column 1	Column 2	Column 3	Column 4	Column 5
name	Integer (32-bit signed)	String (max length 4096)	Number (float)	Boolean (True=1, False=0)	Date (yyyy-MM-dd)
id	id0	id1	id2	id3	id4
type	Integer	String	Number	Boolean	Date
analytical-Type	dimension	dimension	dimension	dimension	dimension

```
{
    "version": "1.0",
    "columns": [
        {
            "name": "Integer (32-bit signed)",
            "id": "id0",
            "type": "Integer",
            "analyticalType": "dimension"
        },
        {
            "name": "String (max length 4096)",
            "id": "id1",
            "type": "String",
            "analyticalType": "dimension"
        },
        {
            "name": "Number (float)",
            "id": "id2",
            "type": "Number",
            "analyticalType": "dimension"
        },
        {
            "name": "Boolean (True=1, False=0)",
            "id": "id3",
            "type": "Boolean",
            "analyticalType": "dimension"
        },
        {
            "name": "Date (yyyy-MM-dd)",
            "id": "id4",
            "type": "Date",
            "analyticalType": "dimension"
        }
    ]
}
```

9 Administration

9.1 Installing a Data Access Extension

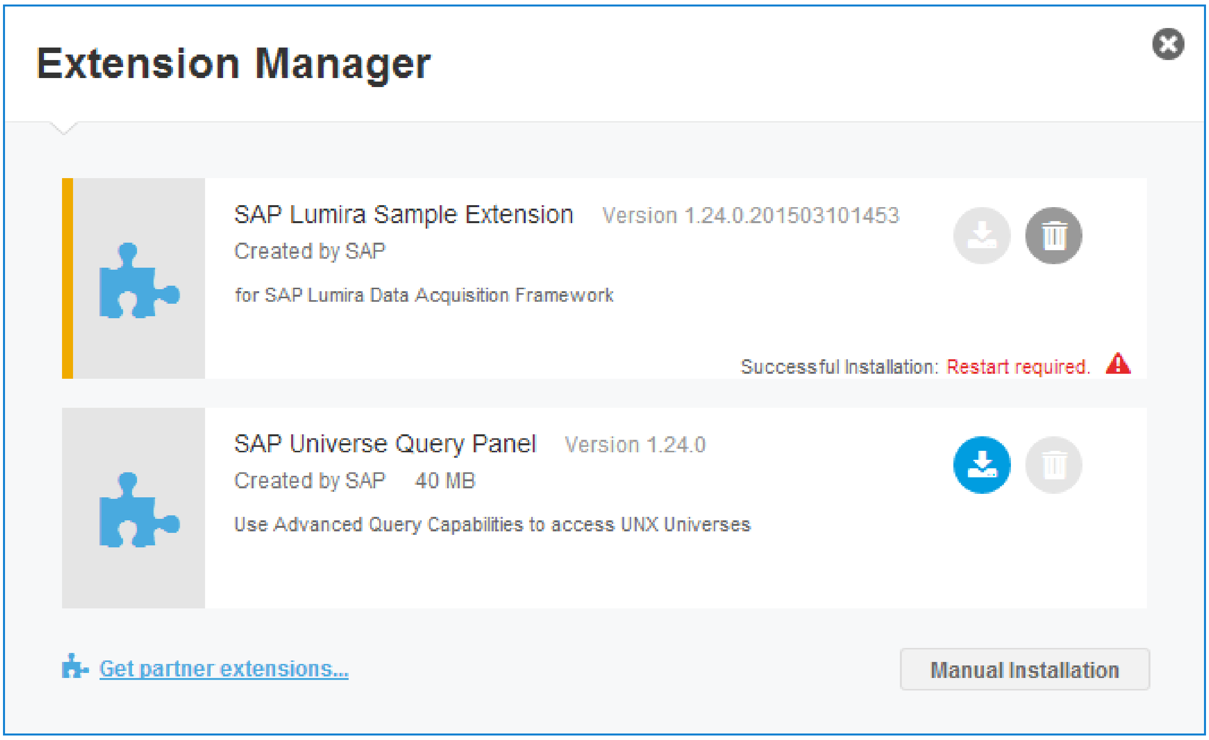
Context

After you create a data access extension, you need to install it into SAP Lumira using the *Extension Manager* and restart.

To deploy an extension into SAP Lumira, perform the following steps:

Procedure

1. Start the SAP Lumira application.
2. Navigate to ▶ *File* ▶ *Extensions* ▶ (the keyboard shortcut is *Ctrl-J*) to open the *Extension Manager* dialog box.
3. Click the *Manual Installation* button located at the lower right of the *Extension Manager* dialog box.
4. From the *Open* file dialog box, navigate to the extension file `<filename>.zip` and double-click it, or click the *Open* button.
5. Verify the extension is listed in the *Extension Manager* as indicated in the following screenshot, click the *Close* button ✖ at the top right of the dialog box, and restart SAP Lumira.



Extension Manager

SAP Lumira Sample Extension Version 1.24.0.201503101453
Created by SAP
for SAP Lumira Data Acquisition Framework
Successful Installation: Restart required.

SAP Universe Query Panel Version 1.24.0
Created by SAP 40 MB
Use Advanced Query Capabilities to access UNX Universes

> Note
>
> The extensions are versioned, so if you have a same-named extension installed, an *Install Extension* alert dialog displays with the message "A newer or equivalent version of the extension "<extension name>" has already been installed. In this case, remove the extension using the *Trash* icon 🗑, and try again, and restart SAP Lumira.

You see the newly added blue puzzle extension with the message "Successful Installation: Restart required."

9.2 Removing a Data Access Extension

To remove a data access extension from SAP Lumira, perform the following steps:

Procedure

1. In SAP Lumira, open ▶ *File* ▶ *Extensions* ◀ (the keyboard shortcut is *Ctrl-J*) to open the *Extension Manager* dialog box.
2. Identify the data access extension to remove, and click the *Trash* icon 🗑, and click *OK* to confirm.
3. Click the *Close* button ✖ at the top right of the *Extension Manager* dialog box. The confirmation message displays: "Successful Uninstallation: Restart required. ⚠
4. Restart SAP Lumira, and check the *Extension Manager* to verify the removal.
   You have removed a data access extension from SAP Lumira.

> Note
>
> The Extension Manager does not have the ability to disable or enable an extension, so if you anticipate reinstalling an extension, you would need to have the original ZIP archive transport unit of that extension.

9.3 Sharing Data Access Extensions

Sharing a data access extension is possible by emailing the extension ZIP file to other SAP Lumira users and instructing them to use the *Extension Manager* to install it.

There may be environmental considerations you should include such as:

- The version number of SAP Lumira

- If SAP Lumira is 32-bit or 64-bit (a 64-bit extension requires a 64-bit SAP Lumara installation)
- A more recent version of the extension will require users to remove older extensions
- An extension may require a specific version of SAP Lumira

9.4 Updating Data Access Extensions

When a new version of a Data Access Extension is available, the SAP Lumira Extension Manager will alert you to remove the old extension first before upgrading to the newer version.

Context

To update a data access extension with a newer version, perform the following steps:

Procedure

1. Refer to Removing a Data Access Extension. (Note that it is NOT necessary to restart SAP Lumira.)
2. Refer to Installing a Data Access Extension, and restart SAP Lumira.
3. Test your visualizations that use the extension to see if there are any problems.
   You have upgraded a data access extension.

9.5 Maintaining Logs

Logs are maintained during the following scenarios:

- Basic errors while opening or closing or reading a data source. For Example: file, URL or any service
- Error that occur during execution of preview, edit, and refresh workflow
- Syntax errors, for example, a mismatch in quoted data objects.

While developing an extension, since the testing has to be done outside SAP Lumira, you must manage any specific errors which occur during the testing process.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of wilful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

www.sap.com/contactsap