HP Enterprise Maps

Software Version: 3.00
Windows and Linux Operating Systems

Developer Guide

Document Release Date: October 2015
Software Release Date: October 2015

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2003-2015 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: **https://softwaresupport.hp.com**

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to: **https://hpp12.passport.hp.com/hppcf/createuser.do**

Or click the **the Register** link at the top of the HP Software Support page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at: **https://softwaresupport.hp.com**

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests

- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

https://hpp12.passport.hp.com/hppcf/createuser.do

To find more information about access levels, go to:

https://softwaresupport.hp.com/web/softwaresupport/access-levels

HP Software Solutions Now accesses the HPSW Solution and Integration Portal Web site. This site enables you to explore HP Product Solutions to meet your business needs, includes a full list of Integrations between HP Products, as well as a listing of ITIL Processes. The URL for this Web site is **http://h20230.www2.hp.com/sc/solutions/index.jsp**

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Chapter 1: In This Developer Guide

This is a *Developer Guide* for *HP Enterprise Maps* and describes additional features and methods to enable developers to better interact with HP EM.

This guide contains the following chapters:

- "Data Model" on page 10
  Describes the Data Model changes.

- "Using DQL" on page 25
  Describes how to use DQL to write queries.

- "Data Sources" on page 46
  Describes how to use data sources that are predefined queries used by reports for visualization and data collection.

- "Scripting" on page 60
  Describes how to extend the current customization framework so that custom UI components can be included in the catalog pages.

- "XML Publishing" on page 105
  Describes how to extend publishing with script artifacts.

- "CSV Import and Export Tools" on page 117
  Describes how to use the CSV Import and Export tools to import and export CSV files.

- "WebDAV Compliant Publishing" on page 130
  Describes how to use WebDav clients with the publishing location space.

- "HP EM Extension for Inkscape " on page 132
  Describes how HP EM integrates with the open source Inkscape vector graphics editing tool via an extension module.

- "Atom-Based REST Interface" on page 136
  Describes the Atom REST Interface.

- "Lifecycle Remote Client" on page 162
  Describes how to use a remote client for lifecycle manipulation.

- "Validation Client" on page 166
  Describes the Validation Client command-line tool for policy compliance validation.

- "Report Creation" on page 168
  Describes how to create reports.

Chapter 2: Data Model

This section covers the following topics:

System Data Model

The System Data Model is a schema describing the hierarchy of artifact types in HP EM.

The model consists of a hierarchy of artifact types with each artifact type defining the set of properties applicable to it. The hierarchy enables properties to be defined for a higher level artifact and then inherited by the artifact types beneath it. Common properties are also organized into property groups which are assigned to artifact types.

The installation directory contains a full description of all the default artifact types, property groups, and properties accessible at `EM_HOME/doc/advanced/sdm/index.html` or `http://host:port/hp-em-doc/advanced/sdm/index.html`.

The documentation provides a set of menus on the left and artifact type, property group, or property descriptions in the main pane on the right.

> **Note:** All menu content uses the artifact type, property, and property group localnames.

Use the top menu to control the content of the menu below using the following links:

- **All Artifacts**

  View the list of all artifacts in the default model split into Public and System models. Artifact types in *italics* are abstract artifact types which do not have instances in the Catalog, but instead act as collective artifact types, such as Implementations, to group artifact types with instances, such as SOAP Services and Web Applications, and for property inheritance purposes.

- **All Properties**

  View the list of all properties in the default model.

- **Property Groups**

  Click a property group to view a list of all the artifact types that use the property group.

Click an artifact type, property, or property group name to view its details in the main pane and view the following sections for details:

- "Artifact Type Documentation" below

- "Property Documentation" on page 13

- "Property Group Documentation" on page 13

Artifact Type Documentation

The documentation for an artifact type displays the following information:

- The title showing the artifact localname. System artifacts are denoted with (system) and abstract artifacts denoted with (abstract).

- The hierarchy of artifact types that the artifact belongs to.

- The property groups applicable to the artifact type. Click a property group to view its details.

- Any directly associated sub-artifacts.

- The description showing the label used in the user interface and a description of the artifact type.

- The set of properties applicable to the artifact type divided into the following:

  - **Properties Summary** - These properties are directly associated with the artifact type in the model.

  - **Properties inherited from property groups** - Lists the set of properties applicable to the artifact defined by property groups assigned to the artifact type. Each applicable property group displays in its own table.

  - **Properties inherited from parent artifacts** - Lists the set of properties inherited from artifact types higher in the hierarchy, Each artifact type displays in its own table.

  - Each table shows the following information about each property:

    - **Name** - Click the property name to view its details.

    - **Type** - The property type. For details, see "Property Types" on page 14.

    - **Cardinality** - The number of times the property can occur for an artifact type with the following possible values:

      - [0..1] - Optional property that can occur only once.

      - [0..*] - Optional property that can occur multiple times.

- [1..1] - Required property that only occurs once.
- [1..*] - Required property that must occur at least once.

Artifact `bacServerArtifact` (system)

```
artifactBase
  '---registryArtifact
      '---bacServerArtifact
```

Implemented property groups

systemProperties

Description

BSM / UCMDB Server — Represents a BSM / UCMDB Server

Properties summary		
Name	Type	Cardinality
+ baseUrl	: nameUrlPair [name, url]	[1..1]
+ encryptedPassword	: text	[0..1]
+ environment	: taxonomy `uddi:systinet.com:soa:model:taxonomies:environments` [val, name, taxonomyURI]	[0..1]
+ runtimePolicy	: relation ⟶ to runtimePolicyOf [wsPolicyArtifact]	[0..*]
+ ucmdbEncryptedPassword	: text	[0..1]
+ ucmdbUsername	: text	[0..1]
+ username	: text	[0..1]

ArchiMate Compliant Relations	
Relation Name	Relation Target Types
assignedTo	diagramArtifact

Properties inherited from property groups

Properties inherited from systemProperties		
Name	Type	Cardinality
+ _deleted	: boolean	[1..1]
+ _domainId	: text	[1..1]
+ _owner	: text	[1..1]
+ _revision	: integer	[1..1]
+ _revisionCreator	: text	[1..1]
+ _revisionTimestamp	: date	[1..1]
+ _uuid	: uuid	[1..1]
+ categoryBag	: categoryBag [categoryGroups.categories, categoryGroups.taxonomyURI, categoryGroups.categories.val, categories.name, categories.taxonomyURI, categories.val, categoryGroups.categories.taxonomyURI, categories, categoryGroups, categoryGroups.categories.name]	[0..1]
+ description	: text	[0..1]
+ identifierBag	: identifierBag [categories.name, categories.taxonomyURI, categories.val, categories]	[0..1]
+ keyword	: taxonomy `uddi:uddi.org:categorization:general_keywords` [val, name, taxonomyURI]	[0..*]
+ name	: text	[1..1]

Properties inherited from parent artifacts

Properties inherited from artifactBase		
Name	Type	Cardinality

Properties inherited from registryArtifact		
Name	Type	Cardinality
+ documentation	: relation ⟶ to documentationOf [documentationArtifact]	[0..*]
+ publishLocation	: text	[0..1]
+ registers	: relation ⟵ from registeredIn [externalEntityArtifact, uddiEntityArtifact, bacEntityArtifact, syncArtifact]	[0..*]

Property Documentation

The documentation for a property displays the following information:

- The title showing the property localname.

- The description showing the label used in the user interface and a description of the property.

- The property type. For more details, see "Property Types" on the next page.

- The set of artifact types and property groups that the property belongs to displaying the following information:

    - **Name** - Click the artifact name to view its details.

    - **Cardinality** - The number of times the property can occur for an artifact type with the following possible values:

        - [0..1] - Optional property that can occur only once.

        - [0..*] - Optional property that can occur multiple times.

        - [1..1] - Required property that only occurs once.

        - [1..*] - Required property that must occur at least once.

Property `environment`

Description

Environment — Separates per environment

Type

taxonomy `uddi:systinet.com:soa:model:taxonomies:environments` [val, name, taxonomyURI]

Declared on artifacts

Name	Cardinality
bacServerArtifact	[0..1]
contractArtifact	[0..*]
hpsoaStmServerArtifact	[0..1]
infrastructureInterface (endpointArtifact)	[0..1]
r_deviceArtifact (r_datapowerXI50Artifact r_l7GatewayArtifact r_oracleServiceBusArtifact r_urlRewritingDeviceArtifact)	[0..1]
ucmdbRepository	[0..1]
uddiRegistryArtifact	[0..*]

Property Group Documentation

The documentation for a property groups displays the following information:

- The title showing the property group localname.

- The description showing the property group label used in the user interface.

- The set of artifact types that the property group applies to. Click an artifact type name to view its details.

- The set of properties in the group displaying the following information:

- **Name** - Click the property name to view its details.

- **Type** - The property type. For details, see "Property Types" below.

- **Cardinality** - The number of times the property can occur for an artifact type with the following possible values:
  - [0..1] - Optional property that can occur only once.
  - [0..*] - Optional property that can occur multiple times.
  - [1..1] - Required property that only occurs once.
  - [1..*] - Required property that must occur at least once.

Property group `c_workInterval`

Work Interval

Declared on Artifacts

programArtifact projectArtifact workPackageArtifact

Properties Summary

Name	Type	Cardinality
+ completion	: integer	[0..1]
+ endDate	: date	[0..1]
+ plannedFinishDate	: date	[0..1]
+ plannedStartDate	: date	[0..1]
+ startDate	: date	[0..1]

Property Types

HP EM uses the following property types.

Property Types

Type	Description
address	A full postal address
boolean	Boolean value - TRUE or FALSE
category	Used to assign several categories from a taxonomy to the artifact
categoryBag	Categorizes an artifact by taxonomies
dailyInterval	A time interval defined by a start day and end day, e.g. Monday to Friday
dateTime	A specific date and time
documentRelationship	Used to reference other artifacts etc.
identifierBag	Identifies the artifact by taxonomy. For categorization, use

Property Types, continued

Type	Description
	category bag instead
instanceDetail	Property type used by UDDI integration.
integer	Integer number
double	A double precision floating point number
nameUrlPair	URL with an optional name assigned
nameValuePair	A name and value pair
plainText	One-line text, suitable for textual information such as names
portDocumentRelationship	Port-document relationship
qnamedDocumentRelationship	Used to reference parts of WSDLs, WS-Policies, etc.
scheduled	Property type used to display scheduling information for task
selector	Property type used to display selector for a task
text	One-line text, suitable for machine readable information such as e-mail addresses
textarea	Multi-line text, suitable for information such as descriptions
xqueryParameter	Property type used to display parameters for executing an XQuery
encryptedPassword	Encrypted Password

Model Changes

The following major changes have been done to the SDM Model between Enterprise Maps 2.00 and Enterprise Maps 3.00:

- **Survey management:** modeling survey questions, answers and data storage.
- **CSA Reference Model:** modeling Cloud Service Automation artifacts.
- **Spreadsheet Template:** new template artifacts for Spreadsheet Import.
- **Cloud Transformation:** extends existing Application Component artifacts to hold Cloud assessment data.
- **PPM extended properties:** extends existing Project and Porfolio Management artifacts.

The following sections list the specific changes to the model:

-
-

- "Property Groups" below
- "Properties" below

Collection Changes

- deploymentArtifacts: now contains new deploymentSpecifications.
- deviceArtifacts: now contains new serverArtifacts, serverClassArtifacts and serverProfileArtifacts.
- locationArtifacts: now contains new dataCenterArtifacts.
- systemSoftwareArtifacts: now contains new operatingSystemArtifacts.

Artifacts

added:

- deploymentSpecification
- dataCenterArtifact
- operatingSystemArtifact
- serverArtifact
- serverClassArtifact
- serverProfileArtifact
- surveyArtifact
- surveyDefinitionArtifact
- csvTemplateArtifact

Property Groups

added:

- c_applicationDeploymentCostEstimation
- c_cloudAssessment_BusinessAlignment
- c_cloudAssessment_TechnicalAlignment
- c_cloudDeploymentCompatibilityScores
- c_serverFinancialProfile
- c_serverHardwareProfile

Properties

added:

- *properties:*
  - estimatedNetworkTraffic

  - estimatedNumberOfServers

  - geographyAlignment

  - regulatoryAlignment

  - securityAlignment

  - serviceAvailabilityAlignment

  - vendorSupportAlignment

  - workloadVariabilityAlignment

  - architectureAlignment

  - cloudOperatingModelAlignment

  - hpCSAAlignment

  - infrastructureRequirementsAlignment

  - internetAccessAlignment

  - performanceRequirementsAlignment

  - businessAlignment

  - financialAlignment

  - technicalAlignment

  - businessAlignmentInformationQuality

  - financialAlignmentInformationQuality

  - technicalAlignmentInformationQuality

  - expectedLifetime

  - perHourCost

  - totalCostOfAcquisition

  - cpuCores

  - ramSize

  - storageSize

- targetContactRole

- deploymentSpecificationType

- networkingCostFormula

- osType

- serverType

- dateStarted

- expectedCompletionDate

- surveyProgress

- surveyStatus

- artifactSelectionCustomizationID

- surveyAccess

- transformationCost

- currentHwCostEstimate

- minEstimatedHwCost

- domainType

- cloudTransformationStrategy

- disposition

- id

- applicationType

- applicationArchitecture

- numberOfUsers

- virtualized

- *relationships:*
  - recommendedDataCenter / recommendToApplications

  - serverClass/ usedByApplicationComponents

  - usedOperatingSystem/ osUsedByApplications

  - defaultServerProfile/ defaultProfileForDataCenter

- surveyDefinition/ surveysDefinitionOf

- surveyedArtifacts/ surveys

- New <attribute label="" mappedTo="" major=""> in relationship annotation

Versioning Schema

HP EM uses versioning strategy configurations in XML format described by the following versioning schema:

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://hp.com/2009/11/systinet/platform/versioning/schema"
    targetNamespace="http://hp.com/2009/11/systinet/platform/versioning/schema"
    elementFormDefault="qualified">

  <xs:complexType name="mask">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:ID" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="maskref">
    <xs:attribute name="ref" type="xs:IDREF"/>
  </xs:complexType>

  <xs:simpleType name="separator">
    <xs:restriction base="xs:string">
      <xs:length value="1"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="group">
    <xs:simpleContent>
      <xs:extension base="xs:positiveInteger">
        <xs:attribute name="separator" type="separator" default="."/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="groupLocation">
    <xs:simpleContent>
      <xs:extension base="group">
        <xs:attribute name="prefix" type="xs:string" default=""/>
        <xs:attribute name="suffix" type="xs:string" default=""/>
```

```
            <xs:attribute name="mandatory" type="xs:boolean" default="true"/>
            <xs:attribute name="primary" type="xs:boolean" default="false"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <xs:complexType name="groups">
      <xs:sequence>
         <xs:element name="group" type="groupLocation" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="maskOrMaskref">
      <xs:sequence>
         <xs:choice>
            <xs:element name="mask" type="xs:string"/>
            <xs:element name="maskref" type="maskref"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="sourceMask">
      <xs:complexContent>
         <xs:extension base="maskOrMaskref">
            <xs:sequence>
               <xs:element name="group" type="group" minOccurs="1"
maxOccurs="unbounded"/>
               <xs:element name="property" type="xs:string"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:simpleType name="operator">
      <xs:restriction base="xs:string">
         <xs:enumeration value="prefix"/>
         <xs:enumeration value="suffix"/>
         <xs:enumeration value="contains"/>
         <xs:enumeration value="full"/>
      </xs:restriction>
   </xs:simpleType>

   <xs:complexType name="other">
      <xs:complexContent>
         <xs:extension base="sourceMask">
            <xs:sequence>
               <xs:element name="match" type="xs:string" minOccurs="0"/>
            </xs:sequence>
```

```
        <xs:attribute name="operator" type="operator" default="prefix"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="source">
    <xs:sequence>
      <xs:element name="version" type="sourceMask"/>
      <xs:element name="other" type="other"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="location">
    <xs:complexContent>
      <xs:extension base="maskOrMaskref">
        <xs:sequence>
          <xs:element name="version" type="groups"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="masks">
    <xs:sequence>
      <xs:element name="mask" type="mask" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="schema">
    <xs:sequence>
      <xs:element name="masks" type="masks" minOccurs="0"/>
      <xs:element name="source" type="source"/>
      <xs:element name="location" type="location"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:element name="versioningSchema" type="schema"/>

</xs:schema>
```

Policy Artifacts

Policy Manager entities (Technical Policy and Assertion) are represented in HP EM by artifacts. These artifacts are Technical Policy Artifact (hpsoaTechnicalPolicyArtifact) and Assertion Artifact (assertionArtifact).

Policy Manager entities are related to each other. A technical policy can reference both technical polices and assertions. The references are included in the entity data. The technical policy data contains references to other technical policies and to assertions. Since the entity data is in the artifacts, the references are included in the artifact data.

In addition these references between policy manager entities are represented by relations between Policy Manager artifacts. The details of the relations are listed in the following table.

Source (Referencing) Artifact	Outgoing Relation Property on Source	Incoming Relation Property on Target	Target (Referenced) Artifact
hpsoaTechnicalPolicy Artifact	r_ referencedTechnical Policy	r_ referencedTechnicalPolic yInverse	hpsoaTechnicalPolicy Artifact
hpsoaTechnicalPolicy Artifact	r_ referencedAssertion	r_ referencedAssertionInvers e	assertionArtifact

The relations are created during the creation and update of the source artifact. The relations just reflect the references in the data. In the case that a user creates or deletes relations, the references in the artifact data stay unchanged. This creates an inconsistency between the references in the data and the relations between artifacts. The relations are created, based on the references, when the source artifact is saved (created or updated).

Policy Schema

The policy schema structure defines technical policies. HP EM uses the WS-Policy specification as a modeling framework for technical policies. Technical policies are prepared by Architects and Policy Developers who codify them as requested by the line-of-business managers, architectural councils, operational managers, etc.

Note: In WS-Policy terms, a technical policy = WS-Policy + name + documentation. HP EM policies are covered by the WS-PolicyAttachment specification.

Policy structure consists of a `wsp:Policy` element wrapping any number of assertion elements, as shown in "Policy Definition Structure". HP EM does not support `wsp:All` or `wsp:ExactlyOne`.

Policy Definition Structure

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:ex="http://www.example.com/assertions">
  <ex:Assertion1/>
  <ex:Assertion2 param="value" />
  <ex:Assertion3/>
  <ex:Assertion4/>
</wsp:Policy>
```

How to Edit Relationship Attribute Annotation

1. Go to path `EM_HOME/extensions/com.hp.soa.systinet.ext.core.jar/EXTENSION-INF/sdm/sdmConfig.xml`.

2. Find <artifactDescriptor> that has uri matches artifact type we want to edit its relationship annotation.

3. Navigate to <annotation> which has 'relationships' in the inner <name>.

4. Edit relationship annotation in the inner <value>.

5. Stop server.

6. Run Setup tool "Apply Extensions" scenario.

7. Start server.

> **Caution:** Relationship annotation value must be in encoded form.
>
> For instance to change the value of 'mappedTo' (attribute of 'accesses' relationship between Application Service and Data Object artifacts) from 'description' to 'useType', follow the encoded annotation below:
>
> ```
> &lt;relationships
> xmlns=&quot;http://ea.manager.hp.com/2013/04/model/annotations&quot;
> xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;&gt;
> &lt;accesses label=&quot;Accessed Data Objects&quot;
> priority=&quot;600&quot; localName=&quot;dataObjectArtifact&quot;
> skeleton=&quot;true&quot;&gt;
>     &lt;attribute label=&quot;Method&quot;
> mappedTo=&quot;description &quot; major=&quot;true&quot;/&gt;
>     &lt;/accesses&gt;
> &lt;relationships;
> ```
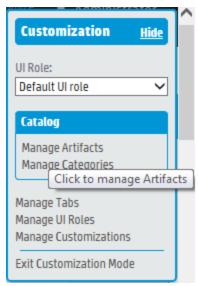
Then replace with following annotation:
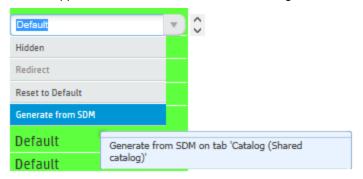
```
&lt;relationships
xmlns=&quot;http://ea.manager.hp.com/2013/04/model/annotations&quot;
xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;&gt;
&lt;accesses label=&quot;Accessed Data Objects&quot;
priority=&quot;600&quot; localName=&quot;dataObjectArtifact&quot;
skeleton=&quot;true&quot;&gt;
    &lt;attribute label=&quot;Method&quot;
mappedTo=&quot;useType &quot; major=&quot;true&quot;/&gt;
    &lt;/accesses&gt;
&lt;relationships;
```

Thereafter, follow the instructions given below to update the customization of Application Service artifact:

1. In Customization menu, click **Customize UI** and select **Manage Artifacts**.

2. Locate Application Service artifact. In the Catalog column select **Generate from SDM** option.

3. Click **Apply** button at the bottom of the page.

Changes in SDM are propagated to UI as follows:

```
<component componentName="/core/layout/relationshipItemComponent" id="accesses_4_
ItemComponent" xmlns:c="http://soa.systinet.hp.com/2009/02/ui/customization">
    <parameter name="relationshipName">accesses</parameter>
    <parameter name="label">Accessed Data Objects</parameter>
    <parameter name="artifactTypes">dataObjectArtifact</parameter>
    <parameter name="attrLabels">Method </parameter>
    <parameter name="attrMappedTargets">useType </parameter>
    <parameter name="attrMajors">true </parameter>
    <parameter name="attrTypes">string </parameter>
</component>
```

Note: attrTypes `parameter` (comma-separated values) shows data types of relationship attributes in sequence. It is obtained through **Generate from SDM** selection and cannot be edited.

Chapter 3: Using DQL

The DQL query language provides a simple query solution for the System Data Model (SDM). It enables you to query all aspects of the model – artifacts, properties, relationships, governance, and compliance.

This chapter describes DQL in the following sections:

- "Introduction to DQL" below
- "DQL Reference" on page 34
- "DQL With Third-Party Products" on page 43

Introduction to DQL

DQL is an SQL-like language that enables you to query the repository of artifacts in HP EM defined by the SDM model. DQL preserves SQL grammar, but uses artifacts instead of tables, and artifact properties instead of table columns. As DQL is based on SQL you can apply your SQL knowledge to DQL.

A simple example is to return the name and description of all business service artifacts.

```
select name, description
  from businessServiceArtifact
```

In HP EM, you can use DQL queries in the following use cases:

- To create reports in HP EM Report Editor. For details, see the *HP Enterprise Maps Workbench - Report Editor Editor Guide*.
- You can also use DQL in any SQL designer using the DQL JDBC driver. For more details, see "DQL in SQL Designers" on page 44

The following sections contain DQL examples:

- "Primitive Properties" on the next page
- "Complex Properties" on the next page
- "Artifact Inheritance" on the next page
- "Categorization Properties" on page 27
- "Fixing Multiple Properties" on page 28
- "Relationships" on page 29
- "Shortcuts" on page 30
- "Modifiers" on page 31
- "Virtual Properties" on page 32
- "Embedding SQL Queries" on page 33

Primitive Properties

Primitive properties are simple properties, such as numbers, characters, and dates, that may occur once or multiple times for an artifact depending on the cardinality as defined in the SDM.

For example, in the SDM Model, each person is represented by a person artifact. The person artifact includes a name property with single cardinality and an email property with multiple cardinality.

The following query returns the name and all emails for each person in the repository.

```
select name, email
  from personArtifact
```

Instances of primitive properties with multiple cardinality are all returned as comma separated values. For example, all the emails for a person return as a concatenated, comma-separated string. If there is no instance of the property for an artifact, a null value is returned.

The following query returns the name, description, and version of all business service artifacts whose version is 2.0.

```
select name, description, version
  from businessServiceArtifact
  where version = '2.0'
```

Note: By default, DQL queries return the latest revisions of artifacts unless you specify revision modifiers. For details, see "Modifiers" on page 31.

Complex Properties

Complex properties are composed of one or more single or multiple-valued sub-properties (for example, address contains sub-properties addressLines in multiple cardinality, country in single cardinality, etc. The sub-property addressLines is also a complex sub-property, containing a value and useType.). It is only possible to query the sub-property components of primitive types. Components of sub-properties are separated by **.** (in MS Access you can use **$** as a separator).

```
select address.addressLines.value, address.country
  from personArtifact
  where address.city = 'Prague'
```

For a full reference of all complex properties in the default SDM, see "System Data Model" on page 10.

Artifact Inheritance

Artifacts in HP EM form a hierarchy defined by the SDM model. Artifacts lower in the hierarchy inherit properties from higher abstract artifact types. `artifactBase` is the root abstract artifact type in the SDM hierarchy. All other artifacts are below it in the hierarchy and inherit its properties. You can query abstract artifacts and return a result set from all the instances of artifact types lower in the hierarchy.

Property groups function in a similar way, querying a property group returns results from all artifact types that inherit properties from the group.

The following query returns results from all implementation artifacts; SOAP Services, XML Services, and Web Applications.

```
select name, serviceName
  from implementationArtifact
```

Notice that in this query, serviceName is a specific property of SOAP Service artifacts. In the result set, name is returned for all implementation artifacts but serviceName is only returned for SOAP service artifacts. For other implementation types, the serviceName is NULL.

> **Caution:** Different artifact types may define the same properties with different cardinalities. In cases where two artifact types define the same property with different cardinality, querying a shared parent abstract artifact for these properties may fail. Examples that fail include **SELECT environment FROM artifactBase** and **SELECT accessPoint FROM artifactBase**.

Categorization Properties

Categorization properties are a special case of complex properties.

Categorization properties have the following sub-properties:

- `val` - machine readable name of the category.
- `name` - human readable name of the category.
- `taxonomyURI` - identifies the taxonomy defining the category set.

  > **Note:** TaxonomyURI is not defined for named category properties.

HP EM uses categorization properties in the following ways:

- **Named category properties** (for example, business service criticality).

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized using the named criticality categorization property with a high failure impact.

  ```
  select name, description, version
    from businessServiceArtifact
    where criticality.val =
          'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
  ```

  > **Note:** TaxonomyURI is not defined for named category properties. The name of the category property implies the taxonomy.

- **categoryBag**

  `categoryBag` is a complex property that includes sub-property `categories` which is a categorization property and `categoryGroups`. `categoryGroups` also contains categorization sub-property `categories` and a `taxonomyURI` defining the meaning of the group. HP recommends querying _ category instead of `categoryBag` to ensure that all categories are queried.

The following query returns the names, descriptions, and versions of all business service artifacts which are categorized by the Gift certificate category (14111608) of the uddi:uddi.org:ubr:categorization:unspsc taxonomy.

```
select name, description, version
  from businessServiceArtifact
  where categoryBag.categories.taxonomyURI =
        'uddi:uddi.org:ubr:categorization:unspsc'
    and categoryBag.categories.val = '14111608'
```

- **identifierBag**

  identifierBag is a complex property similar to categoryBag that includes sub-property categories. identifierBag does not contain the categoryGroups subproperty. HP recommends querying _category instead of identifierBag to ensure that all categories are queried.

- **_category**

  This generic categorization property holds all categorizations from categoryBag, identifierBag, and all named categorization properties from the given artifact type.

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized with a high failure impact.

```
select name, description, version
  from businessServiceArtifact
  where _category.val =
        'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
    and _category.taxonomyURI =
        'uddi:systinet.com:soa:model:taxonomies:impactLevel'
```

> **Caution:** When you use the generic _category property you must specify the taxonomy using the _category.taxonomyURI sub-property. When you use a named categorization property the taxonomy is implicitly known and does not need to be specified.

Fixing Multiple Properties

Consider a business service with keywords, 'Finance' and 'Euro'. The intuitive query for finding a 'Euro Finance' service is as follows:

```
select name, description, version
  from businessServiceArtifact b
  where b.keyword.val = 'Finance'
    and b.keyword.val = 'Euro'
```

This query does not work as a single instance of keyword can never be both 'Finance' and 'Euro'

The solution is to fix instances of multiple properties as shown in the following query:

```
select name, description, version
  from businessServiceArtifact b, b.keyword k1, b.keyword k2
  where k1.val = 'Finance'
```

```
    and k2.val = 'Euro'
```

Relationships

A relationship is a special kind of complex property pointing to another artifact. HP EM uses relationships to join artifacts.

The following queries are semantically identical and return all business services and the contact details of their provider. These queries do not return business services that do not have providers.

- The following query is an example of an *SQL92-like* join which uses the USING clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p using provides
```

The relationship property `provides` leads from person artifacts to business service artifacts is specified after the `using` keyword.

- The following query is an example of an *SQL92-like* join which uses the ON clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p on bind(provides)
```

The relationship property `provides` leads from person artifacts to business service artifacts is specified with the `bind` predicate in the `WHERE` clause.

- The following query is an example of an *old-style* join which uses the BIND predicate.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
  where bind(p.provides, b)
```

The BIND predicate specifies that the provides relationship of the person artifact points to business service artifacts.

The following query also returns all business services and the contact details of their provider. This query is an example of a LEFT JOIN. The LEFT JOIN extends the previous queries by also returning business services that do not have providers.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  left join personArtifact p using provides
```

Each relationship has the following sub-properties which you can query:

- rType - the SDM QNames of the relationship type.
- useType - the values of the useType relationship property
- target - the UUIDs of the artifact the relationship points to (deprecated).

It is possible to specify a particular provider type using useType. The following queries return all business services and their contact details where the provider is an architect.

```
select b.name, b.version, p.name as contact, p.email
```

```
  from businessServiceArtifact b, personArtifact p
 where bind (p.provides, b)
   and p.provides.useType = 'architect'
```

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p on bind(p.provides, b)
    and p.provides.useType = 'architect'
```

It is possible to traverse several relationships using several old-style joins or SQL-92-like join clauses in the same query. The following example queries business services in applications, which are also part of a project.

```
select b.name, b.description, a.name as Application, p.name as Project
  from businessServiceArtifact b
  join hpsoaApplicationArtifact a using hpsoaProvidesBusinessService
  join hpsoaProjectArtifact p using contentRelationshipType
```

In cases where artifacts may be joined by multiple properties, you can use a generic _relation property together with the additional rType condition.

```
select A.name as A_name, B.name as B_name
  from hpsoaApplicationArtifact A left join artifactBase B on bind(A._relation)
    and A._relation.rType in (
        '{http://systinet.com/2005/05/soa/model/property}
hpsoaProvidesBusinessService',
        '{http://systinet.com/2005/05/soa/model/property}r_providesBusinessProcess'
      );
```

You can use the target relationship sub-property to bind the source and target of a relationship.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
    where p.provides.target = b._uuid
```

Caution: The target property and this style of comparison is deprecated and its use is not recommended. Use the bind predicate instead.

Shortcuts

HP EM makes it possible to use shortcut in DQL to query the artifacts defined in shortcut definition.

As given below the queries are semantically identical and return names of application components defined in the shortcut definition:

```
<shortcut id="projectToAppShortcut" relationshipType="realizes" autoApprove="true"
label="Realized components" inverseLabel="Realized by projects"
delayedCalculation="true">
```

```
    <source localName="projectArtifact" relationshipType="realizes" />
    <intermediate localName="deliverableArtifact" relationshipType="realizes" />
    <target localName="applicationComponentArtifact" />
</shortcut>
```

Using Shortcut Id

- The following query is an example of an *SQL92-like* join which uses the USING clause.

```
select b.name
  from projectArtifact a
  join applicationComponentArtifact b using a.projectToAppShortcut
```

- The following query is an example of an *SQL92-like* join which uses the ON clause.

```
select b.name
  from projectArtifact a
  join applicationComponentArtifact b on bind(a.projectToAppShortcut)
```

Using Relationship Type

- The following query is an example of an *SQL92-like* join which uses the USING clause.

```
select b.name
  from projectArtifact a
  join applicationComponentArtifact b using a.realizes (shortcut)
```

- The following query is an example of an *SQL92-like* join which uses the ON clause.

```
select b.name
  from projectArtifact a
  join applicationComponentArtifact b on bind(a.realizes) (shortcut)
```

Modifiers

Modifiers define primary sets of objects (artifacts and their revisions) to query. If no modifier is specified, the last revisions of undeleted artifacts for which the user has read access are queried.

The following modifiers are available:

- Revision related modifiers (mutually exclusive):

  - **all_rev** - queries all revisions of artifacts.

  - **last_approved_revision** - queries the last approved revisions of artifacts.

- Security related modifiers (mutually exclusive):

  - **my** - queries artifacts belong to the user.

  - **writable** - queries artifact the user has write permission for.

  - **no_acl** - queries all artifacts regardless of security.

- Shortcuts related modifiers:
  - relation - relationship type is used to query relationship.

  - shortcut - relationship type is used to query shortcut.

- Other modifiers:
  - **include_deleted** - queries all instances, including deleted artifacts.

You can use multiple comma-separated modifiers.

The following query returns all business services that you own that are marked as deleted.

```
select b.name, b.version, b.keyword.name
  from businessServiceArtifact b (my, include_deleted)
  where _deleted = '1'
```

Virtual Properties

DQL defines virtual properties, that are not defined by the SDM. HP EM stores or calculates these properties enabling DQL to query meta information about artifacts. These virtual properties provide information about lifecycle, compliance, domains, etc.

The following example returns lifecycle details from the last approved revisions of all business service artifacts, ordered by lifecycle stage.

```
select name, _lastApprovedStage.name Stage, _revision
  from businessServiceArtifact(last_approved_revision)
  order by Stage
```

The following example returns the name and compliance status of last approved revisions of all business services which a compliance status of at least 80%.

```
select b.name, b._complianceStatus
  from businessServiceArtifact b (last_approved_revision)
  where b._complianceStatus >= 80
```

HP EM repository content exists within a domain structure where each artifact exists within only one domain. The default functionality of DQL queries all domains but HP EM provides virtual properties enabling you to query artifacts within a particular domain. The following example returns business service names and the domain details of all business service artifacts that exist within the EMEA domain.

```
select A.name, A._domainId, A._domainName
  from businessServiceArtifact A
  where A._domainId="EMEA"
```

DQL provides the following macros for querying within domain hierarchies:

- **#SUBDOMAINS('*domainId*')**

  Queries the specified domain and all its sub-domains.

- **#SUPERDOMAINS('*domainId*')**

  Queries the specified domain and all its parent domains.

The following query returns all business services in the EMEA domain and any of all of its sub-domains.

```
select A.name
```

```
from businessServiceArtifact A
where A._domainId in #SUBDOMAINS('EMEA')
```

The following query returns the name and virtual properties artifactTypeName and owner from the latest revisions of consumer properties (the property group for all consuming artifact types).

```
select name, _artifactTypeName, _owner
  from consumerProperties
```

For details of all virtual properties, see "Properties in DQL" on the next page.

Embedding SQL Queries

DQL works with SDM entities (artifacts and properties) only and cannot directly access database tables. In some cases it is necessary to obtain values from outside the SDM (for example, system configuration). You can use an SQL subquery in a NATIVE clause of a DQL query. By default, DQL expects SQL to return an unnamed single column of values.

The following example returns business services owned by the administrator using the name defined during installation:

```
select name,description, version
  from businessServiceArtifact
  where _owner in (
    native {select svalue from systemConfiguration
             where name='shared.administrator.username'})
```

You can use NATIVE clauses instead of expressions, as a condition in WHERE clauses, as a column in SELECT clauses, and as a artifact reference in FROM clauses. For details, see "DQL Grammar" on page 37 .

If you use a NATIVE clause to formulate part of a FROM clause, you must specify parameters to bind columns defined by SQL to properties used by DQL.

Each parameter consists of the following:

- The property name defines how DQL addresses columns returned from the NATIVE SQL statement.
- The property type which may be returned by the metadata of a column is optional and if not specified is assumed to be a text string.

The parameters are enclosed in brackets in the native clause, delimited by commas, and the type is separated from the name using whitespace.

The following example shows a query with NATIVE SQL in a DQL FROM clause.

```
select B.p_id, B.s_val, A.name, B.state_index
  from (
    native(s_val, s_name, state_index integer, p_name, p_id)
      {select S.val as s_val, S.name as s_name, S.state_index as state_index,
              P.name as p_name, P.id as p_id
         from rylf_state S, rylf_process P
         where S.fk_rylf_process=P.id and P.name='Application Lifecycle'}) B
  left join artifactBase A on A._currentStage.val = B.s_val
```

```
order by B.p_id, B.state_index
```

The NATIVE statement returns the following columns; `s_val`, `s_name`, `p_name`, and `p_id` of type String, and `state_index` of type Integer.

> **Note:** Native clauses can not contain variables (? or :<variable>).

DQL Reference

This section provides a reference to properties and DQL grammar in the following sections:

- "Properties in DQL" below
- "DQL and SQL" on page 37
- "DQL Grammar" on page 37

Properties in DQL

Artifact (property group) properties hold values which may be queried in DQL expressions.

DQL recognizes the following properties:

- SDM Properties

  Properties defined in the SDM Model. For details, see "System Data Model" on page 10.

- Virtual, System, and Other Properties

  Properties holding metadata about artifact instances.

Properties may be one of the following:

Property Kind	Description
Primitive	Holds string, number, or boolean values. For example, name, description, version. A primitive property is defined in DQL statements by the artifact type name or alias, the property delimiter (. or $), followed by the property name. For example, personArtifact.name. The artifact name or alias is optional (with the delimiter) when the property is specific to a single artifact type in the query.
Complex	Hold complex structures such as address. Only primitive sub-properties of complex properties may be queried. Properties and sub-properties are separated by `.` or `$`. For example, `personArtifact.address.city`.
Categorization	Hold categorization data and are handled in a similar way to complex properties. Categories consist of `name`, `val`, and `taxonomyURI` components. For example, `businessServiceArtifact.criticality.name`.
Relationships	Properties that specify a directional relationship to other artifacts.

All values that you can query are of a particular data type. The following table describes these data types, and gives examples of how to use them in a query.

Data Type	Description	Example
Number	Numeric values	_revision = 1
String	Text values	_revisionCreator = 'admin'
Date Time	Date and Time (ms since 00:00 1/1/1970)	_revisionTimestamp > 1274447040124 /* revisions made since 15:04 21/5/2010 */
Boolean	True or False flags	_deleted = '1'

Properties may have the following cardinalities:

Property Cardinality	Description
Single	Only one instance of the property exists for an artifact and it may be optional or required.
Multiple	The property is a list of values and so occurs multiple times for an artifact. In WHERE clauses, using multiple properties returns particular artifacts if any instance of the multiple property matches the condition. In SELECT clauses, using multiple properties returns all instances of the multiple property as a concatenated, comma-separated string.

DQL uses the following system properties:

System Property	Description
_artifactTypeName	The human readable name of the artifact type (the SDM label of the artifact). HP recommends using _sdmName in conditions, and _ artifactTypeName in SELECT clauses.
_category	All categorizations for an artifact with `name`, `val`, and `taxonomyURI` components.
_deleted	The deletion marker flag (boolean).
_id	The database ID of an artifact instance (number, deprecated - use _uuid).
_longDescription	HP EM supports a long description including HTML tags up to 25000 characters by default. HP recommends using the `description` property in DQL queries instead as queries using _longDescription may affect performance and the HTML tags may corrupt report outputs. `description` contains only the first 1024 text characters of `_longDescription` (may vary according to your database type). **Note:** The `platform.repository.max.description.length` property determines the maximum length of _longDescription. You can modify this property in `EM_HOME/conf/setup/configuration-propeties.xml`.
_owner	The user, group, or role designated as the artifact owner.

System Property	Description
_ownerName	The human readable name of the user (taken from the use profile), group, or role artifact.
_path	Legacy REST path of the artifact (string, deprecated - use _uuid).
_relation	A generic virtual property that may be used to specify all outgoing relationships.
_revision	The revision number of an artifact instance.
_revisionCreator	The user who created the revision of the artifact.
_revisionTimestamp	The date and time the revision was created.
_sdmName	The local name of the artifact type. HP recommends using _sdmName in conditions, and _artifactTypeName in SELECT clauses.
_uuid	The unique artifact indentifier.

DQL uses the following virtual properties:

Property Class	Property	Description
UI Property	_isFavorite	Marked by the user as favorite flag (boolean).
	_rating	The average rating of the artifact (double).
Security Property	_shared	Indicates that the artifact is shared and visible to users in the Sharing Principal role (boolean). For more details, see "How to Share Artifacts" in the *User Guide*.
	_writable	User write permission flag (boolean). **Note**: Using this property may have a performance impact. If possible, use the **writable** modifier instead. For details, see "Modifiers" on page 31.
Contract Property	_enabledConsumer	The artifact is a valid consumer artifact type.
	_enabledProvider	The artifact is a valid provider artifact type. The artifact must also be marked as 'Ready for Consumption'.
Domain Property	_domainId	Value of the domainId property defined for the domain artifact that the artifact belongs to.
	_domainName	The readable name of the domain.

Lifecycle Property	_currentStage	Current working stage of an artifact.
	_governanceProcess	process applicable to the artifact.
	_isApproved	Lifecycle approval flag (boolean).
	_lastApprovedRevision	Revision number of the last approved revision (number).
	_lastApprovedStage	The name of the last approved stage.
	_lastApprovalTimestamp	Timestamp for the last approval (number, ms since 00:00 1/1/1970).
	_lifecycleStatus	The status of the current lifecycle stage.
Policy Manager Property	_complianceStatus	Lifecycle compliance. Defined as a percentage of successful technical policy validations performed in the current lifecycle stage.

DQL and SQL

DQL supports most features of SQL with the following exceptions:

- SELECT * is not supported.
- RIGHT and FULL OUTER JOIN are not supported.
- It is not possible to use properties with multiple cardinality in GROUP BY, HAVING, or ORDER BY clauses.

DQL Grammar

A DQL query consists of the following elements with their grammar explained in the following sections:

- "Select" on the next page
- "FROM Clause" on page 39
- "Conditions" on page 39
- "Expressions" on page 41
- "Lexical Rules" on page 42

Typographical Conventions

Convention	Example	Description
KEYWORDS	**SELECT**	A reserved word in DQL (case-insensitive).
parsing rules	*expr*	Name of a parsing rule. A parsing defines a fragment of DQL which consists of keywords, lexical rules, and other parsing rules.
LEXICAL RULES	*ID*	Name of a lexical rule. A lexical rule defines a fragment of DQL which consists of letters, numbers, or special characters.
[ ]	[**AS**]	Optional content.

Convention	Example	Description
[...]	[, *select_item*, ...]	Iterations of optional content.
|	**ASC | DESC**	Alternatives.
{}	{ + | - }	Group of alternatives.
..	0..9	A range of allowable characters.

Select

```
select :
  subquery [ ORDER BY                              order_by_item [, order_by_
item ...]]

subquery :
  subquery [ set_operatorsubquery ...]
  | (subquery)
  | native_sql
  | subquery_base

subquery_base :
  SELECT [ DISTINCT ] select_item [, select_item ...]
  FROM                               from_clause_list
  [ WHERE                              condition ]
  [ GROUP BY                          expression_list
    [ HAVING                          condition ]
  ]

select_item :
  expr [ [ AS ] alias ]

alias :
  ID | QUOTED_ID

order_by_item :
  expr [ ASC | DESC ]

set_operator :
  UNION ALL | UNION | INTERSECT | EXCEPT

native_sql :
  NATIVE [ (column_name [ column_type ] [ , ... ] ) ]
  { sql_select }
```

Explanation:

- The { } around the sql_select are required and sql_select is an SQL query.
- The column_name and column_type specify parameters to pass from the SQL query to the DQL query.

FROM Clause

```
from_clause_list :
  { artifact_ref | subquery_ref | fixed_property | native_sql }
  [ from_clause_item ... ]

from_clause_item :
  , { artifact_ref | subquery_ref | fixed_property | native_sql }
  | [ LEFT [ OUTER ] ] JOIN
    { artifact_ref | subquery_ref } join_condition

artifact_ref :
  artifact_name [ alias ] [ (artifact_modifiers) ]

subquery_ref :
  (subquery)alias

fixed_property :
  property_refalias

artifact_modifiers :
  ID [ ,ID ... ]

artifact_name :
  ID

join_condition :
  | USINGproperty_ref
```

Conditions

```
condition :
  condition_and [ OR                                    condition_and ... ]

condition_and :
  simple_condition [ AND                          simple_condition ... ]

simple_condition :
  (condition)
  | NOT                            simple_condition
  | exists_condition
  | like_condition
  | null_condition
  | in_condition
  | simple_comparison_condition
  | native_sql
```

```
 | bind

simple_comparison_condition :
  exprcomparison_opexpr

comparison_op :
 = | <> | < | > | <= | >=

like_condition :
  expr [ NOT ] LIKE                              like_expression [ ESCAPE
                          STRING ]

like_expression :
  STRING
  | variable_ref

null_condition :
  expr                                IS [ NOT ] NULL

in_condition :
  expr [ NOT ] IN( { subquery | expression_list } )
  | macro

exists_condition :
  EXISTS(subquery)

bind :
  BIND(property_ref [ , alias ] )

macro :
  macro_name [ (expression_list) ]

macro_name :
  #ID
```

Explanation:

- Conditions can be evaluated to true, false, or N/A. *condition* consists of one or more *condition_and* that are connected by the **OR** logical operator.

- *condition_and* consists of one or more *simple_condition* connected by the **AND**

- *simple_condition* is one of following:

  - *condition* in parentheses.

  - Negation of *simple_condition*.

  - *exists_condition*

  - *like_condition*

- *null_condition*

- *in_condition*

- *simple_comparison_condition*

- *native_sql*

- *simple_comparison_condition* is a comparison of two expressions using one of the comparison operators: =, <>, <, >, <=, >=
- *like_condition* compares an expression with a pattern. Patterns can contain wildcards:
  - _ means any character (including numbers and special characters).

  - **%** means zero or more characters (including numbers and special characters).

  - **ESCAPE** <char> where <char> is used to prefix _ and % in patterns so that these characters are interpreted as they are and not as wildcards.

    Note: <char> must be a single character. For example: '!'.

- *alias* references the target artifact.

Expressions

```
expr :
  term [ { + | - | CONCAT } term ... ]

term :
  factor [ { * | / } factor ... ]

factor :
  (select)
  | (expr)
  | { + | - } expr
  | case_expression
  | NUMBER
  | STRING
  | NULL
  | function_call
  | variable_ref
  | property_ref
  | native_sql

case_expression :
  CASE                                      case_item [ case_item ... ]
    [ ELSE                                      expr ]
  END

case_item :
  WHEN                                      condition
```

```
  THEN                                          expr

function_call :
  ID( [ DISTINCT ] { [ * ] | [ expression_list ] } )

property_ref :
  { ID | QUOTED_ID } [ { . | $ } { ID | QUOTED_ID } ... ]

expression_list :
  expr [ ,expr ... ]

variable_ref :
  ? | :ID
```

Explanation:

- Variables are of two kinds:
  - Positional variables - ? in DQL.

  - Named variables - :<name_of_variable>

- When variables are used in DQL, each variable must have a value bound to the variable.

Lexical Rules

```
CONCAT :
  ||

STRING :
  [ N | n ] ' text '

NUMBER :
  [ [ INT ] . ] INT

INT :
  DIGIT [ DIGIT ... ]

DIGIT :
  0..9

ID :
  CHAR [ { CHAR | DIGIT } ... ]

CHAR :
  a..z | A..Z | _
```

Explanation:

- *ID* is sequence of characters, numbers and underscores beginning with a character or underscore.
- *QUOTED_ID* is text in quotes.
- *CONCAT* means a concatenation of strings - syntax **||**

DQL With Third-Party Products

DQL is provided by a JDBC driver which you can use with common SQL designers supporting 3rd-party JDBC drivers (or ODBC with an ODBC-JDBC bridge).

The following sections describe the driver and its use with 3rd party products:

- "DQL JDBC Driver" below
- "DQL in SQL Designers" on the next page
- "DQL in MS Access" on the next page

DQL JDBC Driver

The DQL JDBC driver translates DQL queries into SQL queries and executes them using the underlying JDBC driver for the used database. The translation is provided by a remote invocation of HP EM.

All the required JAR files for the DQL driver are available in `EM_HOME/client/lib/jdbc`:

- `pl-dql-jdbc.jar`
- `hessian-version.jar`
- Database driver JAR files are copied here during installation (for example, `ojdbc6.jar`).

The following table describes the driver configuration required to use the driver with third-party products.

DQL JDBC Driver Configuration

Property	Description
Connection String	jdbc:systinet:http(s)://<username>@<host:port>/<context>[|schema=schema name][|model=list of allowed models] I • `<username>` is the Enterprise Maps username who executes the DQL query using Enterprise Maps permissions security. • <host:port> are the connection details of your Enterprise Maps installation (for example, localhost:8080 for HTTP or secure:8443 for HTTPS. • <context> is the application server context, the default is soa. |schema=schema name is the schema of the user who owns HP Enterprise Maps database tables. This parameter is optional. When omitted it is supposed that the user account used to access the database is also the owner of HP Enterprise Maps tables. In case a common user or read-only user is used, use the power user schema name, unless the DQL JDBC Driver cannot provide metadata regarding artifacts and properties. |model=list of allowed models is optional and represents a comma-separated list of models. Only artifacts from allowed models are provided in JDBC metadata as tables. The available models are sys and public. By default, only artifacts from the public model are provided.

Property	Description
	For example, jdbc:systinet:http://admin@demoserver.acme.com:8080/soa|schema=SOA320
DB Credentials	The database username and credentials used for direct access to the HP EM database. In most cases it is the user who owns all tables for HP EM - called the *power user*. In case of "Manual Database Arrangement" with a power user and a common user (who has only read/write access to tables, but can not create other tables), use the common user account. In case the common user is still too powerful to be shared, the DB administrator can create another - "read-only user" with read-only access to HP EM tables. Note that the read-only user must also have created synonyms/aliases for HP EM tables to pretend that HP EM tables are in the schema of the read-only user. For more details, see *Installation and Configuration Guide*, section *Database Installation Types* under *Preparing Databases* .
DQL JDBC Classname	com.hp.systinet.dql.jdbc.DqlDriver

Note: The DQL JDBC driver must be able to connect to the database from the client. Use the full hostname for your database used during installation or setup. In the event of connection problems, verify the firewall settings between the local server and the database server.

DQL in SQL Designers

SQL Designer software can use the DQL driver if the designer is JDBC-aware.

To configure a JDBC-aware SQL Designer:

1. Add the DQL JDBC JAR files to the classpath.

2. Create a JDBC connection using the properties described in "DQL JDBC Driver Configuration" on the previous page.

After you establish the DQL JDBC connection, the following functionality should be available in your SQL Designer:

- Schema introspection, browsing the list of artifact types and property groups as tables, and their properties as columns.

- DQL query execution.

DQL in MS Access

MS Access 2007 can execute DQL queries using an ODBC-JDBC bridge. Before using MS Access, you must configure the ODBC datasource in Windows.

To configure an ODBC-JDBC bridge:

1. Download and install an ODBC-JDBC bridge. For example, *Easysoft ODBC-JDBC Gateway*.

2. Configuration typically consists of:

- JDBC driver configuration using the properties described in "DQL JDBC Driver Configuration" on page 43.

- Bridge configuration. For details, see the documentation for the bridge software.

DQL syntax varies from the examples given in "Introduction to DQL" on page 25 in the following cases:

- Complex properties must use $ notation and be enclosed by [ ].

  `personArtifact.[address$addressLines$value], personArtifact.[address$country]`

- To use modifiers such as (include_deleted) use the Pass-Through option in MS Access.

- Left Joins do not work. Use plain joins instead.

- For fixed properties, use the Pass-Through option in MS Access.

- For timestamps, use the Pass-Through option in MS Access.

- Native queries do not work in MS Access.

- For property aliases, do not use quoted aliases.

Chapter 4: Data Sources

Data sources are predefined queries wherein their results are consumed by reports for visualization. The advantage of this concept is that the data visualization and data collection process is separated and a single data source can be used by multiple reports.

Data sources are defined using the **Administration** tab **> Customization > Manage Scripts > Data sources** option in the UI.

There are two basic types of data sources:

- "DQL-Based Data Sources" on the next page
- "Closure Definition-Based Data Sources" on page 48

Data Sources Parameters

The query results of a data source can be customized by parameters. A typical example is to limit the data in a report to show only artifacts from particular domain(s).

Parameter	Description	Example
domainTypes	- List of domainTypes separated by comma. For example: Reference Models, As-is/To-be Architecture, Demo Data. - Can be used with domainIDs simultaneously. The result is an intersection. For example: - Domain A (type: Reference Models, id: domA) - Domain B (type: Reference Models, id: domB) - Domain C (type: Demo Data, id: domC) If domainTypes='Reference Models' and domainIDs='domA, domC', the result is Domain A. - If both parameters are missing then domainTypes is treated as "any domain except Reference ones" (Reference Models).	`<parameter name="domainTypes" label="domainTypes" type="string" defaultValue="Reference Models, As-is/To-be Architecture"/>`
domainIDs	List of domain IDs separated by comma.	`<parameter name="domainIDs" label="domainIDs" type="string" defaultValue="defaultDomain"/>`

Evaluating Data Sources

The data source content can be accessed directly from the web browser. The URL for the data source looks like the following:

```
http://host:port/em/web/query?dataSource=/scripts/ApplicationComponents.xml
[&limit=-1]
```

Parameter	Description	Default Value
dataSource	Required. Location of the data source script to be evaluated.	
limit	Limits the results returned. Set to -1 to return all results.	15

DQL-Based Data Sources

DQL based data sources are defined using the DQL language (DOC: create link here). See the following example:

```
<query>

  select a._uuid,a.name from applicationComponentArtifact a where
a.name=:pattern

</query>
```

The query is just wrapped between the `<query>` xml element. The query requires a `pattern` parameter.

The configuration of `<query>` element is defined as below:

Name	Type	Default Value	Description
orderBy	element	N/A	Specifies a comma separated list of field names that is used for sorting the results. For example:
maxResults	attribute	200	The report processing thread quits after producing a specified number of results. For example:

Examples:

orderBy:

```
<query>
 <orderBy>f.name asc</orderBy>
 select f.name as name, f._uuid as uuid from businessFunctionArtifact f
</query>
```

or defined in client side via *sort* parameter:

```
Ext4.create('EA.model.tools.DataSourceStore', {
    dataSource: this.config.dataSource,
     extraParams: {
            sort: [{"property":"surveyName","direction":"asc"}]
    }
});
```

or passed directly in the request URL as following:

```
https://[host:port]/[context]/web/query?dataSource=/scripts/[script_name]&sort=
[{"property":"sdmName","direction":"asc"},{"property":"name","direction":"desc"}]
```

maxResults:

```
<query maxResults="5">
  select f.name as name, f._uuid as uuid from businessFunctionArtifact f
</query>
```

or defined in client side via *pageSize* parameter:

```
Ext4.create("EA.model.tools.DQLStore", {
  query: " ",
  pageSize: 10,
  sorters: sorters
});
```

or passed directly in the request URL as following:

```
https://[host:port]/[context]/web/query?dataSource=/scripts/[script_name]&limit=10
```

Closure Definition-Based Data Sources

Closure Definition-based data sources are described in the following sections:

- "ClosureQuery Configuration Reference" below
- "Performance Considerations" on page 55
- "Displaying the Closure Query Result in a Custom UI Table" on page 56

ClosureQuery Configuration Reference

The data source takes two basic parameters:

- The traversal rules specified with an XML configuration (see the `<closure>` tag for reference).
- The `seed` parameter of `seedQuery` which specifies the artifacts the impact report is created for.

seed parameter

The `seed` parameter specifies the `uuid` of the artifact you are interested in.

seedQuery parameter

The `seedQuery` parameter specifies a dql query which is expected to return a result set of one column with a list of `uuid`s.

`<closure>` element

The wrapping element of the configuration. It defines the following:

Name	Type	Default Value	Description
maxDepth	attribute	5	Maximum distance of the result from the seed artifact specified in the number of traversed relationships.
maxResults	attribute	200	The report processing thread quits after producing a specified number of results.
maxProcessingTime	attribute	60000	The report processing thread quits its operation after the specified time in miliseconds.
nice	attribute	0	The report processing thread sleeps every 100 processed results for the given number of miliseconds. Expected to be used for longer running reports that may jam the server for other users.
debug	attribute	true	When true, detailed tracing information is written into the log file.
orderBy	element	N/A	Specifies a comma separated list of field names that is used for a sort of the results. For example: "severity DESC ,investmentRequired".
resultArtifacts	element	required	Artifacts that form the result.
traversableArtifacts	element	N/A	Artifacts that can be walked through when creating the report. Only one of traversableArtifacts,nontraversableArtifacts can be specified.
nonTraversableArtifacts	element	N/A	Artifacts that cannot be walked through when creating the report. Only one of `traversableArtifacts ,nontraversableArtifacts` can be specified.
traversableRelations	element	N/A	Relationships that can be walked through when creating the report. Only one of `traversableRelations ,relationStopList` can be specified.
relationStopList	element	N/A	Relationships that cannot be walked through when creating the report. Only one of `traversableRelations ,relationStopList` can be specified.
defaultSeedQuery	element	N/A	A dql query returning a set of `uuid`s that

Name	Type	Default Value	Description
			should serve as seeds for the query; it is overriden by the `seed` parameter.
seedsAsResults	attribute	false	Indicates an artifact as provided by `defaultSeedQuery` that will be included as a result artifact. It is valid if the `resultArtifacts` defines that kind of artifact.
parameters	element	N/A	Declares the required parameters to this data source that can be used within nested DQL statements.
nodesTraversedOnlyOnce	attribute	false	When true, a single artifact is the result at the most once. When false, a single artifact can be the result multiple number of times, if each occurrence has a different parent in the result tree.

- You can combine the `traversableArtifact` section with `traversableRelations` or `relationStopList`. If you do so conditions of both settings will be applied. In the same way you can combine `nonTraversableArtifacts`.
- Result artifacts are not traversable by default; they are added to results when reached according to specified rules. If you want to traverse relationships leading from these artifacts you have to add them to the list of traversable artifacts.

`<artifact>`

The list of artifact types that form the results. Each artifact result type may define a set of fields that will form the result. (name,description,domainId are the default fields added automatically).

```
    <artifact sdmName="businessServiceArtifact" filter="from
businessServiceArtifact a where a.consumable='1' and a._uuid=:uuid">

        <field name="implementationCount" query="select count(i._uuid) from
businessServiceArtifact b join implementationArtifact i using service where b._
uuid=:uuid"/>

    </artifact>
```

- Using the optional `filter` attribute you can filter matching artifact instances. The 'artifact' tag can be nested within `traversableArtifacts`, `nonTraversableArtifacts` or `resultArtifacts`. The artifact instance is matched when the query returns at least 1 result. You need to utilize the 'uuid' parameter in the query which holds the artifact UUID which is subject to the matching.
- You can use abstract artifacts in place of `sdmName`. In that case the rule will apply to all artifact which extend the specified artifact in addition.

- You can use the optional `reachedUsing`attribute which can filter traversed artifacts based on the relationship these have been reached. There are the following options of the value of the attribute:

incoming	The traversed artifact will be treated by the engine only if it was reached over an incoming relationship. The artifact will be treated as non existing otherwise.
outgoing	The traversed artifact will be treated by the engine only if it was reached over an outgoing relationship. The artifact will be treated as non existing otherwise.
comma separated list of relationships	The traversed artifact will be treated by the engine only if it was reached over relationship which sdm name is present within one of the values of the list defined by this attribute. The artifact will be treated as non existing otherwise.

In the following example the report is launched from the endpoint artifact (which is linked to a webServiceArtifact). It will traverse through the webServiceArtifact using the endpointOf relationship (which is an incoming relationaship inside SDM). If you would change the value of reachedUsing to 'incoming', the traversal through webServiceArtifact would happen as well. If you would change it to 'outgoing' you would not get any results.

```
<closure maxDepth="20" maxResults="1000" maxProcessingTime="60000" debug="true">

    <resultArtifacts>

        <artifact sdmName="businessServiceArtifact"/>

        <artifact sdmName="endpointArtifact"/>

    </resultArtifacts>

    <traversableArtifacts>

        <artifact reachedUsing="endpointOf" sdmName="implementationArtifact"/>

    </traversableArtifacts>

</closure>
```

\<field\>

The `field` element specifies an extra field in the result row and can be used as child of the `artifact` tag within `resultArtifacts`. There are two ways to specify the field:

- via dql query

```
<field name="implementationCount" query="select count(i._uuid) from
applicationServiceArtifact b join applicationInterfaceArtifact i using uses where
b._uuid=:uuid"/>
```

- via property sdm name specification

```
<field name="consumable" property="consumable"/>
```

The second variant has much better performance and should be used where possible. Note that to make the field actually visible you have to add an extra column to the table definition and link the field to it. Check the examples with the environment property.

There are several predefined fields (artifact fields) that you do not need to explicitly define :

_domainId
_domainName
_owner
name
description
lastApprovedStage

The `field` tag accepts the following attributes:

Name	element/ attribute	Type	Default Value	Description
query	element or attribute	string	N/A	`dql` query that is given the `uuid` parameter. The query may return a single value or even a list of multiple rows.
closure	element	XML	N/A	Nested closure definition that will be executed with the current result artifact as the seed artifact.
relationAttribute	attribute	string	N/A	Returns specified attribute value of the relationship that led to discovery of the current result artifact.
multipleResults	attribute	boolean	false	Indicates that the query returns multiple rows. The field value than will be a list of objects where properties will correspond to query column values.
limitResult	attribute	integer	20	Maximum number of results to include in the resulting JSON, if query parameter is set.
description	element	string	N/A	Complete description of the meaning of the field, so that this text is used within the UI.
property	attribute	string	N/A	Returns the value of a property of the current result artifact.

For more information, see the DQL documentation or the SDM model documentation.

<relation>

Specifies a relation and can be used with the `relationStopList` tag and `traversableRelations` tag.

```
<relation sdmName="composedOf"/>
```

The `relationship` tag has the following attributes:

Attribute	Description
sdmName	SDM name of the relationship (to DOC: please create a crosslink here)
sourceArtifact	match the relationship only if the relationship is a property of given source artifact(s) - comma separated
targetArtifact	match the relationship only if the relationship is referencing given target artifact(s) - comma separated

<parameters>

The data source may require parameters to evaluate. These are defined within the parameters section of the data source definition.

```
<parameters>

    <parameter name="plateau" label="Plateau" type="uuid"
artifactLocalName="plateauArtifact">Required to display
 ....</parameter>

    <parameter name="minimumCost" label="Min. Cost" type="number"/>

</parameters>
```

The `parameter` tag has the following attributes:

Attribute	Description
name	Name of the parameter as it is referenced from DQL.
label	Short name of the parameter (as used in UI dialogs).
type	If the type is 'uuid', this attribute further determines the artifact type whose UUID may be passed as the value of the parameter.

Examples

Example 1

List all reachable artifacts from the seed artifact:

```
<closure maxDepth="5" maxResults="1000" maxProcessingTime="60000" debug="true">

    <resultArtifacts>
```

```
        <artifact sdmName="artifactBase"/>

    </resultArtifacts>

</closure>
```

Example 2

Show all contacts having a contract on the seed business service:

```
<closure maxDepth="5" maxResults="100" maxProcessingTime="60000" debug="false">

    <resultArtifacts>

      <artifact sdmName="contactArtifact"/>

    </resultArtifacts>

    <traversableArtifacts>

      <artifact sdmName="contractArtifact" query="from contractArtifact a
where a._uuid=:uuid and a.contractState.val =
'uddi:systinet.com:soa:model:taxonomies:contractAgreementStates:accepted'"/>

    </traversableArtifacts>

</closure>
```

Example 3

Show all business services that the seed one is transitively referencing using the relationship
composed of:

```
<closure maxDepth="5" maxResults="100" maxProcessingTime="60000" debug="false">

    <resultArtifacts>

      <artifact sdmName="businessServiceArtifact"/>

    </resultArtifacts>

    <traversableRelations>

      <relation sdmName="composedOf"/>

    </traversableRelations>

</closure>
```

Example 4

Use of nested closure definition:

```
<closure maxDepth="2" maxResults="10000" maxProcessingTime="30000" debug="false"
seedsAsResults="true">
```

```
    <defaultSeedQuery>select a._uuid from applicationComponentArtifact
a</defaultSeedQuery>

    <resultArtifacts>

      <artifact sdmName="applicationComponentArtifact">

       <field name="services">

        <closure maxDepth="2">

    <resultArtifacts>

      <artifact sdmName="applicationServiceArtifact"/>

    </resultArtifacts>

        </closure>

       </field>

      </artifact>

    </resultArtifacts>

</closure>
```

Example 5

Accessing relation attributes:

```
<closure maxDepth="2" maxResults="10000" maxProcessingTime="30000" debug="false"
seedsAsResults="true">

    <defaultSeedQuery>select a._uuid from plateauArtifact a</defaultSeedQuery>

      <resultArtifacts>

        <artifact sdmName="applicationComponentArtifact">

         <field name="cost" relationAttribute="cost"/>

        </artifact>

      </resultArtifacts>

</closure>
```

Performance Considerations

For good performance it is required to specify the traversable artifacts / relations so that parts of the repository which will not produce any result will not be searched. The report is being built fairly quickly when the "query" attributes are not used frequently - even on a notebook running the database and the repository simultaneously it is able to produce 100 results per second. In the debug mode you can check the report generation times and trace the search for the artifact closure.

Memory requirement aren't a big deal - when the 200 result set is built it is hardly possible to detect the change in memory used within the server. The debug log looks like the following:

```
15:54:20,991 INFO  [ACCESS] webui.document.ACCESS:16100,INFO,http-0.0.0.0-8080-4(185),"admin","be247a9a-d614-4ddc-9c26-a514b1d52042",ARTIFACT : be247a9a-d614-4ddc-9c26-a514b1d52042  was accessed
15:54:25,356 INFO  [CompositeReportProcessor] Searching shortest path between businessServiceArtifact endpointArtifact
15:54:25,356 INFO  [CompositeReportProcessor]   found: implementationArtifact / endpoint
15:54:25,356 INFO  [CompositeReportProcessor]   found: businessServiceArtifact / service
15:54:25,376 INFO  [CompositeReportProcessor] max depth: 5
15:54:25,376 INFO  [CompositeReportProcessor] max results: 100
15:54:25,376 INFO  [CompositeReportProcessor] dequed be247a9a-d614-4ddc-9c26-a514b1d52042
15:54:25,382 INFO  [CompositeReportProcessor] be247a9a-d614-4ddc-9c26-a514b1d52042 has 8 relationships
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation r_providerOwner pointing to contractRequestArtifact / 4a1a43bb-ceda-41cd-8ccc-55d7123d4f9c
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation r_consistsOf pointing to hpsoaApplicationArtifact / 4c9ef061-51c3-47e2-90bb-43796125add6
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation r_providerOwner pointing to contractArtifact / 57bff006-c78f-41dd-998b-320176fd9f0b
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation r_dependsOn pointing to businessServiceArtifact / 5a86ff15-0e98-4abb-afb2-47b914d7699f
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation documentation pointing to documentationArtifact / 6943018a-0706-4bf3-adcb-0f135da8e6bb
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation providedBy pointing to organizationUnitArtifact / 6c907018-7d55-42e1-9f4a-729ef8b94456
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation service pointing to webServiceArtifact / 86e4b654-e34e-41bb-b976-032ed0de3bc9
15:54:25,382 INFO  [CompositeReportProcessor] Enqued 86e4b654-e34e-41bb-b976-032ed0de3bc9 / webServiceArtifact
15:54:25,382 INFO  [CompositeReportProcessor] Processing relation r_providerOwner pointing to contractRequestArtifact / ecdd86f5-4b63-4359-b853-14ec9efe7536
15:54:25,382 INFO  [CompositeReportProcessor] dequed 86e4b654-e34e-41bb-b976-032ed0de3bc9
15:54:25,384 INFO  [CompositeReportProcessor] 86e4b654-e34e-41bb-b976-032ed0de3bc9 has 36 relationships
15:54:25,384 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to endpointArtifact / 0df8e0fa-7568-4364-99b1-99d921d3b4cc
15:54:25,384 INFO  [CompositeReportProcessor] prepared query:select _domainId,_domainName,name,description,environment.val from endpointArtifact where _uuid=:uuid
15:54:25,427 INFO  [CompositeReportProcessor] Adding to results endpointArtifact / 0df8e0fa-7568-4364-99b1-99d921d3b4cc / AcmeATMTransationIfaceHttpGet_Endpoint
15:54:25,427 INFO  [CompositeReportProcessor] Enqued 0df8e0fa-7568-4364-99b1-99d921d3b4cc / endpointArtifact
15:54:25,427 INFO  [CompositeReportProcessor] Processing relation documentation pointing to documentationArtifact / 14e8ffd1-42ce-4dec-a145-22aad4fa4ed0
15:54:25,427 INFO  [CompositeReportProcessor] Processing relation r_operation pointing to hpsoaOperationArtifact / 21f88b8a-e2f3-4a65-97d2-3e13331c002b
15:54:25,427 INFO  [CompositeReportProcessor] Processing relation artifactSlo pointing to sloArtifact / 248bba25-ff46-4c7a-8780-4bad2f13a249
15:54:25,428 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to endpointArtifact / 2555c068-8dc8-42f7-8c8d-24e60c0731e2
15:54:25,485 INFO  [CompositeReportProcessor] Adding to results endpointArtifact / 2555c068-8dc8-42f7-8c8d-24e60c0731e2 / AcmeATMTransationIfaceHttpPost_Endpoint
15:54:25,485 INFO  [CompositeReportProcessor] Enqued 2555c068-8dc8-42f7-8c8d-24e60c0731e2 / endpointArtifact
15:54:25,485 INFO  [CompositeReportProcessor] Processing relation r_operation pointing to hpsoaOperationArtifact / 3371c3b8-8418-4a08-8282-8f66fb546b2f
15:54:25,485 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to endpointArtifact / 3bbeed51-8ebf-4022-862a-0c591de4ae75
15:54:25,499 INFO  [CompositeReportProcessor] Adding to results endpointArtifact / 3bbeed51-8ebf-4022-862a-0c591de4ae75 / AcmeATMTransationIfaceHttpGet_Endpoint
15:54:25,499 INFO  [CompositeReportProcessor] Enqued 3bbeed51-8ebf-4022-862a-0c591de4ae75 / endpointArtifact
15:54:25,499 INFO  [CompositeReportProcessor] Processing relation r_operation pointing to hpsoaOperationArtifact / 3cf87205-1f45-49ec-b911-4c685627a57e
15:54:25,499 INFO  [CompositeReportProcessor] Processing relation definition pointing to wsdlArtifact / 462ba7c3-0e15-47d9-8f19-a8e9f332ef00
15:54:25,499 INFO  [CompositeReportProcessor] Processing relation provider pointing to contractRequestArtifact / 4a1a43bb-ceda-41cd-8ccc-55d7123d4f9c
15:54:25,499 INFO  [CompositeReportProcessor] Processing relation r_provider pointing to contractRequestArtifact / 4a1a43bb-ceda-41cd-8ccc-55d7123d4f9c
15:54:25,499 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to endpointArtifact / 4b69731c-17f2-4bd5-8641-72112edf78e3
15:54:25,508 INFO  [CompositeReportProcessor] Adding to results endpointArtifact / 4b69731c-17f2-4bd5-8641-72112edf78e3 / AcmeATMTransationIfaceHttpGet_Endpoint
15:54:25,508 INFO  [CompositeReportProcessor] Enqued 4b69731c-17f2-4bd5-8641-72112edf78e3 / endpointArtifact
```
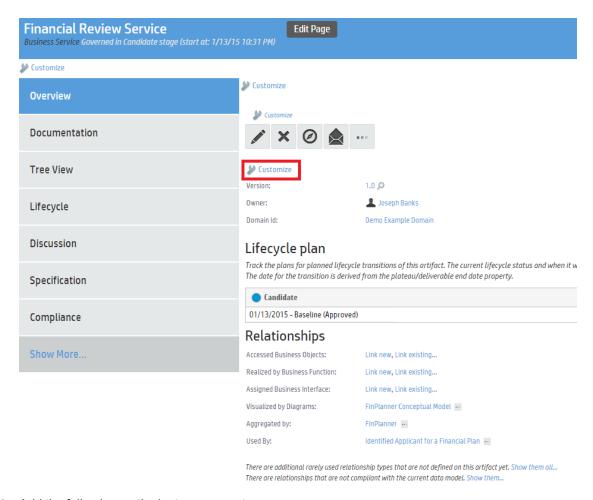
```
15:54:25,597 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to webServiceArtifact / 86e4b654-e34e-41bb-b976-032ed0de3bc9
15:54:25,597 INFO  [CompositeReportProcessor] dequed e9835901-a538-4eed-90c7-58cc20843722
15:54:25,598 INFO  [CompositeReportProcessor] e9835901-a538-4eed-90c7-58cc20843722 has 1 relationships
15:54:25,598 INFO  [CompositeReportProcessor] Processing relation endpoint pointing to webServiceArtifact / 86e4b654-e34e-41bb-b976-032ed0de3bc9
15:54:29,101 INFO  [CompositeReportProcessor] Generated 12 results in 242msec and JVM used memory difference before report execurion and now is -44507KB memory
```

Displaying the Closure Query Result in a Custom UI Table

This section describes what needs to be done to include a simple impact/dependency report on the main business service artifact page.
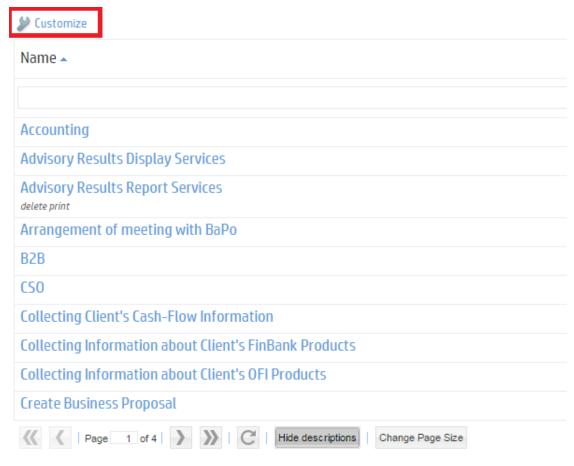
To add a table on the overview tab:

1. Add the table component into the business service artifact overview tab.

2. Switch to customization mode and navigate to a business service.

3. Click the **Customize** link under context actions:

4. Add the following as the last component:

```
<component componentName="/core/table" id="table">
<parameter
name="customizationId">provider.viewArtifact.businessServiceArtifact.impact.tab
le</parameter>
</component>
```

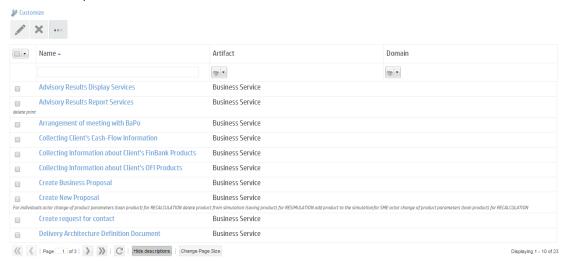5. Thereafter, click the **Customize** link above the newly added table:

6. Replace the definition with the following to show Business Services that have relationship (direct or indirect) with current artifact:

```
<?xml version="1.0" encoding="UTF-8"?>
<customization xmlns="http://soa.systinet.hp.com/2009/02/ui/customization"
xmlns:cust="http://soa.systinet.hp.com/2009/02/ui/customization"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="cust table.xsd">
    <datasource>
      <type>dataSource.composite.report</type>
      <parameter name="seed">${artifact._uuid}</parameter>
      <parameter name="configuration"><![CDATA[
          <closure maxDepth="20" maxResults="1000"
maxProcessingTime="60000" debug="true" nodesTraversedOnlyOnce="true">
```

```
                    <resultArtifacts>
                       <artifact sdmName="businessServiceArtifact"/>
                    </resultArtifacts>
            </closure>
]]>
            </parameter>
        </datasource>
        <table selectionModel="multiple">
          <rowId queryColumn="id"/>
          <column id="name" label="Name">
              <content queryColumn="name"/>
          </column>
          <column id="type" label="Artifact">
              <content queryColumn="_sdmName"/>
          </column>
          <column id="_domainName" label="Domain">
              <content queryColumn="_domainName"/>
          </column>
          <rowPreview id="description">
              <content queryColumn="description"/>
          </rowPreview>
        </table>
</customization>
```

7. The table is updated as follows:

Chapter 5: Scripting

Scripting allows you to extend the current customization framework so that custom UI components can be included in the catalog pages. These components might be used to integrate platform with other applications; it should be possible to access datasources of other applications and display it's data correlated with repository content. Other usecase might be that the information to be displayed is obtained via an web service call.

It allows small manipulations with catalog UI via javascript and the browser DOM. For example add an explanatory comment above a property display component or hide some unwanted components/buttons etc. which can't be removed via the main customization framework.

It allows execution of custom code during artifact repository operations; allow to build custom repository event handlers to perform validation;prefill artifact property values

It allows execution of custom code during lifecycle promotions. This is intended to be used to prefill some artifact properties, change artifact access rights when artifact changes lifecycle stage.

Dashboard Customization

Ext JS 4.2.1

The dashboard is based on Ext JS library - each portlet is an Ext JS component.

Check the following links for information about Ext JS:

Ext JS widgets	http://www.sencha.com/products/extjs/examples/
API documentation	http://docs.sencha.com/extjs/4.2.2/
Ext JS home	http://www.sencha.com/products/extjs/

HP Enterprise Maps provide extension classes on top of Ext JS so that you can access the data within EM easily. The full documentation of these classes can be found at EM_HOME/doc/javascript-api.

Ext JS 3

HP Enterprise Maps use Ext4 namespace prefix for referencing Ext 4 classes. The 'Ext' namespace is reserved due to backward compatibility reason for Ext version 3 classes. Do not use the Ext 3 classes because such support may be removed from the product without any further notice.

Creating Custom Portlets

To create a custom portlet:

1. Create a new portlet script by opening Administration/Customization/Manage scripts and selecting it to be of script portlet type.

2. The script will have to contain a Ext JS class that will be inherited from 'EA.portal.Portlet'.

3. The name of the class must start with "EA.scripts" namespace prefix. All classes in this namespace are loaded from the collection of scripts.

4. The rest of the name after "EA.scripts" is converted to the location attribute of the script artifact and a '.js' suffix is added. For example, if the class name is 'EA.scripts.demo.LayerStatisticsChart' the script name should be '/demo/LayerStatisticsChart.js'

Check the following example. It shows the number of artifacts in individual archimate layers.

```
Ext4.define('EA.scripts.demo.LayerStatisticsChart', {

    extend: 'EA.portal.Portlet',

    requires: [
        'Ext4.data.JsonStore',
        'Ext4.chart.theme.Base',
        'Ext4.chart.series.Series',
        'Ext4.chart.series.Line',
        'Ext4.chart.axis.Numeric'
    ],

initComponent: function() {

        var dqlStore=Ext4.create('EA.model.tools.DQLStore', {
            query: "<query>(select 'Business Layer' as name, count(a._uuid) as
artifactCount,'business' as layer
from c_businessArchitectureElement a) union "+
                    "(select 'Application Layer' as name, count(a._uuid) as
artifactCount,'application' as layer
from c_applicationArchitectureElement a) union"+
                    "(select 'Technology Layer' as name, count(a._uuid) as
artifactCount,'technology' as layer
from c_technologyArchitectureElement a) union"+
                    "(select 'Motivation' as name, count(a._uuid) as
artifactCount,'motivational' as layer
from c_motivationalArchitectureElement a) union"+
                    "(select 'Implementation and Migration' as name, count(a._
uuid) as artifactCount,'implementation'
as layer from c_implementationAndMigrationElement a) order by artifactCount "+
                    "</query>",
            fields: [
                {
                    name: 'name',
                    type: 'string'
                },
                { name: 'artifactCount' },
                { name: 'layer' }
            ]
    });

    dqlStore.load();

    Ext4.apply(this, {
        layout: 'fit',
        height: 300,
```

```
        items: {
              xtype: 'chart',
              animate: true,
              style: 'background:#fff',
              shadow: false,
              store: dqlStore,
              axes: [{
                  type: 'Numeric',
                  position: 'bottom',
                  fields: ['artifactCount'],
                  label: {
                      font:'HPSimplified',
                      renderer: Ext.util.Format.numberRenderer('0')
                  },
                  title: 'Artifact count',
                  minimum: 0
               }, {
                  type: 'Category',
                  label: {
                        font:'HPSimplified'
                  },
                  position: 'left',
                  fields: ['name']
              }],
              series: [{
                  type: 'bar',
                  axis: 'bottom',
                  xField: 'name',
                  yField: ['artifactCount'],
                  renderer: Ext4.create
 ('EA.model.tools.LayerToColorConvertor').getChartColorRenderer(function(record)
 { return record.get('layer');})
              }]
          }
      });
      this.callParent(arguments);
    }
});
```

Overriding Behaviour of Existing Portlets

The extensibility described above doesn't apply to whole portlets. You may extend the existing structure maps and Heat Map with new functionality. See the following structure map portlet definition:

```
{
    id: 'capabilityToProjectMapping',
    dataSource: '/scripts/BusinessFunctions.xml',
    visualizations: [{
```

```
        label: 'Background Color',
        items: [{
            type: 'EA.portlets.visualization.NumberBasedColorVisualization',
            field: 'plannedCost',
            name: 'Project planned costs'
        }]
    }]
}
```

Notice the reference to `EA.portlets.visualization.NumberBasedColorVisualization`. This is an HP EM built-in class - you can replace this one with your own. For example you may create a class `EA.scripts.visualization.MoneyBasedColorVisualization` that will change the behaviour of the built-in visualization class - it will render currency symbols:

```
Ext4.define('EA.scripts.visualization.MoneyBasedColorVisualization', {
    extend: 'EA.portlets.visualization.NumberBasedColorVisualization',

    getDescription : function (lowerMargin, higherMargin) {
        if (lowerMargin == null) return 'Cost N/A';
        var description = '$' + layoutManager.addCommas(lowerMargin.toFixed(0))
+ ' - $' + layoutManager.addCommas(higherMargin.toFixed(0));
        return description;
    },
    getTextValue: function(node) {
        var value = node.data[this.getField()];
        return (value == null || value == '') ? 'N/A' : (value == 0 ? value :
'$' + value);
    }

});
```

All you need to do to use the new visualization you have change the type in the portlet declaration:

```
{
    id: 'capabilityToProjectMapping',
    dataSource: '/scripts/BusinessFunctions.xml',
    visualizations: [{
        label: 'Background Color',
        items: [{
            type: 'EA.scripts.visualization.MoneyBasedColorVisualization',
            field: 'plannedCost',
            name: 'Project planned costs'
        }]
    }]
}
```

To understand the API and features you may utilize when you create new script extensions, please check the documentation in `EM_HOME/doc/javascript-api`.

General Catalog Customization

It is possible to enter custom html fragments inside the platform UI customization file. This way you can add extra explanation labels above property declarations, context actions etc. You can also place javascript fragments there and extending the platform UI with you own dialogs. There are two tags used for this: `<html>` tag and `<server>` tag

If you want to try these examples yourself, switch the repository into the UI customization mode, and click on the customize link just under the **Catalog** tab. Paste the here mentioned code snippets as the first child of the first html element you will find

```
?xml version="1.0" encoding="UTF-8"?>
<customization xmlns="http://soa.systinet.hp.com/2009/02/ui/customization"
xmlns:cust="http://soa.systinet.hp.com/2009/02/ui/customization"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="cust
columns.xsd">
    <columns>
        <column id="leftColumn">

            <!-- PLACE YOUR CUSTOMIZATION CODE HERE -->
            <server id="my_server_code">
              <script>
                function calculateArtifactCount() {
                    var result=queryService.query("&lt;query>select count(*) as
cnt from artifactBase a where a.name like
                    :pattern&lt;/query>", { pattern: '%' });
                    return result.records[0].cnt;
                }
              </script>
            </server>
            <html id="my_extension">
             ....
```

`<html>` Tag

As mentioned earlier the first tag you can use within customization files is the <html> tag.

```
<column ...>
  <html id="my_extension">
    <include>
      <script>
        alert('Hello');
      </script>
      <span style="font-size:30px">Here is some text !</span>
    </include>
  </html>
```

```
</column ...>
```

This is a very basic example of an extension HTML - it will place the content of the element into the page. When the page with this customization is being rendered, first the the alert() javascript function is executed:

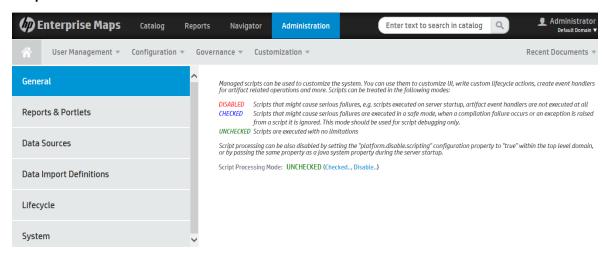And then the "Here is some text !" is placed at the beginning of the page:

The html element can be placed inside `<column>`,`<group>` and `<contextActions>` elements of the customization files.

If the customization is a bigger one you can place the extension HTML file into an include file like this:

Using Include File/Properties

```
<html id="my_extension">
  <import location="/scripts/common2.js"/>
  <include>
     ..
  </include>
</html>
```

You can include the fragment from multiple places and edit it by selecting **Administration > Manage scripts**.
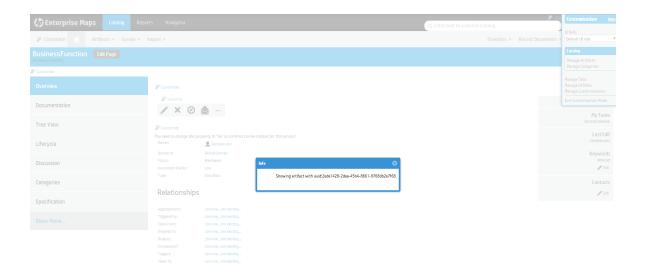
You can place the script code into a platform system/configuration property as well. The code using this property will be something like the following:

```
<html id="my_extension">
  <import property="platform.scripts.my-customization-script"/>
  <include>
      ..
  </include>
</html>
```

Passing Parameters to the Code Inside the html Tag

On pages showing/editing an artifact you can pass parameters into the code within the html tag.

```
<html id="my_extension">
  <parameter name="artifactUUID">${artifact._uuid}</parameter>
  <include>
    <script>
       Ext4.onReady(function () {
           Ext4.Msg.show({
               title: 'Info',
               msg: 'Showing artifact with uuid:'+my_extension.artifactUUID,
               buttons: Ext.MessageBox.OK,
               icon: Ext.MessageBox.INFO
           });
       });
    </script>
    <div style="font-style:italic;margin-top:5px">
     You need to change this property to 'Yes' so contract can be created for
this service !
     </div>
  </include>
</html>
```

When executed, you will get the following result (if the customization is placed on the view implementation page, not the catalog home as other examples):
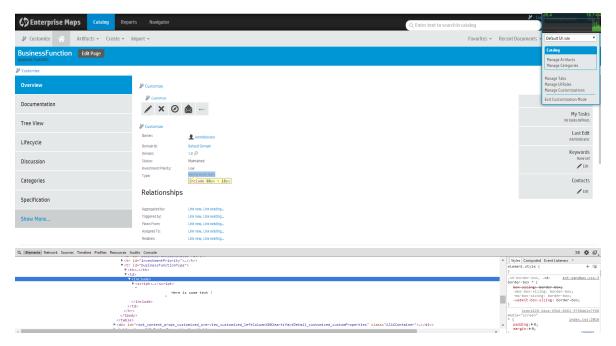
In this case all the parameters must be serializable to String - you cannot pass the whole artifact instance. This example also demonstrates a piece of javascript that gets executed after the whole web page has been downloaded from the server (use of the Ext4.onReady function).

Manipulating Web Page DOM

You can also manipulate the page DOM from javascript. For example to following fragment will make the 'Change WSDL link disappear':

```
<html id="my_extension">
  <include>
    <script>
      Ext.onReady(function () {
          var e=Ext4.get('root_content_props_customized_overview_customized_
rightColumnX002econtextButtons_customized_changeWsdllink');
          e.setVisibilityMode(Ext.Element.DISPLAY);
          e.setVisible(false);
      });
    </script>
    Here is some text !
  </include>
</html>
```

You can use the Firebug plug-in (or other similar tools available for major HTML browsers) to search for id of the item you want modify. The link below shows the id of the "Change WSDL" link located using the "Inspect" function of Firebug. You will see the following result: (if the customization is placed on the view implementation page, not the catalog home as other examples).

Here is another example which hides "end governance" and "set lifecycle process" links in the artifact life cycle tab for non-admin users. Include this script into one of the groups of the left side menu - customizations in this place will execute for all catalog tab pages:

```
<server id="admin_detection">
  <script>
    function isAdmin()
    {
        return Packages.com.hp.em.security.auth.SecurityContext.current
().isInRole("Administrator");
    }
  </script>
</server>
  <html id="my_extension">
    <parameter name="user">admin</parameter>
    <include>
        <script>
          Ext4.onReady(function () {
              var endGovernance=Ext.get('root_content_props_customized_
lifecycle_columns_customized_rightColumnX002ebuttonPanel_
lifecycleTabX002eendGovernanceungovernLink');
              if (endGovernance!=null)
              {
                endGovernance.setVisibilityMode(Ext.Element.DISPLAY);
                endGovernance.setVisible(false);
                var setProcess=Ext.get('SetProcess_handler');
                setProcess.setVisibilityMode(Ext.Element.DISPLAY);
                setProcess.setVisible(false);
                isAdmin( function(isAdminValue)
```

```
                        {
                                setProcess.setVisible(isAdminValue=='true');
                                endGovernance.setVisible(isAdminValue=='true');
                        });
                }
        });
    </script>
  </include>
</html>
```

Here is one more example which shows you a mechanism how to build new layout of the edit/view artifact pages based on the existing property widgets:

```
<customization xmlns="http://soa.systinet.hp.com/2009/02/ui/customization"
   xmlns:cust="http://soa.systinet.hp.com/2009/02/ui/customization"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="cust artifactDetail.xsd">
   <content>
       <group id="properties" label="">
           <property id="name" name="name"/>
           <property id="description" name="description"/>
           <property id="version" name="version"/>
           <html id="example">
              <include>
                <table>
                    <tr>
                       <td>XXX</td><td><table><tr id="new_version"/></table></td>
                    </tr>
                </table>
                <script>
                        var version=Ext4.get("version");
                        var new_version=Ext4.get("new_version");
                        new_version.dom.innerHTML=version.dom.innerHTML;
                        version.dom.innerHTML='';
                </script>
              </include>
           </html>
           <property id="r_serviceType" name="r_serviceType"/>
           <property id="criticality" name="criticality"/>
           <property id="readyForConsumption" name="readyForConsumption"/>
```

<server> Tag
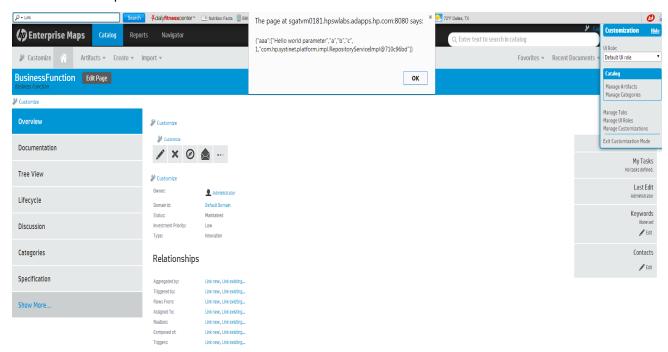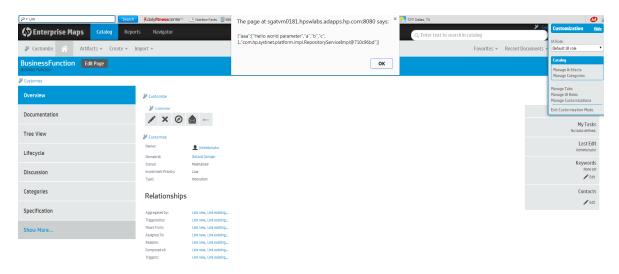
The server tag can be used to define custom business logic executed on the server side. Check the following example:

```
...
   <server id="my_server_code">
      <script>
         function test1(param) {
         return { aaa: [param, 'a','b','c', 1, repositoryService] };
         }
      </script>
   </server>

   <html id="my_extension">
      <include>
         <script>
           function responseListener(result)
           {
              alert(Ext4.encode(result));
           }
           test1('Hello world parameter',responseListener);
         </script>
      </include>
   </html>
...
```

When such customization is evaluated (request for rendering is placed) the content of the server tag is compiled and stored on the server and only function stubs are put into the resulting html. When such a stub is invoked it sends it's parameters to the server (in the example above it is the 'Hello World' string) where the previously compiled function is executed. Results are then returned back to the browser.

The results of the previous customization looks like this:

You can see that the repository service has not been sent to the browser, the java toString() method has been used to serialize it and the result has been sent instead.

All this is done asynchronously and the client must provide a call back function (responseListener in our example) to the stub which is reponsible for processing function results.

Server Side Execution Environment

The javascript interpreter on the server side is implemented by the Rhino engine. You can use java based runtime Enterprise Maps platform APIs in your scripts, check the following documentation on interfacing java with Rhino: http://www.mozilla.org/rhino/ScriptingJava.html

You can use the following objects from the script:

JS Object	Java Object
UUID	com.hp.em.repository.util.PropertiesUtil
beanFactoryHelper	com.hp.em.spring.BeanFactoryHelper
repositoryService	com.hp.em.repository.RepositoryService
artifactFactory	com.hp.em.repository.sdm.ArtifactFactory
queryService	com.hp.em.sc.ui.scripting.dataService.DqlJsQueryService
repositoryPreListeners	com.hp.em.sc.ui.scripting.events.ScriptedEventListener
repositoryPostListeners	com.hp.em.sc.ui.scripting.events.ScriptedEventListener
log	org.apache.commons.logging.Log

Check the separately provided javadoc to see methods provided by these objects.

Reading System Configuration From a Server Script

It is possible to access system configuration from the server scripts. Like on the following example:

```
...
    log.info('Java version:'+environment.getConfigurationProperty
('java.version'));
    log.info('Platform url base:'+environment.getConfigurationProperty
('platform.url.base'));
...
```
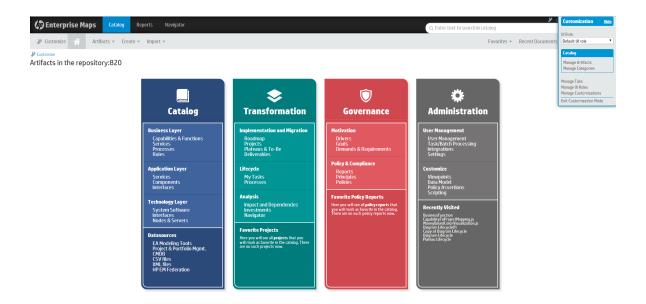
If the property cannot be found within the platform system configuration a search within the environment variables is performed.

Executing DQL in <server>

The following example shows how the execute DQL from the tag:

```
<server id="my_server_code">
  <script>
    function calculateArtifactCount() {
      var result=queryService.query("&lt;query>select count(*) as cnt from
artifactBase a where a.name like :pattern&lt;/query>", { pattern: '%' });
      return result.records[0].cnt;
    }
  </script>
</server>

<html id="my_extension">
  <include>
    <div style="font-size:24px">Artifacts in the repository:<span
id="resultContainer"></span></div>
    <script>
       Ext.onReady(function () {
       function responseListener(result)
       {
           Ext4.get('resultContainer').dom.innerHTML=result;
       }
       calculateArtifactCount(responseListener);
      });
    </script>
  </include>
</html>
```

The result of this customization is the following:

Note that with the server tag you need to escape <, & characters as these are included within an XML file. If you would place the above text into an imported file you wouldn't do that. Also note that the query function is intended to be used for smaller data load. For huge data use the queryAsString method which returns an JSON string or check the Ext JS grid example below. Client side javascript execution environment

In the example above it is also demonstrated that you can use the Ext JS libraries on the client side within the tag. This is a very prowerful feature since you can load Ext JS stores with DQL queries and use any of the Ext JS components to visualize it.

Using Platform DQL/JSON Query Service

Platform provides two endpoints that can be used to query the repository content; the first one is used to execute DQL and the other one for obtaining taxonomy data.

Ext JS Grid Filled by AsynchronouslyLoaded Data From Server

Check the example below how to use the Ext JS grid withEnterprise Maps. Here a DQL query is used to fill Ext JS data store. The query is passed to a servlet which in return sends JSON data back. The servlet supports standard paging parameters of ext js stores as well.

```
<html id="my_extension">
   <include>
     <div id="reportContainer"></div>
      <script>

       Ext4.onReady(function () {

       var testStore=Ext4.create('EA.model.tools.DQLStore', {
           query: "<query>select a._uuid as uuid, a.name as name, a.description
as description from personArtifact a</query>",
           fields: [
                    {
```

```
                        name: 'uuid',
                        type: 'string'
                    },
                    {
                        name: 'name',
                        type: 'string'
                    },
                    {
                        name: 'description',
                        type: 'string'
                    }              ]
        });
        testStore.load();

        var testGrid = new Ext4.grid.Panel({
            store: testStore,
            renderTo    : 'reportContainer',
            width: 'auto',
            height:500,
            emptyText: 'No results to display',
            autoExpandColumn: 'name',
            columns: [
                    {
                         id: 'name',
                         text: 'Name',
                         dataIndex: 'name',
                         sortable: true
                    },
                    {
                        id: 'description',
                        text: 'Description',
                        dataIndex: 'description',
                        width: 150,
                        sortable: true
                    }
                ]
        });

        });

    </script>
   </include>
 </html>
```

The result of this customization is the following:

Querying Taxonomy Data

In a very similar way you can load taxonomy data into an Ext js store:

```
var environments = new Ext.data.JsonStore({
    autoDestroy: true,
    url: SERVER_
URI+'/../../taxonomy?taxonomy=uddi:systinet.com:soa:model:taxonomies:environment
s',
    root: 'records',
    idProperty: 'key',
    fields: ['key', 'value' ]
});
environments.load();
```

From this example you can see that there are two new endpoints/servlets available on the platform server. All use ui authentication (so the results returned correspond to permissions of the currently logged user and you can use them separately - only from the platform user interface). One is used to process DQL queries and return JSON data and the other one is used to list taxonomy data.

Executing Code on Server Startup/Shutdown

Using Administration/Customization/Manage scripts you can create scripts that are executed during server startup/shutdown. In the example blow you can see how that works; the script must define the onStartup function which is executed on the server startup or the script update. You can also define onTearDown function that is executed on server shutdown or before the script is updated. The example below registers two listeners for artifact related actions. This way you can implement some custom integrity constraints and so on.

```
var onStartup=function() {
  repositoryPreListeners.add('demo-listener-pre-id',function (event) {
    log.info('pre listener: '+event.getType()+' sdmName:'+event.getSdmName()+
            ' artifact:'+event.getArtifact());
  });

  repositoryPostListeners.add('demo-listener-post-id',function (event) {
    log.info('post listener: '+event.getType()+' sdmName:'+event.getSdmName()+
            ' artifact:'+event.getStoredArtifact());
  });
}
```

```
var onTearDown=function() {
  repositoryPreListeners.remove('demo-listener-pre-id');
  repositoryPostListeners.remove('demo-listener-post-id');
}
```

Handler/on Startup Script Processing Mode

It is important that those scripts are written correctly - a single error may lead to completely inaccessible repository. Therefore repository can operate in the following modes (You can change the mode on the 'Manage Scripts' UI page):

DEBUG	Scripts are executed in a safe mode, when a compilation failure occurs or an exeception is raised from a script it is ignored.
PRODUCTION	Scripts are executed with no limitations.
DISABLED	Scripts executed on server startup and artifact event handlers are not executed at all.

Javascript-Based Repository Event Handlers

You can define custom repository event handler - events are generated for every artifact create/update/delete/purge/get/find operation. You can enhance the customization capabilities of HP EM capabilities in several ways:

1. Performing data integrity / constraints checks before artifact create/update
2. Prefill default values for certain artifact properties, even based on the data already entered into the modified artifact
3. Custom security constraints even on property level by hooking the get operation

Handler Template
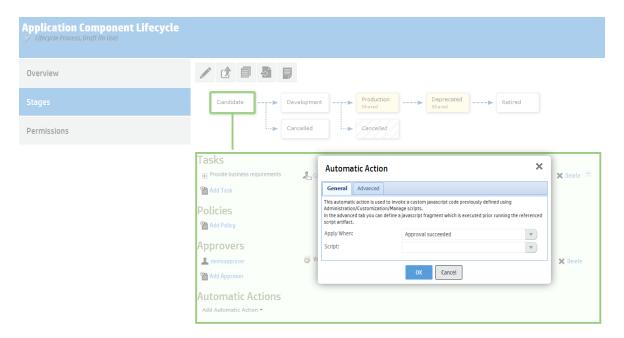
You may utilize the following template when writing a new repository handler.

```
/**
 * Startup function MUST define repository handler code as well
 * the registration of the handler using a specified handler
 * identifier.
 */
var onStartup=function() {
  // 0. specify event handler identifier (replace all occurences in this file)
  var eventHandlerId = 'TODO-SPECIFY-HANDLER-ID';
  // 1. Specify an array containing event types to trigger, posible elements are
  // GET, GET_DATA, CREATE, DELETE, UNDELETE, UPDATE, PURGE, CHANGE_OWNERSHIP
  var eventTypes = ['CREATE', 'UPDATE'];
  // 2. Specify artifact SDM name according to sdm model
  var sdmName = 'businessServiceArtifact';
```

```
  // 3. Implement handler code
  var handler = function(event){
    log.info('TODO implement handler');
  }

// A helper function that cares about about exception handling
// and event filtering
var handlerWrapper=function(event){
  try{
    // implement handler
    var type = event.getType();
    var eventMatch = false;
    for(var i=0; i<eventTypes.length; i++){
      if (eventTypes[i] == type){
        if (event.getSdmName() == sdmName){
          handler(event);
        }
        break;
      }
    }
  } catch(e){
    log.error(e);
  }
}

// registration of the handler for pre and/or post execution
// pre/post handlers have different identifier namespaces
repositoryPreListeners.add('TODO-SPECIFY-HANDLER-ID',handlerWrapper);
// repositoryPostListeners.add('TODO-SPECIFY-HANDLER-ID',handlerWrapper);
}

/**
 * Teardown function is used to unregister the handler, the same
 * identifier must be used to unregister the handler.
 */
var onTearDown=function() {
  repositoryPreListeners.remove('TODO-SPECIFY-HANDLER-ID');
  repositoryPostListeners.remove('TODO-SPECIFY-HANDLER-ID');
}
```

Lifecycle-Triggered Script Execution

Using Administration/Customization/Manage scripts you can create scripts that are executed during lifecycle approval process. There is an automatic action called 'Execute Script' (see screen shot below) that is able to execute those script (those must be of type 'Lifecycle action'). It is also possible to define an extra script in the lifecycle action which is executed in the same environment and prior to the main script. It is intended to be used to pass parameters to the main script.

The very basic script might look like as simple as this:

```
System.err.println('Governance record:'+governanceRecord);
log.info('Governance record:'+governanceRecord);
```

You can see that there is a com.hp.em.platform.lifecycle.GovernanceRecord passed via the reference named governanceRecord to the script.

Tips

Sometimes there is a repository object/API in which there is no documentation. In that case, the following function might help you to introspect the properties of such an object:

```
// declare functions that are used by the handler code
var dumpObject = function(o){
  var properties = java.beans.Introspector.getBeanInfo(o.getClass
()).getPropertyDescriptors();
  for(var i=0; i<properties.length; i++){
    var prop = properties[i];
    var value = null;
    if (prop.getReadMethod()!=null){
      try{
        value = prop.getReadMethod().invoke(o,null);
      } catch (e){
        // ignore
        log.error(e);
      }
```

```
    }
    if (value!=null){
      log.info(" "+prop.getName()+'='+value);
    }
  }
}
```

Scripted Task Execution

Overview

"Scripted task execution" is a concept that allows to easily introduce new implementations of EM tasks using embedded scripts. New task implementation can be created/modified/deleted anytime without the need of EM restart. Embedded scripts can be then scheduled (or executed ad-hoc) as any other EM tasks.

First Steps

To create a new type of task, you need to create a task script. After a task script is created, you can schedule an HP EM task that allows both scheduled and ad-hoc execution.
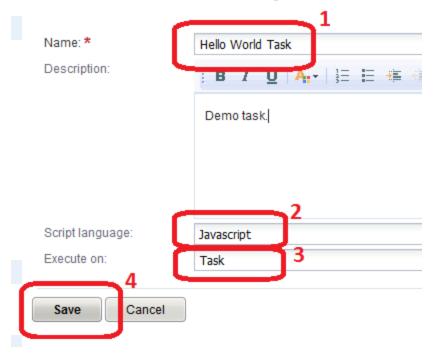
Create a Task Script

1. Login as Administrator and select the **Administration** tab **> Customization > Manage Scripts** to open the Script Management page.

2. From the **System** tab, click the **Create new script** icon to open the Create New Script editor.

3. Fill in the script properties. Add a unique name (for example, "Hello World Task")and select **Javascript** as the **Script language** and **Task** as the **Execute on** type. Click **Save**.

Embedded script

4. Click **Edit Script** to add the script content and click **Save**.
   The script content must be a JavaScript function execute(). The function should return a string value that is a short description of the execution result, as shown in this example:

```
function execute(){
    return 'Hello World!';
}
```

5. The Task script is created.

Hello World Task

📄 *Embedded script*

Description:	Demo task.
Script language:	Javascript
Execute on:	Task
Location:	/scripts/Hello World Task

```
1 function execute(){
2    return 'Hello World!';
3 }
```

Create an HP EM Task That Will Execute a Task Script
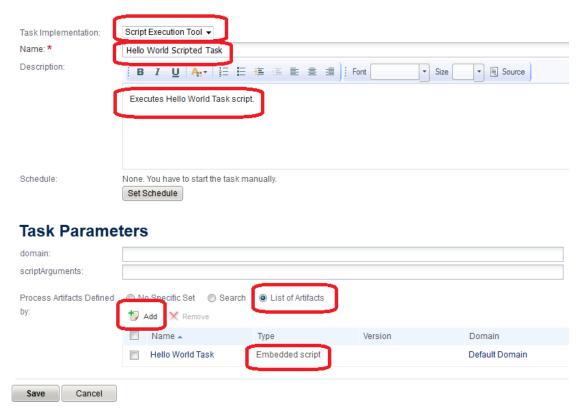
1. Login as Administrator and select the **Administration** tab **> Configuration > Tasks** to open the Tasks Management page.

2. Click the **Create new task** icon and click **Add Javascript Task**.

3. Fill in the task properties and click **Save**:
   a. **Task Implementation**: (Required) Select the **Script Execution Tool**.

   b. **Name**: (Required) Add a unique name -- for example: *Hello World Scripted Task*.

   c. **Description**: (Optional) Add a description of what the task does.

   d. **Schedule**: (Optional) Can be specified as with any other task.

   e. **Domain**: (Optional) Parameter can be specified to set a working domain (domain identifier) for task execution. **topLevelDomain** is used if no domain is specified.

f. **ScriptArguments**: (Optional) Parameter can be used to setup script arguments. It must carry a comma-separated name=value pairs. The execute function (of the embedded task script) is then called with specified arguments.

g. **Process Artifacts Defined by**: (Required) Select **List of Artifacts**, **Add**, and **Embedded script**.

Add Custom Task

Task Implementation: Script Execution Tool ▾

Name: * Hello World Scripted Task

Description:

| **B** *I* U̲ | A. | ≣ ≣ | ⧉ ⧉ ⧉ | ≣ ≣ ≣ ≣ | Font [▾] Size [▾] | 🗎 Source |

Executes Hello World Task script.

Schedule: None. You have to start the task manually.

[Set Schedule]

Task Parameters

domain: [   ]

scriptArguments: [   ]

Process Artifacts Defined by: ○ No Specific Set   ○ Search   ● List of Artifacts

📄 Add   ✖ Remove

☐	Name ▲	Type	Version	Domain
☐	Hello World Task	Embedded script		Default Domain

[Save] [Cancel]

4. The Task is created. You can run the task immediately to test it. Click **Run** and then confirm that you want to run it in the confirmation dialog.

Hello World Scripted Task

☆☆☆☆☆ no votes

📝 *Custom Task*

Executes Hello World Task script.

▶ Run

✏ Edit Schedule

Task Implementation: Script Execution Tool

🖉 Edit

Task Parameters

domain:

5. The Task's execution history shows that the task was executed.

Execution History

Starte...	Result	▼	Run Time	Details
5:26 P...	✔ Finished		00:00:02	'Hello World!'

Modify the Script Anytime

You can modify the task script any time without restarting the server. This is also a way to debug/tune your task. Try to change the task script to return "Hello Europe!" and run the task again.

More Examples

Example 1: Script that executes HP EM publishing.

```
function execute(){
  /* ************************************************** */
  /* Setup repository location of published resource(s)    */
  /* ************************************************** */
  var repoLocation = '/test';

  /* *************************** */
  /* Create temporary directory      */
  /* *************************** */
  // setup a temporary directory of your choice, it is required
  var tmpDir = new Packages.java.io.File(System.getProperty
("java.io.tmpdir")+'/s4tmp_'+System.currentTimeMillis());
  tmpDir.mkdirs();
```

```
/* **************************** */
/* Create publisher input       */
/* **************************** */
// 1. publish zip file from URL
// var input =
Packages.com.hp.em.publishing.struct.PublisherInput.lazyZip
('http://blabla',repoLocation);

// 2. publish non-zip file from URL
// var input = new Packages.com.hp.em.publishing.struct.PublisherInput
('http://blabla',repoLocation);
// 3. publish zip file from filesystem (it has to be uploaded to tmpDir)
var srcFile = new Packages.java.io.File('c:\\tmp\\hello\\hello1.zip');
var tmpFile = new Packages.java.io.File(tmpDir,srcFile.getName());
Packages.org.apache.commons.io.FileUtils.copyFile(srcFile, tmpFile);
var input = Packages.com.hp.em.publishing.struct.PublisherInput.lazyZip
(tmpFile,repoLocation);
// 4. publish non-zip file from filesystem (has to be uploaded to tmpDir)
//var srcFile = new Packages.java.io.File('c:\\tmp\\hello\\1\\hello.wsdl');
//var tmpFile = new Packages.java.io.File(tmpDir,srcFile.getName());
//Packages.org.apache.commons.io.FileUtils.copyFile(srcFile, tmpFile);
// var input = new Packages.com.hp.em.publishing.struct.PublisherInput
(tmpFile,repoLocation);


/* **************************** */
/* Customize publisher input     */
/* **************************** */
// *** synchronization policy is used by synchronization task, when scheduled
***
//input.setSyncPolicy
(Packages.com.hp.em.publishing.struct.SyncPolicy.NONE);
//input.setSyncPolicy
(Packages.com.hp.em.publishing.struct.SyncPolicy.APPROVE);
input.setSyncPolicy
(Packages.com.hp.em.publishing.struct.SyncPolicy.AUTO);
// *** keep or overwrite changes during publishing ***
//input.setOverwriteChanges
(Packages.com.hp.em.publishing.Publisher.CollisionSetting.KEEP);
input.setOverwriteChanges
(Packages.com.hp.em.publishing.Publisher.CollisionSetting.OVERWRITE);
// *** disable duplicate resolution, we cannot ask the user to resolve
duplicates ***
input.setDuplicateResolution(false);
// *** setup credentials that might be required to get HTTP resources  ***
//var creds = new Packages.org.hp.em.http.CredentialsList();
//creds.addCredentials(new Packages.org.hp.em.http.BasicCredentials(/*
URI */ null, 'user', 'password');
```

```
  //input.setCredentialsList(creds);
  // *** specify whether update of artifacts should be performed with the
identity of their owner ***
  //input.setUpdateAsOwner(false);

  /* ***************************** */
  /* Create/Setup publishing options */
  /* ***************************** */
  var options = beanFactoryHelper.getBean
(Packages.com.hp.em.publishing.options.OptionsManager).getDefaultOptions();
  // *** customize bpel options ***
  var bpelOptionsFactory = beanFactoryHelper.getBean
(Packages.com.hp.em.publishing.options.BpelOptionsFactory);
  var bpelOptions = bpelOptionsFactory.fromOptionsList(options);
  //bpelOptions.setDecomposition
(Packages.com.hp.em.publishing.options.BpelOptions.DecompositionType.BUSINESS_
PROCESS);
  bpelOptions.setDecomposition
(Packages.com.hp.em.publishing.options.BpelOptions.DecompositionType.NONE);
  var newBpelOptions = bpelOptionsFactory.toOptions(bpelOptions);
  options.remove(newBpelOptions);
  // remove old options (options are equal if they are of the same type)
  options.add(newBpelOptions);
  // *** customize wsdl options ***
  var wsdlOptionsFactory = beanFactoryHelper.getBean
(Packages.com.hp.em.publishing.wsdl.WsdlOptionsFactory);
  var wsdlOptions = wsdlOptionsFactory.fromOptionsList(options);
  //wsdlOptions.setDecomposition
(Packages.com.hp.em.publishing.wsdl.WsdlOptions.DecompositionType.ALL);
  //wsdlOptions.setDecomposition

(Packages.com.hp.em.publishing.wsdl.WsdlOptions.DecompositionType.IMPLEMENTATION
S);
  wsdlOptions.setDecomposition
(Packages.com.hp.em.publishing.wsdl.WsdlOptions.DecompositionType.NONE);
  wsdlOptions.setServiceType(new Category
('uddi:hp.com:soa:model:service:type', 'Business service', 'businessService'));
  //wsdlOptions.setServiceType(new Category
('uddi:hp.com:soa:model:service:type', 'Application service',
'applicationService'));
  //wsdlOptions.setServiceType(new Category
('uddi:hp.com:soa:model:service:type', 'Infrastructure service',
'infrastructureService'));
  var newWsdlOptions = wsdlOptionsFactory.toOptions(wsdlOptions);
  options.remove(newWsdlOptions); // remove old options (options are equal if
they are of the same type)
  options.add(newWsdlOptions);
```

```
   /* **************************** */
   /* Start publishing asynchronously */
   /* **************************** */
   // setup temporary directory, will be deleted by publisher
   input.setRootDir(tmpDir);
   // run publishing
   var asyncPublisher = beanFactoryHelper.getBean
(Packages.com.hp.em.publishing.async.AsyncPublishing);
   var report = asyncPublisher.publish(input, options);
   return "Publishing started asynchronously, see
/em/platform/rest/artifact/reportArtifact"+report.substring(report.lastIndexOf
('/'));
}
```

Example 2: Parametrized script that starts the (OS) process.

```
function execute(command, arg1, arg2, arg3, arg4){
  if (command == null){
    return "No command specified!";
  }
  list = new Packages.java.util.ArrayList(5);
  if (arg1!=null) list.add(arg1);
  if (arg2!=null) list.add(arg2);
  if (arg3!=null) list.add(arg3);
  if (arg4!=null) list.add(arg4);
  process =  new Packages.java.lang.ProcessBuilder(list).start();
  return "Process was started: "+list.toString();
  // return "Process finished with exit code: "+process.waitFor();
}
```

Example 3: Script that recalculates Top Reports.

```
function execute(){

  function getArtifact(reportDefinitionName) {
    var artifacts = repositoryService.findArtifacts(
      new Packages.com.hp.em.repository.command.FindCommand(
        'hpsoaBirtReportArtifact',
        new Packages.com.hp.em.repository.criteria.filtering.PropertyFilter
('r_reportDefinitionName',new Packages.java.lang.String(reportDefinitionName)),
        Packages.com.hp.em.repository.structures.ArtifactPartSelector.ALL_
PROPERTIES));
    return artifacts.get(0);
  }

  function recalculate(hpsoaBirtReport) {
    var definitionId =
```

```
Packages.com.hp.em.report.ui.impl.birt.BirtReportHelper.NONE_DEFINITION_ID;
    if(hpsoaBirtReport.getR_reportDefinitionName()!=null) {
      definitionId = hpsoaBirtReport.getR_reportDefinitionName();
    }
    var reportDocBean =

Packages.com.hp.em.report.ui.impl.birt.BirtReportHelper.createReportDocumentFrom
Xml
(hpsoaBirtReport.getR_reportRequestContent());
    Packages.com.hp.em.report.ui.impl.birt.BirtReportHelper.executeReport
(Packages.com.hp.em.report.ui.impl.birt.BirtReportHelper.getReportingUrl
(),definitionId,reportDocBean);
    }

    var hpsoaBirtReport = getArtifact('top_reports');
    recalculate(hpsoaBirtReport);
    return "recalculated: "+hpsoaBirtReport.getName();
}
```

Shortcut Definition Scripting

Shortcut definition is an XML script which defines how a shortcut instance is created. Shortcut is a term for its short form reference.

Shortcut definition structure:

```
<shortcutsDefinition xmlns="urn:com.hp.eam.shortcuts.definition:1.0">
  <shortcut id="infraServiceToSoftware" canBeEntered=”true”
relationshipType="associatedWith"
  autoApprove="true" label="Associated Software" inverseLabel="Associated
Deployments"
  delayedCalculation="true">
    <source localName="infrastructureServiceArtifact"
relationshipType="realizedBy" />
    <intermediate localName="serverArtifact" relationshipType="assignedTo"
generateName="${source.name}
    Server"/>
    <target localName="systemSoftwareArtifact"/>
  </shortcut>
</shortcutsDefinition>
```

Shortcut definition attributes

Name	Applicable to	Description
id	<shortcut>	The id is unique. You can not define multiple shortcuts with the same id.

relationshipType	<shortcut> <source> <intermediate>	Defines the relationship between source and target artifacts, or between two adjacent artifacts in the path.
autoApprove	<shortcut>	If set to true, shortcuts without a path will be set to "Approved" state. Else, they will remain as "Unresolved" shortcuts. Default value is false.
delayedCalculation	<shortcut>	If set to true, shortcut path must be calculated manually or by the system task "Shortcuts Consolidating Task". Else, whenever a shortcut is created or modified, the path will be calculated immediately. Default value is true.
canBeEntered	<shortcut>	If set to true, it enables a user to create a shortcut directly in the relationship editor (For details see "How to Create Shortcut", in *User Guide*). Default value is true.
label	<shortcut>	Relationship label displayed on source artifact.
inverseLabel	<shortcut>	Relationship label displayed on target artifact.
localName	<source> <intermediate> <target>	SDM name of artifacts in the shortcut.
hierarchyRelationshipType	<target>	Defines the relationship used for implied shortcut (Refer "Implied Shortcut" below)

Implied Shortcut

Implied shortcut is an exceptional shortcut where we do not need to specify the intermediate artifacts in the shortcut path. These artifacts are "implied" by the target artifact and a relationship defined by "hierarchyRelationshipType" attribute.

Example of shortcut definition:

```
<shortcut id="projectToBFunctionShortcut" relationshipType="realizes"
autoApprove="true"
```

```
label="Realized business functions" inverseLabel="Realized by projects"
delayedCalculation="true">
    <source localName="projectArtifact" relationshipType="realizes" />
    <target localName="businessFunctionArtifact"
hierarchyRelationshipType="composes"/>
</shortcut>
```

Example of artifacts with relationships:

- *Project 01* realizes *Business Function 01*

- *Business Function 01* composes *Business Function 02*

In this example there are 2 shortcuts between *Project 01*, *Business Function 01* and *Business Function 02*:

- *Project 01--> Business Function 01*

- *Project 01 --> Business Function 01 --> Business Function 02*

- The shortcut between *Project 01* and *Business Function 02* is "implied" by *Business Function 01*.
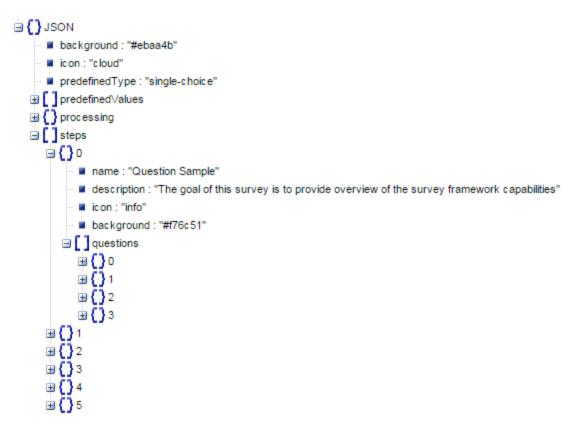
Survey Definition

In Enterprise Maps, a survey is designed after its definition represented by a Survey Definition Artifact in the catalog, which is a specialization of Script Artifact. The survey definition itself is a JSON structure stored as artifact attachment. For more details, see the following topics:

- "Property Mapping Question" on page 93

- "Relationship Question" on page 94

- "Shortcut Question" on page 95

- "Button Question" on page 95

- "Score Calculation" on page 96

- "Post Processing" on page 97

- "Example Script" on page 99

Survey Structure

Here is an overall design of a survey structure in JSON:

```
⊟ { } JSON
    ▪ background : "#ebaa4b"
    ▪ icon : "cloud"
    ▪ predefinedType : "single-choice"
  ⊞ [ ] predefinedValues
  ⊞ { } processing
  ⊟ [ ] steps
    ⊟ { } 0
        ▪ name : "Question Sample"
        ▪ description : "The goal of this survey is to provide overview of the survey framework capabilities"
        ▪ icon : "info"
        ▪ background : "#f76c51"
      ⊟ [ ] questions
        ⊞ { } 0
        ⊞ { } 1
        ⊞ { } 2
        ⊞ { } 3
    ⊞ { } 1
    ⊞ { } 2
    ⊞ { } 3
    ⊞ { } 4
    ⊞ { } 5
```
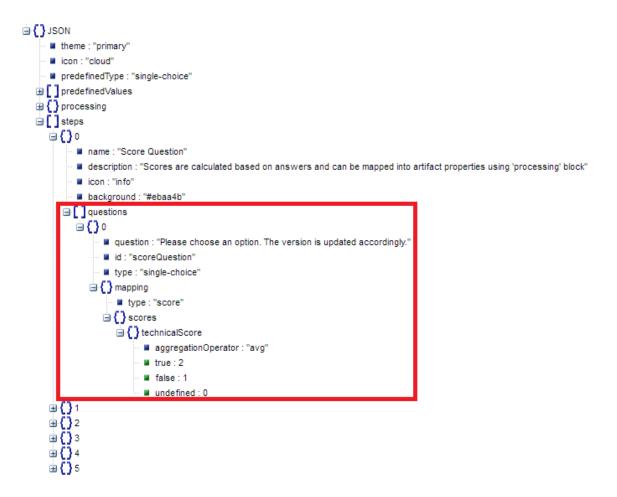
The above structure is illustrated by jsonviewer.stack.hu.

JSON Values

Names	Description
background	theme color of the survey
icon	theme icon of the survey
predefinedType	defines a default question type. It is used if a question has no type
predefinedValues	defines values for the predefined type
processing	mapping of score questions into artifact properties. The same score may be calculated from multiple questions, however it is mapped to a single artifact property only once in the same block.
steps	grouping of questions for a particular domain. For example: security, mobility, etc.
questions	all the questions in a step
onCompletion	defines a function to be called once user clicks *Finish* button.

Question Structure

```
⊟ {} JSON
    ■ theme : "primary"
    ■ icon : "cloud"
    ■ predefinedType : "single-choice"
  ⊞ [ ] predefinedValues
  ⊞ {} processing
  ⊟ [ ] steps
    ⊟ {} 0
        ■ name : "Score Question"
        ■ description : "Scores are calculated based on answers and can be mapped into artifact properties using 'processing' block"
        ■ icon : "info"
        ■ background : "#ebaa4b"
      ⊟ [ ] questions
        ⊟ {} 0
            ■ question : "Please choose an option. The version is updated accordingly."
            ■ id : "scoreQuestion"
            ■ type : "single-choice"
          ⊟ {} mapping
              ■ type : "score"
            ⊟ {} scores
              ⊟ {} technicalScore
                  ■ aggregationOperator : "avg"
                  ■ true : 2
                  ■ false : 1
                  ■ undefined : 0
    ⊞ {} 1
    ⊞ {} 2
    ⊞ {} 3
    ⊞ {} 4
    ⊞ {} 5
```

The above structure is illustrated by jsonviewer.stack.hu.

Question Type

Type	Description
single-choice	represented by radio buttons
multi-choices	represented by check boxes
text	
textarea	multiple-line text which can be mapped into artifact discussion.
number	maps to artifact properties that are of type integer, float or double.
taxonomy	single choice represented by a combo box.
relationship	represented by the relationship editor for a particular relationship (refer "Relationship Question" on page 94).
shortcut	represented by a particular shortcut defined in EM (refer "Shortcut Question" on

Question Type, continued

	).
button	answers the question when you click a button.

Mapping

The answer to a question can be mapped to properties of the surveyed artifact. The supported mapping type is as follows :

- Property: to save the answers into artifact properties.
- Relationship: to add relationship between surveyed artifact and others. This mapping is implied when question type is 'relationship'.

Values

These are pre-defined values which are answers for taxonomy, single-choice or multi-choices questions.

Disabled Questions

Questions can be disabled based on the answers respondents have given to some of other questions simply because it is not required anymore. For example: a user does not need to answer the question "Is storage encrypted?", if they have already answered "No" for another question "Is security required?".

To disable "Is storage encrypted?", set doNotAnswerWhen element as follows:

```
{ question: 'Is storage encrypted ?',
doNotAnswerWhen: [ { questionID: 'securityRequired', values:['false']} ],
…
}
```

A question's answer can lead to disabling other questions in the same step or in subsequent steps. If all questions in the subsequent step are disabled, the particular wizard containing these steps itself is skipped.

Showstopper Questions

A question can be defined as a showstopper question to end the survey if no further answers are required. Once a showstopper is triggered, it disables not only all the questions behind it within the same step but also the steps behind the current one.

Showstopper is applicable only to single-choice, multiple-choice and taxonomy questions. If multiple answers are defined as showstoppers in the same question then showstopper is triggered by any of them.

Showstopper questions are defined as below:

```
{
        question: 'Enter the business function type',
        type: 'taxonomy',
        id: 'businessFunctionType',
        values: [
                {'commodity': 'Commodity'},
```

```
                  {'innovation': 'Innovation',
                    showStopper: true
                  }]
}
```

Property Mapping Question

Answers to a survey can be stored inside artifact properties or its financial profile properties using 'mapping' tag. This tag can be used for data types such as number, text and textarea question.

The following attributes are accepted under 'mapping' tag:

Attribute	Description
type	Type of this property mapping. It must be 'property' or 'financial-property'.
property	SDM property name of the surveyed artifact or its financial profile.

Property Mapping

Maps answers to a property of the surveyed artifact

Example:

```
{
    question: 'How many servers are utilized for this application ?',
    id: 'numberOfServers',
    type: 'number',
    mapping: {
        type: 'property',
        property: 'estimatedNumberOfServers'
    }
}
```

Financial Property mapping

Maps answers to a property of financial profile of the surveyed artifact.

> **Note:** This mapping can be used only if the surveyed artifact has associated financial profile (For example: Application Component and Project). If the surveyed artifact has no financial profile at the time the survey is taken, a new profile is created automatically upon completion of the survey.

```
{
    question: 'What are the current annual 3rd party software license costs related
to this application ?',
    id: 'currentAnnualCostSw',
    type: 'number',
    mapping: {
        type: 'financial-property',
        property: 'annualCostSw'
    }
}
```

Relationship Question

A relationship question is represented by the relationship editor for a particular relationship. Respondents answer the questions by linking or removing artifacts from the relationship defined for the question. This allows respondents to make real changes to the surveyed artifact. The changes are effective immediately even before the survey is over.

How to Define a Relationship Question

Relationship question is defined below:

```
{
        question: 'Please specify the sub-components of artifact:',
        type: 'relationship',
        id: 'subComponents',
        mapping: {
            relationshipName: 'composedOf',
            label: 'Sub-Components',
            artifactTypes: ['applicationComponentArtifact'],
            tableViewOnly: true,
            readOnly: false,
            deleteOnly: false,
            usedType: 'some value'
        }
},
```

Name	Description	Default Values
relationshipName	relationship type defined in SDM	
artifactTypes	SDM names of target artifacts	
tableViewOnly	switches the relationship editor between inline and pop up table view	false
readOnly	enables/disables editing in the relationship editor	false
deleteOnly	allows the erase action only	false
useType	defines the value of useType attribute for the created relationship. When used with providedBy relationship, the relationship editor displays only contacts with the same useType. If useType is invalid, all contacts are displayed.	
label	allows to set display name for this relationship in popup mode (tableViewOnly=false).	

Shortcut Question

A shortcut question allows respondents to answer a question by creating shortcuts between the surveyed artifact and target artifact. As is the case with relationship question, changes to a shortcut question are effective immediately, even before the survey is over.

How to Define a Shortcut Question

As given below:

```
{
    question: 'Please specify the application services realized by this component: ',
    type: 'shortcut',
    id: 'realizedAppService',
    mapping: {
        shortcutId: 'appComponentToAppService',
        label: 'Realized Application Services',
        showLabel: 'true'
    }
},
```

Name	Description	Default Values
type	must be 'shortcut'	
shortcutid	shortcut ID defined in EM	
label	overwrites the label defined in the shortcut	
showLabel	shows or hides the shortcut label (true/false)	true

Button Question

A button question is defined as follows:

```
{
    question: 'Download the spreadsheet template from the following link, fill it in and use the same page to upload the result into Enterprise Maps',
    id: 'downloadSpreadsheetAppDeployment',
    type: 'button',
    extraParams: {
        label: 'Import application deployments',
        icon: 'arrow-right',
        url: 'SERVER_URI/../../../service-catalog/common/imports/csvImport?taskLabel=Spreadsheet&artifactType=infrastructureServiceArtifact',
        click: '$("#done-button").trigger("click")'
    }
}
```

Name	Description	Default Values
type	must be 'button'	
extraParams	defines attributes of the button	
label	label of the button	
icon	icon for the button	
url	opens the URL when you click the button	
click	method to call when you click the button	click *Finish* button of the survey

Score Calculation

Score is a measure from 0 to 100 of the capability or suitability of an artifact. It is calculated based on multiple questions. Each of the answers to a score question have a value assigned to it. When multiple users answer a common question for the same artifact the score is calculated cumulatively using an aggregation operator - minimum, maximum or average (by default).

Score of a given artifact is calculated from the sum of scores of individual questions aggregated from the respondent's answers, divided by sum of the question's scores, taking the highest value for each question, multiplied by 100.

It is also possible to calculate the score of questions even if skipped or not answered by the respondent. Use 'skipped' keyword to define such scores.

For example :

Assuming we have 2 questions that define 'technicalScore' as follows :

```
Question 1:
technicalScore: { aggregationOperator: 'avg', 'true': 50, 'false': -
20, 'undefined': 0, 'skipped': -30 }
Question 2:
technicalScore: { aggregationOperator: 'max', 'true': -10, 'false': 0,
'undefined': 0, 'skipped': -25 }
```

The system then computes the predefined values of this survey for 'technicalScore' as follows :

- SurveyMinScore = max(0, question 1 min + question 2 min) = max (0, -20 -10) = 0

- SurveyMaxScore = max(0, question 1 max + question 2 max) = max (0, 50 + 0) = 50

After the first respondent answers true/true, the total score
must be :

Questions	Scores by User A	Aggregated Scores

After the first respondent answers true/true, the total score
must be :, continued

| Question1 (Average) | 50 | Avg(50) = 50 |
| Question2 (Max) | -10 | Max(-10) = -10 |

SurveyAbsoluteScore = Max(0,50 -10) = 40

SurveyTotalScore = (SurveyAbsoluteScore – SurveyMinScore) / (SurveyMaxScore – SurveyMinScore) X 100% = (40-0) / (50-0) X 100% = 80%

After the second respondent answers false/false, the total
score must be :

Questions	Scores by User B	Aggregated Scores
Question1 (Average)	-20	Avg(50,-20) = 15
Question2 (Max)	0	Max(-10, 0) = 0

SurveyAbsoluteScore = Max(0,15+0) = 15

SurveyTotalScore = (SurveyAbsoluteScore – SurveyMinScore) / (SurveyMaxScore – SurveyMinScore) X 100% = (15-0) / (50-0) X 100% = 30%

Score mapping is a special case of property mapping in which the calculated scores are directly stored into properties (using "processing" block) of the surveyed artifact or are further computed and stored (used `onCompletion` function). For more details, see "Post Processing" below.

Note: Score values can only be mapped to the artifact properties with type Double.

Post Processing

In a survey definition, it is possible to manipulate the calculated scores after a respondent finishes the survey. This manipulation must be defined in `onCompletion` function as below. The method is invoked after the respondent completes the survey and before the data gets saved into HP EM.

```
onCompletion: function(surveyedArtifact, questions, calculatedScores) {
  var avgScore = calculateGroup('avg', surveyedArtifact, 'c_cloudAssessment_
TechnicalAlignment');
  setProperty(surveyedArtifact, 'technicalAlignment', avgScore);
}
```

Predefined Objects

The following predefined objects simplify the usage of this function. Some are required as arguments to this function.

surveyedArtifact

- The artifact being surveyed. An instance of `com.hp.systinet.repository.sdm.ArtifactBase`
- Required as an argument

questions

- All questions included in the surveys. An instance of `org.json.JSONObject`
- For answers to a particular question, use its type (via invoking relevant functions) and id. For example: to get the answer of a text question with id storageEncryption, use `questions.getString('storageEncryption')`
- Supported functions: `getString()`, `getBoolean()`, `getDouble()`, `getInt()`, `getLong()`
- Required as an argument

calculatedScores

- All the scores defined in the survey definition. An instance of `org.json.JSONObject`. Basically, it is a key-value object storing the score and its value: *{"technicalScore":100,"financialScore":0}*
- To get the score, use the score name. For example: to get the score calculation of *technicalScore*, use `calculatedScores.getDouble('technicalScore')`

Others

Since server-side script (Rhino context) is used, we can leverage on other useful utilities such as `repositoryService`, `scriptEnvironmentUtils` etc. as in Javascript assertions.

Predefined Functions

calculate(operator, artifact, String[] properties)

- Calculates the score based on individual scores mapped to artifact properties
- Supported operators: 'avg', 'min', 'max'

  Example: score mapping

```
processing: {
        scores: [{// declare the scores defined by the questionnaire and map
        them to properties
                name: 'regulatoryScore',
                property: 'regulatoryAlignment'
        },
        {
                name: 'vendorSupportScore',
                property: 'vendorSupportAlignment'
        },
        {
                name: 'geographyScore',
                property: 'geographyAlignment'
        },
        {
                name: 'serviceAvailabilityScore',
                property: 'serviceAvailabilityAlignment'
        },
        {
                name: 'workloadVariabilityScore',
                property: 'workloadVariabilityAlignment'
        },
        {
                name: 'securityScore',
```

```
                property: 'securityAlignment'
        }]
},
```
Business alignment score can be calculated and set to `businessAlignment` property as below:

```
var businessAlignmentScore = calculate('avg', surveyedArtifact,
                                    ['regulatoryAlignment',
                                    'vendorSupportAlignment',
                                    'geographyAlignment',
                                    'serviceAvailabilityAlignment',
                                    'workloadVariabilityAlignment',
                                    'securityAlignment']);
setProperty(surveyedArtifact, 'businessAlignment', businessAlignmentScore);
```

calculateScores(operator, surveyDefinitionUuid, art, String[] scores)

- Calculates the score based on individual scores defined in the survey definition.

- surveyDefinitionUuid: uuidstring of the survey definition. For the current definition, use it literally as *surveyDefinitionUuid*.

```
var businessAlignmentScore = calculateScores('avg', surveyeDefinitionUuid,
surveyedArtifact,
                                    ['regulatoryScore',
                                    'vendorSupportScore',
                                    'geographyScore',
                                    'serviceAvailabilityScore',
                                    'workloadVariabilityScore',
                                    'securityScore']);
setProperty(surveyedArtifact, 'businessAlignment', businessAlignmentScore);
```

calculateGroup (operator, art, propertyGroup)

- Calculates the score based on individual scores mapped to artifact properties and defined by a property group. For example:

```
var businessAlignmentScore =n calculateGroup('avg', surveyedArtifact,
'c_cloudAssessment_BusinessAlignment');
setProperty(surveyedArtifact, 'businessAlignment', businessAlignmentSconre);
```

setProperty(artifact, propertyName, newValue)

Sets a value for the property of an artifact

getPrimitiveValue(artifact, propertyName)

Returns the value (must be primitive type) for a property of an artifact

Example Script

Here is an example of how survey is created :

```
/*
*********      SURVEY SAMPLE       ***********
**
** Survey STRUCTURE:
**
**   predefinitions: type, value
**   processing: mapping properties of scores
**   steps: each step will have 1-many questions
**
**
** Question TYPES:
**
**   'single-choice' : radio buttons with default values: { 'Yes' : true, 'No' :
false, 'Don\'t know': undefined }
**   'textarea'      : mapped to string property (long text)
**   'text'          : mapped to string property (short text)
**   'number'        : mapped to integer or double property
**   'button'        : open in UI only
**
** Question MAPPING:
**   An answer of question or its calculated score can be saved to artifact
property
**
** Question DEPENDENCY (Conditional Questions)
**   Question dependency is applicable to single-choice and taxonomy questions
only. A question can be defined as dependent on other questions as below:
**   doNotAnswerWhen: [{questionID:'id1', values:['false']}, {questionID:'id2',
values:['yes']}]
**
** Default VALUES:
**
**   predefinedType: if question has no type, it will use the default
**   predefinedValues: predefined option values of default type
**
** SCORES:
**   question[@type='score']: each question which type is score will have 1-many
type of scores: technical score, financial score,...
**   Each type score, 1 of 4 ways calculation can be set: { 'avg' : average score,
'min' : minimum score, 'max': maximum score, 'first': first answer }
**   avg is default
**
** COLOR PATTERNS:
**
**   Theme: blue(default): #5fa2dd, green: #9cc96b, purple: #986291, orange:
#ebaa4b, red: #f76c51, jade green: #4ebcda
**   Icons: cloud, user, info, home, heart, gift, signal, gear, file, trash,
clock-o, road, lock, inbox, flag, tags, list
** or all the icons which defined in font awesome. See more:
http://fortawesome.github.io/Font-Awesome/icons/
 */
```

```
{
    /*
        You can custom background/buttons by predefined theme values: 'primary',
'success', 'info', 'mint', 'purple', 'pink', 'dark', 'warning', 'danger'
    */
    theme: 'primary',
    icon: 'cloud',
    predefinedType: 'single-choice',
    predefinedValues: [{
            'true': 'Yes'
    },
    {
            'false': 'No'
    },
    {
            'undefined': 'Don\'t know'
    }],
    processing: {
            scores: [
            {
                    name: 'technicalScore',
                    property: 'technicalAlignment'
            }]
    },
    steps: [
    {
            name: 'Score Question',
            description: 'Scores are calculated based on answers and can be
            mapped into artifact properties using \'processing\' block',
            icon: 'info',
            background: '#ebaa4b',
            questions: [{
                    question: 'Please choose an option. Technical alignment
                    is updated accordingly.',
                    id: 'scoreQuestion',
                    type: 'single-choice', //if not defined, default value
                    is 'single-choice'
                    mapping: {
                            type: 'score',
                            scores: {
                                    technicalScore: {
                                            aggregationOperator: 'avg',
                                            'true': 2,
                                            'false': 1,
                                            'undefined': 0
                                    }
                            }
                    }
            }]
    },
```

```
{
        name: 'Property Question',
        description: 'Property questions have answers mapped to artifact
        properties',
        icon: 'signal',
        background: '#9cc96b',
        questions: [
        {
                question: 'Please describe this artifact',
                type: 'textarea',
                id: 'textareaQuestion',
                mapping: {
                        type: 'property', // save the answer into description
                        of artifact
                        property: 'description'
                }
        },
        {
                question: 'Please enter ROI',
                type: 'number',
                id: 'numberQuestion',
                mapping: {
                        type: 'financial-property',
                        property: 'returnOnInvestment' // number can be
                        mapped to integer/float property
                }
        },
        {
                question: 'Please enter estimated number of servers',
                type: 'number',
                id: 'numberOfServers',
                mapping: {
                        type: 'property', // save the answer into Estimated
                        Number of Servers
                        property: 'estimatedNumberOfServers'
                }
        }]
},
{
        name: 'Relationship Question',
        description: 'Relationship questions form new relationships between
        surveyed artifact with other artifacts. This take effect immediately
        even without the survey submitted.',
        icon: 'sitemap',
        background: '#f76c51',
        questions: [
        {
                question: 'What are the sub-components of this application
                component',
                type: 'relationship',
```

```
                        id: 'relationshipQuestion1',
                        mapping: {
                                relationshipName: 'aggregatedBy',
                                label: 'Sub-Component of',
                                artifactTypes: ['applicationLayerArtifact'],
                                tableViewOnly: true,
                                readOnly: false,
                                deleteOnly: false
                        }
                },
                {
                        question: 'Please specify the Business Owner for this
                        application component',
                        type: 'relationship',
                        id: 'relationshipQuestion2',
                        mapping: {
                                relationshipName: 'providedBy',
                                label: 'Business Owner',
                                artifactTypes: ['personArtifact'],
                                useType: 'businessExpert',
                                tableViewOnly: false
                        }
                },
                {
                        question: 'What the business functions this project realizes',
                        type: 'shortcut',
                        id: 'shortcutQuestion',
                        mapping: {
                                shortcutId: 'projectToBFunctionShortcut', // only
                                for Project
                                label: 'Realized Business Function',
                                showLabel: false
                        }
                }]
        },
        {
                name: 'Show-stopper Question',
                description: 'Show stopper questions end survey immediately when a
                show-stopper choice is answered.',
                icon: 'signal',
                background: '#9cc96b',
                questions: [
                {
                        question: 'Is there a business motivation?',
                        type: 'single-choice',
                        id: 'showstopperQuestion',
                        values: [{
                                'true': 'Yes'
                        },
                        {
```

```
                            'false': 'No',
                            showStopper: true
                }]
        }]
},
{
        name: 'Disabled Question',
        description: 'Disabled questions are skipped based on answers of
        other questions',
        icon: 'lock',
        questions: [
        {
                question: 'Is encryption required?',
                id: 'disabledQuestion1'
        },
        {
                question: 'Are the application\'s data encrypted?',
                id: 'disabledQuestion2',
                doNotAnswerWhen: [{questionID:'disabledQuestion1',
                values:['false']}]
        }]
},
{
        name: 'Finish',
        description: 'Thank you for participating in this survey',
        icon: 'check',
        background: '#4ebcda'
}]
}
```

Chapter 6: XML Publishing

HP EM supports "Scripted XML Publishing" which extends HP EM publishing by using script artifacts. These script artifacts contain instructions that recognize and parse new XML document types that are imported from the HP EM UI.

The script artifacts have the following capabilities:

- Can be created, modified, or deleted any time.
- Can modify the HP EM publishing pipeline immediately without the need of applying extra extensions or a server restart.
- Can be easily bundled (by PSO) in a model extension to provide a default way of publishing new document types/artifacts.
- Are written in an XML format using the XML schema (XSD). Your XML editor can easily be configured to provide hints and help regarding script structure.
- Are controlled for XML schema validation and pass semantic analysis upon every change (create/update). Invalid scripts cannot be published.

Creating Scripted XML Artifacts

The script can be published to HP EM using the following simple steps.

To create a scripted XML artifact:

1. Select the **Administration** Tab **> Customization > Manage Scripts**.
2. Click the **Create new script** icon to open the Managed Script editor.
3. Enter a unique name (spaces are OK) and an optional description.
4. For **Script Language**, select **XML**.
5. For **Execute On**, select **XML Data Import**.
6. Click **Save**. A view page of the script opens.
7. Click the **Edit Script** button on the right to open a blank Edit script dialog.
8. Enter the script content and click **Save**. You can copy and paste the script from a text file.
   For example, you could enter the following script:

```
<publisherConfiguration
xmlns="urn:com.hp.systinet.publishing.xml:1.0"
name="simpleBook">
<artifact sdmName="documentationArtifact">
<recognition>
<rootElement name="book" namespace=""/>
</recognition>
<stringProperty sdmName="name">
<text>BOOK: </text>
```

```
<xpath>name/text()</xpath>
</stringProperty>
</artifact>
</publisherConfiguration>
```

Every successfully saved script is active immediately. You can now import and publish a book file. See "Importing and Publishing a Book File" below.

Importing and Publishing a Book File

To import a book file:

1. Select the **Catalog** tab **> Import > File**.

2. Change **File** to **URL** and enter the URL of the simple book.

3. Click **Import**. A documentation artifact is created.

Example:

If the script content of the book is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="urn:simpleBook">
<name>Dunno on the Moon</name>
</book>
```

A documentation artifact is created and the name of the created artifact is **BOOK: Dunno on the Moon**.

Script Properties

This topic describes:

- "Enhanced Script Components" below
- "Script Elements and Attributes " on the next page

Enhanced Script Components

The script is enhanced with the following parts:

- namespace is **urn:com.hp.systinet.publishing.xml:1.0**
- root element is **publisherConfiguration**
  - The root element should have the **name** attribute to identify the script by name. The name must be unique between all publishing scripts.

- *optional* **registration** element is important during recognition of a document/file type. For details see "Recognition Order" on page 111.

- *optional* **namespaceContext** element defines mapping from prefix to namespace. This mapping is supplied to all xpath elements (xpath,select) in the script.
- recognition and parsing of input filesare controlled by nested **artifact** elements, which are optional (but a script without artifact elements is useless).
  - each artifact must have the **sdmName** attribute that matches an existing SDM artifact type name.

  - an artifact element can be marked with enabled='false' to ignore the artifact during publishing.

Script Elements and Attributes

The script is supported by the following elements and attributes:

- "Artifact Recognition " below
- "Extractors" on the next page
- "Artifact Properties " on page 109
- "Relation Property" on page 110
- "Recognition Order" on page 111
- "Variables" on page 112

Artifact Recognition

The **recognition** element defined in the **artifact** element is used to describe how the publisher can recognize the input document/file. The recognition element can contain multiple sub-elements. The artifact element recognizes the input if all recognition's children elements evaluate to **true**.

The following recognition elements are supported:

- **and** is a container element that returns **true** only if all nested recognition elements returns **true**. The artifact's recognition element is an **and** in fact.
- **or** is a container element that returns true only if at least one nested recognition element returns **true**.
- **not** returns **true** if the mandatory nested recognition element returns **false**.
- **rootElement** can contain **namespace** and **name** attributes, returns true only if (all) the following conditions are **true**.
  - name attribute is not supplied or root element local-name (name without namespace prefix) equals to name attribute value.

  - namespace attribute is not supplied or root element namespace equals to namespace attribute value.

- **xpath** contains text with an xpath expression, returns true only if the xpath expression is true (non-empty node-set also yields to true).
- **empty** is a container element that returns true only if all nested extractors create empty value.
- **extension** contains text with a required extension. Returns true only if the input file extension is exactly the same. the extension always start with ".", for example ".xml" or ".doc".

- **any** can contain nested extractor element, it returns **true** only if
  - it does not contain a nested extractor element.

  - or the extractor extracts a non-empty string.

- **xml** returns **true** only if the input document was fully parsed and it is a valid XML file.

Extractors

Extractors are used to extract string data out of the input (file/document or document part). Extractor elements are obviously used to define artifact property value, but they can be also used to recognize data (using empty or any recognition elements described in the Artifact recognition section) or define variables (see "Variables" on page 112).

The following extractors are supported:

- **all** is a container extractor that returns concatenation of values that are created by nested extractors.
- **first** is a container extractor that returns the value of the first nested extractor that creates a value.
- **for-each** is a container extractor that returns concatenation of nested extractors evaluated against each node in the node list defined by the select element; it has to contain:
  - **select** element as a first element with a text defining an xpath to extract a node list.

  - at least one extractor element to extract data for an element in the node list.

- **xpath** returns a string value that is a result of xpath evaluation, where xpath is a text value of this element; xpath value may only use namespace prefixes defined by **namespaceContext** element (defined as a direct child of the root element of the script).
- **regexp** is a container extractor that concatenates the output of all nested extractors and then applies a regular expression pattern to the extracted value. The pattern must be specified in the pattern attribute. It returns a value only if the pattern matches. The value is then either the first substitution group found in the pattern or the whole matched string if there is no substitution group in the pattern.
  - example1: input "file.xml", pattern ".xml$" ... does not match

  - example1: input "file.xml", pattern "^.*\.xml$" ... matches and output is "file.xml"

  - example2: input "file.xml", pattern "^(.*)\.xml$" ... matches and output is "file"

- **replace** is a container extractor that concatenates the output of all nested extractors and then replaces all occurrences of supplied regular expression pattern (using mandatory pattern attribute) by the value supplied in the mandatory replacement attribute.
- **text** returns the value of the text content.
  - example: <text>This is a constant value</text>

- **location** returns the location of the input file as it would appears in the repository location space.
- **variable** returns the value of the variable. It supports the following attributes:
  - **name** is a mandatory variable name.

  - **default** is an optional default value if the variable is not defined.

- **scope** is an optional scope to look for variables (local, shared, all).

- **substituteVariables** is a container extractor that concatenates the output of all nested extractors and then replaces all occurrences of supplied regular expression pattern (using mandatory pattern attribute) by variables. The mandatory pattern attribute must contain a substitution group to know the name of the variable.
  - Example:
    If you input: "This is ${NAME}", pattern is "\$\{.*\}"; if NAME variable is defined to "Pavel".

    Then the output is: "This is Pavel", otherwise the output is "This is ${NAME}."

- **if** extracts a value when a condition matches, it has to contain nested
  - **condition** element that with the condition expressed as recognition element, it is in fact an and extractor container.

  - **value** element that with the value, it behaves as all extractor container.

Artifact Properties

There are several types of artifact properties, the handling of the property depends on the property type. The way how the properties are set to artifact is defined by any of the supported property element of the script's artifact element. These are:

- **stringProperty** defines string/text property with single occurrence(0..1 or 1..1)
  - is an instance of all extractor element, nested elements extracts a text value that is set, the property value depends on property type:

    string — the text is set as property value

    boolean — the value is true only if the text is "true", false otherwise

    category — the text is a category value, which is used to create a category value

    nameUrlPair — text is set to the URL portion of the property value

- **booleanProperty** defines boolean property with single occurrence
  - is an instance of and recognition element, nested elements evaluate the input as either **true** or **false**.

- **integerProperty** defines integer property with single occurrence
  - is similar to stringProperty, but the value is parsed to be an integer

- **dateProperty** defines date property with single occurance, it has
  - zero, one more **format** elements that are used to try parsing the value in the order that they appear, the content of the format element is a string, one of the following values are accepted:

  - **epoch** - a long value is expected as a count of milliseconds from epoch (the date is then constructed using new java.util.Date(millis))

  - **current** independently on value, it result in a current Date

  - **default** default format is ISO8601 (SimpleDateFormat with yyyy-MM-dd'T'HH:mm:ss.SSS'Z'

pattern in "UTC" timezone and lenient parsing)

- any other value is used as a pattern to create java.text.SimpleDateFormat

- the format element can have the following attributes
  - **lenient** means that the parsing will try some heuristics with inputs that do not strictly match the pattern, true by default

  - **timezone** is a timezone string see java.util.TimeZone for details

- **value** is an all extractor that specifies the value to extract from

- **categoryProperty** defines a category property by using either
  - only a **value** attribute is used to specify a category value, associated category's tModelkey and name are obtained from the property descriptor and HP Atlas database. The category value is validated during creation of the script, it has to be a category of a checked taxonomy.

  - a **val** element can be used to specify an all extractor that extracts the category value from the input; name (3rd) and taxonomyUri (1st) are other optional elements that are all extractors as well.

  - if the **taxonomyUri** is not specified, category's taxonomyURI is obtained from associated property descriptor (recommended)

  - if the **name** is not specified, category's value is queried from the HP Atlas database using taxonomyURI and value

- **relation** defines relational property with single occurrence and it is quite complex to understand, see "Relation property" section below.

- **multiProperty** defines property with multiple occurrences (?..n) using nested select element and one 'single occurrence' property. The select element must be the first, it defines an xpath expression that used to split the present input into a node list, where each node is then used to parse a single occurrence property.

Each 'single occurrence' property has an 'sdmName' attribute that carries the name of the property according to SDM model, it can also contain a flag attribute **identifier**. All artifact's properties that are marked as identifiers are considered to be a composed identifier of the artifact, which is used when finding duplicates. A new artifact is considered as a duplicate if it has the same SDM name and identifiers of another artifact that already exists.

Relation Property

Unlike other properties, the value of the relation property is not known from the input. The value value of the relational property represents a connection to another artifact, that need not exists during the publishing. The process of creating a relation can typically create also a target artifact.

Each **relation** element must have **sdmName** attribute to define property name in the SDM model. There are the following type of relation properties:

- relation to local artifact means that relation's second side (target) is an artifact instance that exists (or is created) locally for this artifact. It is defined with

- targetType attribute undefined or set to "localArtifact".

- Nested artifact element that defines the target artifact; in order to resolve duplicates, at least one property should be marked by identifier attribute set to true.

- Semantical example: book with chapters where
  - chapters are local to the book
  - more chapters with the same name can exist in the repository

- file reference means that a relation is represented by another file in the input. It is defined with
  - **targetType** attribute undefined or set to "importedResource"

  - Nested **import** element that defines how to include the resource. The import element must contain these elements:
    - **relativeLocation** element being an all extractor that is supported to create relative or absolute URL that can be used to download the resource.
    - **targetSdmName** elements can follow, each with a text value that must be an SDM name of expected target artifact; **targetSdmName** elements can be only used to enforce specific target artifact sdm names, when no **targetSdmName** is present, the target types are taken from the relationship descriptor.

  - Semantical example: a library references a large number of books. An HTML page references images and CSS files.

- **relation to shared artifact**, it means that relation's second side (target) must be a shared artifact instance. It is defined with the following:
  - **targetType** is used to set up a method of how to find a shared artifact.Following are methods used:
    - **repositoryReference** - a matching artifact is searched the repository.
    - **sharedReference** - a matching artifact searched in the publisher input; if it is not found, use **repositoryReference**.
      - publisher input contains all resources that were recognized in the "still running" processing; including locally decomposed artifacts, imported resources, and all files in the zip file.
    - **sharedArtifact** - if no sharedReference is found, create a new artifact

  - Nested artifact element that defines the target artifact; in order to resolve duplicates, at least one property should be marked by identifier attribute set to true.

  - Semantical example: book with author where
    - author is assumed to be only one author in the repository.
    - author is shared between books.

Recognition Order

The publishing pipeline internally manages a list of DocTypeFactory instances. These instances are called (in the list order) to create DocType instances that are asked to recognize the input.

- The first DocType that recognizes the input is used to parse the data and eventually create artifact (s).

- A DocTypeFactory instance is created by runtime for every publishing script. The factory creates DocType for every artifact element in the script (in the same order).

- The server log contains INFO messages that describe the order of DocTypeFactory instance. These INFO messages are generated during HP EM EAR initialization or upon a change in publishing scripts (create/delete/update).

```
14:17:09,369 INFO [PublishingScriptsRegistration] Registering
factories:
14:17:09,369 INFO [PublishingScriptsRegistration]
ScriptedDocTypeFactory[bookWithChapters_withVariables]
14:17:09,370 INFO [PublishingScriptsRegistration]
ScriptedDocTypeFactory[simpleBook]
14:17:09,370 INFO [PublishingScriptsRegistration]
ScriptedDocTypeFactory[helloworldPublisher]
14:17:09,370 INFO [PublishingScriptsRegistration]
ScriptedDocTypeFactory[simpleUddiPublisher]
14:17:09,371 INFO [PublishingScriptsRegistration] DocTypeFactoryImpl
[sc-publishing-ext.docTypes]
14:17:09,371 INFO [PublishingScriptsRegistration] DocTypeFactoryImpl
[sc-publishing-sca.docTypes]
14:17:09,371 INFO [PublishingScriptsRegistration] DocTypeFactoryImpl
[sc-publishing-wsdl.docTypes]
14:17:09,372 INFO [PublishingScriptsRegistration] DocTypeFactoryImpl
[sc-publishing.docTypes.default]
```

DocTypeFactoryImpl instances are built-in factories that are used to recognized documents such as WSDL, SCA, and XPDL. These are by default at the bottom of the list. The ScriptedDocTypeFactory instances are created out of publishing scripts. The order in which DocType instances are asked if they recognize the input can be changed in the script.

You can do the following:

- Add a **registration** element as a first child of the publishing script's root element. This element can contain multiple **after** or **before** elements, both with mandatory text content that should be the factory name (script name in the case publishing script), for example <after>helloworldPublisher</after><before>simpleUddiPublisher</before>. Note that the names are listed in the log. You may also use the names of built-in DocTypeFactoryImpl instances if required.

- Change the order of artifact elements in the particular publishing script will change the recognition order managed by the ScriptedDocTypeFactory.

Variables

Variables can be used to simplify and/or speed up the script execution. A variable can be set using optional **setVariable** element that can occur multiple times as a first child of the **artifact** element. The **setVariable** element contains

- Optional scope element that identifies the scope of the defined variable(s), **local** means local for the published file/document, **shared** means a variable that is visible between all published files

- Mandatory **name** element is an **all** extractor and defines variable name

- Mandatory **value** element is an **all** extractor and defines variable value

- Optional **select** element can be used to set up multiple variables, the select element text must be an xpath expression that is used to create node list, each node in the list is then used to define variable (using extractors of setVariable's name and value elements)

Variables are used in **variable** or **substituteVariables** extractors, which are explained in the "Extractors" section above.

Read more supported elements and attributes in **publisherConfiguration_1.0.xsd**.

Scripted XML Samples

The following topics provide different use-case samples:

- "Sample 1: Publish a Book With All Its Chapters" below

- "Sample 2: Cross-Reference to Another Book" on the next page

- "Sample 3: Ignore Some Book Files or Document Types" on page 115

- "Sample 4: Books Share the Same Author" on page 116

Sample 1: Publish a Book With All Its Chapters

You want to publish a book with all its chapters, which are artifacts connected to the book. A single input book file now differs:

- The book file has a namespace: "urn:bookWithChapters"

- It has children element chapters. Each chapter has the attribute "name" with chapter name

```
<book xmlns="urn:bookWithChapters">
 <name>Dunno on the Moon</name>
 <chapter name="The mystery of moon stone"/>
 <chapter name="Upside down"/>
 <chapter name="Start"/>
 <chapter name="Landing"/>
 <chapter name="The first day on the Moon"/>
</book>
```

The associated script is:

```
<publisherConfiguration
xmlns="urn:com.hp.systinet.publishing.xml:1.0"
name="bookWithChapters"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="urn:com.hp.systinet.publishing.xml:1.0
publisherConfiguration_1.0.xsd">
<namespaceContext>
```

```
<namespace prefix="b" uri="urn:bookWithChapters"/>
</namespaceContext>
<artifact sdmName="documentationArtifact">
<recognition>
<extension>.xml</extension>
<rootElement name="book" namespace="urn:bookWithChapters"/>
</recognition>
<stringProperty sdmName="name">
<text>BOOK: </text>
<xpath>b:name/text()</xpath>
</stringProperty>
<multiProperty>
<select>b:chapter</select>
<relation sdmName="r_consistsOf" targetType="localArtifact">
<artifact sdmName="documentationArtifact">
<stringProperty sdmName="name" identifier="true">
<text>CHAPTER '</text>
<xpath>@name</xpath>
<text>' of '</text>
<xpath>/b:book/b:name/text()</xpath>
<text>'</text>
</stringProperty>
</artifact>
</relation>
</multiProperty>
</artifact>
</publisherConfiguration>
```

Create the script using the Administration tab in HP EM. Then import an example file (**Catalog** tab **> Import > File**). The result is a documentation artifact **BOOK: Dunno on the Moon** that has a relation that consists of' 5 documentation artifacts named:

- CHAPTER 'The mystery of moon stone' of 'Dunno on the Moon'
- CHAPTER 'Upside down' of 'Dunno on the Moon'
- CHAPTER 'Start' of 'Dunno on the Moon'
- CHAPTER 'Landing' of 'Dunno on the Moon'
- CHAPTER 'The first day on the Moon' of 'Dunno on the Moon'

Sample 2: Cross-Reference to Another Book

Assume that the book requires another book in order to know the context before reading. A book XML file is now modified with reference to another book (as a relative/absolute URL). using readAfter elements:

```
<book xmlns="urn:bookWithChapters">
<name>Dunno on the Moon</name>
```

```
<chapter name="The mystery of moon stone"/>
<chapter name="Upside down"/>
<chapter name="Start"/>
<chapter name="Landing"/>
<chapter name="The first day on the Moon"/>
<readAfter ref="./bookWithChapters_sunCity.xml"/>
</book>
```

The associated changed publishing script adds the following property definition:

```
<multiProperty>
<select>b:readAfter</select>
<relation sdmName="r_dependsOn">
<import>
<relativeLocation>
<xpath>@ref</xpath>
</relativeLocation>
<targetSdmName>documentationArtifact</targetSdmName>
</import>
</relation>
</multiProperty>
```

Use the administration UI to update the script with the reference. Then you can re-import (Catalog tab, Import/File). Now the publishing includes also the reference, the result now contains:

- A new 'BOOK: Dunno in Sun City' documentation artifact with 2 chapters (document artifacts) that are connected.

- 'BOOK: Dunno on the Moon' now has relationship 'r_dependsOn' to 'BOOK: Dunno in Sun City'.

Sample 3: Ignore Some Book Files or Document Types

In some cases, HP EM may need to ignore some book files or document types during publishing. This usecase is also supported by scripted publishing.

An artifact element can have attribute enabled set to false to ignore the recognized input file. For example: the following artifact element in your script will ignore books that have no chapters:

```
<artifact sdmName="documentationArtifact" enabled="false">
<recognition>
<xpath>/b:book[not(b:chapter)]</xpath>
</recognition>
</artifact>
```

Modify the existing to have the above artifact element as a first artifact and then import data file. The publishing will result with a message "No modification". The file bookWithoutChapters.xml will not be imported, and thus ignored.

Sample 4: Books Share the Same Author

Authors (unlike chapters) are shared between books, the author is a **sharedArtifact** between all books in the repository.

The associated changed publishing script adds the following property definition:

```
<multiProperty>
<select>b:author</select>
<relation sdmName="documentationOf" targetType="sharedArtifact">
<artifact sdmName="personArtifact">
<stringProperty sdmName="name" identifier="true">
<xpath>text()</xpath>
</stringProperty>
</artifact>
</relation>
</multiProperty>
```

Create the script using administration UI. Then import an example file:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="urn:bookWithChapters">
<name>Dunno on the Moon</name>
<author>Nikolay Nosov</author>
<chapter name="The mystery of moon stone"/>
<chapter name="Upside down"/>
<chapter name="Start"/>
<chapter name="Landing"/>
<chapter name="The first day on the Moon"/>
<readAfter ref="bookWithChapters_sunCity.xml"/>
</book>
```

The books will then contain a shared reference to a person artifact **Nikolay Nosov**.

Chapter 7: CSV Import and Export Tools

This chapter describes the CSV import and export tools in the following sections:

- "CSV Import Tool" below
- "CSV Export Tool" on page 127

CSV Import Tool

The CSV Import tool is an HP EM utility that imports data from a CSV file to the HP EM server.

- "Installation" below
- "Command Line" below
- "Header Parameter Syntax" on page 121
- "Data Content" on page 122
- "CSV File Creation" on page 125
- "Frequently Occurring Errors" on page 125
- "Useful Logging Settings" on page 127

Installation

EM_HOME is the variable which contains the absolute path to the HP EM Client installation.

The CSV Importer files must be located in the following directories:

File	Directory
csvimport.jar	EM_HOME\client\lib
log4j-csvimport.config	EM_HOME\ client\conf
csvimport.bat	EM_HOME\ client\bin
csvimport.sh	EM_HOME\ client\bin

Command Line

The import tool is located in the following folder:

`EM_HOME\client\bin`

Usage:

```
csvimport.bat/.sh [options]
```

- To import data from a csv file, follow this example:

```
csvimport.bat/.sh -user xxx -password yyy -file artifact.csv -sdmName
artifactName
```

- To import data directly from command line, follow this example:

```
csvimport.bat/.sh -user xxx -password yyy -sdmName artifactName -header
"header1,header2" -data "value1,value2"
```

Option	Description
-user [user]	**Description**: Username (required) **Default value**: Admin **Example**: -user admin
-password [password]	**Description**: Password (required) **Default value:** changeit **Example**: -password changeit
-host [host]	**Description**: HP EM URL (optional) **Default value**: Value obtained from the HP EM configuration file. **Example**: http://localhost:8888/em/
-file [path&filename]	**Description**: Path to the csv import file (required to import a csv file) **Default value**: none **Example**: - file c:\data\webServiceArtifact.csv
-sdmName [sdmName]	**Description**: SDM Name of the importer artifact (optional) **Default value**: Taken from the file name. **Example**: -sdmName webServiceArtifact
-header	**Description**: Header row when importing directly from the command line - When using this type of import. The sdmName must be specified.

Option	Description
	(required) • When this argument and specified data are not imported from the file but are imported from the command line. (optional) **Default value**: none **Example**: -header "name,description"
-dateFormat [yyyy-MM-dd HH:mm:ss]	**Description**: String representation of date in imported files **Default value**: yyyy-MM-dd HH:mm:ss **Example**: -dateFormat 2014-06-14 17:25:10
-data	**Description**: Data row when importing directly from the command line. (sdmName must be specified when using this type of import). Multiline text is supported through HTML code such as . (optional) **Default value**: none **Example**: -data "My name,My description"
- ignoreUnknown [Yes|No]	**Description**: By default (-ignoreUnknown Yes), CSV Importer validate the header of the CSV file (first line of the CSV file) - checks the existence of the all the columns (properties) in SDM. If the column (property) validation fails, the data in that column will be ignored (not imported). This functionality is operational at the value Yes. If the parameter is set to No (-ignoreUnknown No), CSV Importer tries to set each column without validation. This could cause the artifacts with invalid column names (properties) will be not imported. (optional) **Default value**: Yes **Example**: - ignoreUnknown No
-token	**Description**: A delimiter between multiple values of the same column **Default value**: | **Example**: -token #
-separator	**Description**:

Option	Description
	A delimiter between column headers or column values **Default value**: , **Example**: - separator ;
-mode [Insert|Update|Ignore]	**Description**: Artifacts will be modified based on the selected mode. (optional) • Insert - artifacts are always created (create duplicates if the imported artifacts already exist) • Update - artifacts are updated and if artifact does not exist then it is created • Ignore - only artifacts missing in repository are newly created. Existing ones are not updated. **Default value**: Update **Example**: -mode Insert
-dropRelations [Yes|No]	**Description**: Drop existing relations before adding new relations from CSV. (optional) **Default value**: No **Example**: - dropRelations Yes
-h	**Description**: Display help on tool usage. (optional) **Default value**: none **Example**: -h
-updateEmptyFields [Yes|No]	**Description**: Update property value to empty value. (optional) **Default value**: No **Example**: -updateEmptyFields Yes

Note: Make sure you add a quote for separator in Linux. An example is given below:

```
[root@sgatvm0047 bin]# ./csvimport.sh -user admin -password changeit@123 -data
"Business Service;Description" -header "name;description" -separator ";" -
sdmName businessServiceArtifact
```

Header Parameter Syntax

Primitive Property/Taxonomy

```
<property>|<key=true>
```

```
< >    - mandatory
```

```
[ ]     - optional
```

Header Element	Description
Property	sdm name of the property
Key	The CSV Import tool will use key properties as criteria to search for corresponding artifacts before processing to import/update. Any property can be used as a key, we can have more than one key. By default Name is used as a key property unless a key is defined in the header.

Note: System property (such as `_delete`) is read-only, you cannot import system properties to HP EM.

Relationship Property

```
<relationship>|[target=artifact_sdm_name]|<property=property_sdm_name>|
[attribute=attributeName]|[incoming=true|false]|[key=true|false]
```

```
< >    - mandatory
```

```
[ ]     - optional
```

Header Element	Description
Relationship	sdm name of the relationship, for example: composedOf
Target	target artifact type, for example: businessFunctionArtifact
Property	property of the target artifact, for example: name
Attribute	Relationship attribute, for example: usetype
Key	The CSV Import tool will use key properties as criteria to search for the target artifacts before processing to import/update. Any property can be used as a key, we can have more than one key. By default Name is used as a key property unless a key is defined in the header.

Reserved Names

Keyword	Description
owner	Sets the owner of the artifact, if empty – owner is the user starting the script or the existing one when there is no change.
lifecycleProcess	Name of the Lifecycle process.
lifecyleStage	Name of the Stage in the lifecycle process.
lifecycleStageApproved	True/False
attachment	Valid path according to the OS.

Common Relationship Attributes

Attribute	Description
cost	Cost transferred for selected relationship.
internalEffort	Internal effort value transferred for selected relationship.
externalEffort	External effort value transferred for selected relationship.
prerequisite	Prerequisite value transferred for selected relationship.

Example (plateauArtifact.csv)

```
name|key=true,description,aggregates|target=applicationComponentArtifact|property=name,aggregates|target=applicationComponentArtifact|property=name|attribute=cost,aggregates|target=applicationComponentArtifact|property=name|attribute=prerequisite
```

FinPlanner CSV Demo Plateau, Created by CSV import demo ,FinPlanner Application,15000, false

Data Content

Imported CSV must comply with the formatting rules. Their usage is described in the following chapter. The CSV file must have encoding UTF8 without BOM. Values shall be separated by ',' by default, but separator can be specified by 'separator' argument of the import tool.

The first row represents names of properties, for the header line syntax see "Header Parameter Syntax" on the previous page.

The second and next rows contain values of properties to set on creation. Format is as follows:

Property Type	Valid Input Format	Examples
Primitive Properties		
addressPropertyType	A set of recipient, city, state,	recipient=hp&stateProvince=hcmc&postalCode=70000&country=us

Property Type	Valid Input Format	Examples
Primitive Properties		
	province, postal code, country	
booleanPropertyType	1 = TRUE 0 = FALSE	
dailyIntervalPropertyType	A triple of day name, form and to	dayName=everyDay&from=01:15.00&to=01:30.00
dateTimePropertyType	yyyy-MM-dd HH:mm:ss tbd	2011-08-31 22:33:44
doublePropertyType	A double value	123.456
encryptedPasswordPropertyType	Any characters, user is responsible for encrypting it	
integerPropertyType	An integer value	123
nameUrlPairPropertyType	A pair of name and url	name=home&url=http://abc.123
nameUuidPairPropertyType	A pair of name and uuid	name=aaa&uuid=f6f4826f-7ec9-4067-b7c0-f70acebf82b7
nameValuePairPropertyType	A pair of name and value	name=abc&val=123
plainTextPropertyType	Plain text	

Property Type	Valid Input Format	Examples
Primitive Properties		
textareaPropertyType	Multiline text is supported through HTML code such as .	
Taxonomy Properties	Property Value as it is defined in the SDM. tModelKey is taken from property descriptor, but value has to be specified.	businessService
Relationship Properties	Depending on the relationship mapping definition – see the "Relationship Property" identifier for the relationship match, in "Header Parameter Syntax" on page 121.	

Cardinality specifics:

- Optional — single value in a valid format.
- Required — single value in a valid format.

- Multiple — multiple property values are separated with '|'. This can be changed by the 'token' argument of the import tool.

CSV File Creation

The CSV Importer accept only well-formatted CSV format in UTF-8 without BOM encoding. The following chapter describe one of the way how to do it from the MS Excel® format (xls).

Download and install the latest version of the LibreOffice. (http://www.libreoffice.org/download/).

To generate an appropriate CSV file:

1. Open the MS Excel® sheet in LibreOffice Calc and select the spreadsheet to be exported to CSV.
2. Select **File > Save As** to open the Save As dialog.
   a. Select the appropriate file name for the CSV file (usually the sdm_name of the imported artifact). Extension should be .csv.
   b. Save as type: choose Text CSV (.csv) (*.csv).
   c. Click **Save** and select **Use Text CSV** in the dialog. The CSV Export dialog opens.
3. Do the following in the CSV Export dialog:
   a. Enter the following field values:
      ○ **Charset**: Select Unicode (UTF-8)
      ○ **Field delimiter**: Input char , (or your separator)
      ○ **Text delimiter**: Input char "
   b. Click **OK**.

The CSV file is ready to import.

Frequently Occurring Errors

- **ERROR: impexp.GenericArtifactImporter - Error processing line '1': com.hp.utils.CSVReaderException: ERROR: Invalid Column Name descripton Line #1**
  The column name does not match the System Data Model (SDM). Property name is validated against the SDM.

  Hint: Check the name of the property.

- **ERROR: impexp.GenericArtifactImporter - Error processing line '1': com.hp.utils.CSVReaderException: ERROR: The CSV have incopatible format (line #1) has more data columns than heeader columns.**
  The number of the columns in the header and in the data row do not fit. Usually because of the separator character in the data.

  Hint: Check the content on the appropriate row.

- **ERROR: SDM Name 'xyzArtifact' not found in repository.**
  The artifact type xyzArtifact does not exist in System Data Model (SDM).

  Hint: Artifact name is extracted from file name (without extension) or value of parameter -sdmName . Please check if the value is correct.

- **ERROR com.hp.tools.importer.GenericArtifactImporter - Error while processing row: 2, Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Unable to set column xyz to value: v1|v2|v3 - error: Cannot import multiple values into single cardinality property for Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.**
  Cardinality of property xyz is single, cannot import multiple value to this property.

  Hint: Multiple value is declared by following format:

     `value1|value2|value3|…`

  Please review if you use this kind of value for single cardinality property.

- **ERROR com.hp.tools.importer.GenericArtifactImporter - Error while processing row: 2, Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Unable to set column xyz to value: - error: Cannot import empty value into required property for Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.**
  Cardinality of property xyz is required, cannot import an empty value to this property.

  Hint: check if you have used an empty value for this property in csv file.

- **ERROR com.hp.tools.importer.GenericArtifactImporter - Error while processing row: 2, Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Unable to set column xyz to value: abc - error: Cannot find target of required relation for Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.**
  Relationship property xyz is required. This error happens when the importer could not find any target artifact for relationship property xyz.

  Hint: check if your search criteria for property xyz in your csv is correct and relevant. These criteria must match at least one target artifact.

- **ERROR com.hp.tools.importer.GenericArtifactImporter - Error while processing row: 2, Artifact UUID: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Unable to set column xyz to value: abc - error: Taxonomy category name could not be resolved. Property type 'category'. (column:xyz - value:abc).**
  You have used a wrong category name for taxonomy property xyz.

  Hint: check if category name according to the taxonomy used by xyz property.

- **ERROR com.hp.tools.importer.GenericArtifactImporter - Error initializing repository client.**
  Server error, your EM server has been down or not running.

  Hint: check server status, access the self-test page to check if there is no problem.

- **ERROR com.hp.em.repository.remote.client.impl.RepositoryClientImpl - Server error: 401 – Unauthorized**
  Invalid user name or password.

  Hint: check if your credential is valid.

- **ERROR com.hp.tools.ImportTool - File parameter cannot be combined with header or data parametres**
  Importer only accepts data from csv file or from command line. Use of both modes is not supported.

- **ERROR com.hp.em.repository.remote.client.impl.RepositoryClientImpl - com.hp.em.repository.exceptions.RepositoryException: Value of required property 'xxx' in artifact 'businessServiceArtifact' is not set nor default value is not provided**
  Data in your CSV must be compliant to HP EM SDM. 'xxx' is a mandatory property, you must provide at least one value for this property.

Useful Logging Settings

Log4j configuration is stored in file `client/conf/log4j-csvimport.config`. You can modify it to suit your needs. Followings are some predefined settings:

- Log to a file in folder `csvimpoort/log`:

  log4j.rootLogger=INFO, Appender

  log4j.appender.Appender=org.apache.log4j.RollingFileAppender

  log4j.appender.Appender.File=csvimpoort/log/sample.log

  log4j.appender.Appender.layout=org.apache.log4j.PatternLayout

  log4j.appender.Appender.layout.ConversionPattern=%p: %c{2} - %m%n

- Log to a file in folder `csvimpoort/log` and to console

  log4j.rootLogger=INFO, Appender1,Appender2

  log4j.appender.Appender1=org.apache.log4j.ConsoleAppender

  log4j.appender.Appender2=org.apache.log4j.RollingFileAppender

  log4j.appender.Appender2.File= csvimpoort/log/sample.log

  log4j.appender.Appender1.layout=org.apache.log4j.PatternLayout

  log4j.appender.Appender1.layout.ConversionPattern=%p: %c{2} - %m%n

  log4j.appender.Appender2.layout=org.apache.log4j.PatternLayout

  log4j.appender.Appender2.layout.ConversionPattern=%p: %c{2} - %m%n

CSV Export Tool

The platform provides an endpoint that returns JSON data for a DQL query supplied. HTTP execution can be demonstrated with a help of UNIX's wget utility:

```
wget --user admin --password changeit --no-check-certificate -qO - --post-
data="dql=<query>select A.name from personArtifact A</query>"
http://localhost:8080/em/remote/query/
```

Comparing to a single execution of wget, the tool gets a complete result (all rows) and converts JSON data to a CSV format.

This endpoint is used by the Remote DQL command line tool (described below), which is provided with the CSV import distribution.

Remote DQL Command Line Tool

The remote DQL command line tool executes DQL queries against running the HP EM server and writes the result in a CSV format.

The following options are available:

```
-h, --help              Display this help
```

Remote Execution

The following three options are mandatory to execute DQL with a running HP EM:

`--url <baseurl> HP EM base URL, such as http://localhost:8080/em` —Use the same URL that is configured in HP EM.

```
--user <user>           user name

--password <password>   password
```

DQL Command

You can specify DQL using a non-option argument (with no option before the command) or with the --in option with a file containing DQL query.

```
--in <file>             plain text file with a DQL command
```

Output

The output is printed to console by default, unless you specify --out option. The out option can be specified multiple times, the n-th occurrence of the –out option with be used an output of the n-th --in option. CSV format is used.

`--out <file>` Output file to write the result

DQL Execution Parameters

You can also use DQL with parameters, for example: select name from artifactBase where name like :LIKE

All parameters must be bound before execution -- use either of the following options:

`--param LIKE=%a` Parameter LIKE is set to %A

`-PLIKE=%a` Parameter LIKE is set to %A

Examples

1. Execute a DQL with one parameter name LIKE, save the result to a file.

   ```
   remoteDql.bat --url http://localhost:8080/em --user admin --password
   changeit -PLIKE=a% "select count(1) as totalCount from artifactBase where
   name like :LIKE" --out countA.csv
   ```

2. Execute a SQL by wrapping it into DQL -- administrator rights are required.

```
remoteDql.bat --url http://localhost:8080/em --user admin --password
changeit "select name,sValue from (native(name,sValue){select name,sValue
from systemConfiguration where domain='topLevelDomain' and ownerName='<all>'
and sValue not like '<%' order by name}) N"
```

Chapter 8: WebDAV Compliant Publishing

HP EM uses a WebDAV compliant workspace to store data content uploaded to the repository using the publishing functionality.

HP EM supports WebDAV Level 1 (no locking). For details, see http://www.ietf.org/rfc/rfc4918.txt.

> **Caution:** WebDAV functionality is unavailable for HP EM integrated with CA Single Sign On because CA Single Sign On does not support the WebDAV protocol.

The WebDAV protocol enables document access in a file-system manner. You can access, create, modify, and delete documents using a WebDAV compliant client.

The publishing location is available at the following URL which varies depending on the authentication and transport security you use:

- Authenticated (username/password required)

  `http://SERVER:PORT/em/platform/restSecure/location`

  `https://SERVER:SSLPORT/em/platform/restSecure/location`

- Anonymous (username/password not required)

  `http://SERVER:PORT/em/platform/rest/location`

  `https://SERVER:SSLPORT/em/platform/rest/location`

> **Tip:** In Linux clients you may need to use `webdav` or `davs` as the protocol instead of `http(s)`.

HP recommends using the authenticated URL. HP EM permissions apply to operations performed in the publishing location using WebDAV.

You can use the URL in your WebDAV client, for example, in any of the following ways:

- As a publishing location in your IDE.

  For example, Eclipse or Visual Studio with appropriate WebDAV plugins, specifically, Plugin for Eclipse and Plugin for Visual Studio.

- As a mapped web folder in Windows.

> **Note:** Windows requires the KB907306 patch for the correct client functionality:
>
> http://www.microsoft.com/downloads/details.aspx?FamilyId=17C36612-632E-4C04-9382-987622ED1D64&displaylang=en

HP recommends deploying HP EM using standard HTTP/HTTPS ports (80/443) to ensure the correct client functionality.

In Windows Vista, a file from the publishing workspace opened in MS Office applications may appear as read-only. In this case, make a local copy and resubmit it to the server after you make your changes.

- Using a 3rd party file manager program with the appropriate plugin. For example, Total Commander with the plugin available at http://ghisler.fileburst.com/fsplugins/webdav.zip.

> **Note:** `Use multi-step upload method` must be disabled in Total Commander or any file is published as a documentation artifact. Restart Total Commander after changing any plugin settings.

Consult your WebDAV client documentation for details of their WebDAV functionality.

WebDAV access enables you to work with documents published to the repository using the publishing location like a file system (depending on the client). HP EM handles create and update operations using its publishing functionality, so relationships between documents are established and maintained with respect to the document content (for example, when a WSDL references an XSD, HP EM publishes the XSD and a relationship between them is established). These details are available in the HP EM UI in the document artifact details.

WebDAV publishing is an alternative to UI-based publishing. Unlike the configuration of UI publishing (for example, what artifacts to create), WebDAV publishing can only be configured globally using the configuration described in "Configuration Management" in the *Administration Guide*.

The most common WebDAV client operations are:

- Retrieving the content of published documents.

  For example, import a WSDL to your IDE client for service implementation development.

- Publishing new documents.

  For example, publish a WSDL to the repository from your IDE client. HP EM uses its publishing feature to create the document and associated artifacts. Relationships are automatically maintained.

- Republishing documents.

  For example, importing a WSDL to your IDE client, modifying it, and then republishing. HP EM uses its publishing functionality to update the document and maintain associated artifacts and relationships.

- Deleting documents.

  For example, using your IDE client to delete an obsolete WSDL. HP EM uses Delete instead of Purge enabling retrieval of the document if required.

- Changing document locations.

  WebDAV clients can use the MOVE operation to change the server location for an artifact in the repository. HP EM maintains metadata and history. This functionality enables remote management of the publishing location.

- Creating, renaming, and deleting directories.

  The publishing location is effectively a file system, enabling you to organize your documents in the publishing location using your WebDAV client.

- Copying documents or whole directories.

  Create duplicates of publishing folders or documents in the publishing location.

Chapter 9: HP EM Extension for Inkscape

HP EM integrates with the open source Inkscape vector graphics editing tool via an extension module. By using the HP EM Extension for Inkscape, you can easily edit SVG graphic files and other graphical elements residing in the HP EM data models.

HP EM Extension for Inkscape is described in the following sections:

- "Installing the HP EM Extension for Inkscape " below
- "Using the HP EM Extension for Inkscape" below
- "Applying a New SVG File to Your EM Home Page" on page 134
- "Using Log Files for HP EM Extension for Inkscape" on page 135

Installing the HP EM Extension for Inkscape

Before you install the HP EM Extension for Inkscape, make sure that you have the following on your system:

- Inkscape 0.48.5-1 32-bit for Windows (You can download Inkscape from: www.inkscape.org.)
- JDK/JRE 7 (32-bit) or later

To install the HP EM Extension for Inkscape:

1. Execute hp-em-inkscape-3.00.msi

   > **Note:** Inkscape MUST be installed before launching the installer.

2. After installation is finished, the following files must be present:

File Name	Directory
addArtifact.inx	INKSCAPE_HOME\share\extensions\
addLayer.inx	INKSCAPE_HOME\share\extensions\
publishToEAM.inx	INKSCAPE_HOME\share\extensions\
eamanager.py	INKSCAPE_HOME\share\extensions\
eamanager(folder)	INKSCAPE_HOME\share\extensions\

3. Restart Inkscape. HP Enterprise Maps appears in Extensions menu bar.

Using the HP EM Extension for Inkscape

By using the HP EM Extension for Inkscape, you can add artifacts or layers or publish your graphics directly to EM.

To add an artifact:

1. Click **Extensions > Enterprise Maps > Add Artifact**.

2. Input the exact values for the Label and the Artifact Local Name in the **Add Artifact** fields.

3. Click **Apply** to create the new artifact. Alternatively, you can click **Close** to exit without creating a new artifact.

To add a layer:

1. Click **Extensions > Enterprise Maps > Add Layer**.

2. Input the exact values for the Label and the Artifact Local Name in the Add Layer fields.

3. Click **Apply** to create the new layer. Alternatively, you can click Close to exit without creating a new layer.
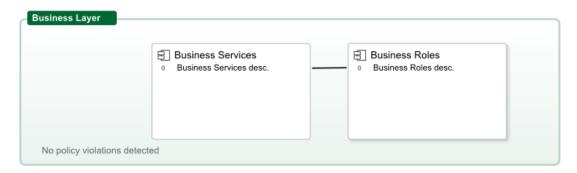
To publish to HP EM:

1. Click **Extensions > Enterprise Maps > Publish to EM**.

2. Input the exact values for the Viewpoint Name, HP EM URL, Username, and Password fields.
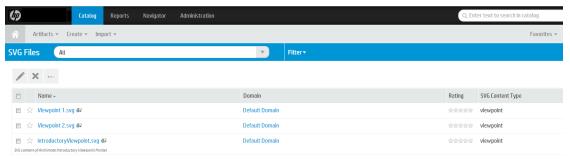
   > **WARNING**: HP EM password is stored by Inkscape in plain text. Open this dialog and remove the password before exiting Inkscape.

3. Click **Apply** to publish the new file to HP EM. Alternatively, you can click **Close** to exit without publishing.

4. You can publish a single SVG file multiple times. Based on the file name, HP EM will do one of the following actions:

   a. Publish the new file if there is no SVG artifact having the same name already, or

   b. Update the existing file if there is an artifact with that name.

5. After you publish, you can view the SVG artifacts in the HP EM Catalog.

   The new artifact and layer will appear in HP EM.

You can also view artifacts in the Catalog.

Applying a New SVG File to Your EM Home Page

You can apply any SVG file you choose to appear on your HP EM home page.

To apply a new SVG file to Your HP EM home page:

1. Log in to HP EM.
2. Go to the Catalog showing the list of SVG files and open the artifact that you want to use.
3. Copy its UUID to the clipboard.
4. Go to the **Administration** tab **> Customization > Manage Scripts**, and then open the Archimate Introductory Viewpoint.
5. Click **Edit Script**, and then paste the UUID into the config.viewPointuuid value. When you finish, save the script.

6. Go to the **Catalog** tab and right click on your home page dashboard. The context menu will be displayed.
7. Click **Archimate Introductory Viewpoint** and the new file will display.

Using Log Files for HP EM Extension for Inkscape

You can use log files to track or investigate any errors that occur. The log files are located at:

`INKSCAPE_HOME\share\extension\em\log`

There are two log files to use:

- The file `inkscape.log` contains the trace output for adding artifacts and layers.
- The file `publish.log` contains trace output for publishing of SVG files to HP EM.

Chapter 10: Atom-Based REST Interface

HP EM uses an ATOM-based REST interface.

Access the HP EM platform service document using the following URL:

`http://hostname:port/context/platform/rest`

Hostname, port, and context are set during installation. For example, if you used the default settings and installed to your local machine, use the following URL:

`http://localhost:8080/em/platform/rest`

If set up during installation, an HTTPS secure endpoint is available which requires credentials to access.

A default secure endpoint uses the following URL:

`https://localhost:8443/em/platform/rest`

> **Note:** Use `restSecure` instead of `rest` if you are using HTTP basic authentication.

The service document consists of workspaces, which in turn contains feeds made up of entries, as shown in the following example:

Platform Service Document

```
<?xml version="1.0" encoding="UTF-8"?>
<app:service xml:base="http://localhost:8080/em/platform/rest/"
    xmlns:app="http://www.w3.org/2007/app">
 <app:workspace>
  <atom:title type="text"
      xmlns:atom="http://www.w3.org/2005/Atom">SDM collections</atom:title>
  <app:collection href="./artifact/reportArtifact">
   <app:accept/>
   <atom:title type="text"
       xmlns:atom="http://www.w3.org/2005/Atom">Collection of Reports</atom:title>
   <app:categories href="./category-document/
       uddi:systinet.com:systinet:model:taxonomies:artifactTypes:_artifactType"/>
   <app:categories href="./category-document/
       uddi:systinet.com:systinet:model:taxonomies:reportTypes:reportType"/>
   <app:categories href="./category-document/

uddi:systinet.com:systinet:model:taxonomies:reportCategories:reportCategory"/>
   <app:categories href="./category-document/
       uddi:systinet.com:systinet:model:taxonomies:reportStatus:reportStatus"/>
   <app:categories href="./category-document/

uddi:systinet.com:systinet:model:taxonomies:reportResultCodes:reportResultCode"/>
  </app:collection>
```

```
 ...
</app:workspace>
<app:workspace>
 <atom:title type="text"
     xmlns:atom="http://www.w3.org/2005/Atom">Publishing Locations</atom:title>
 <app:collection href="./location">
  <app:accept/>
 </app:collection>
</app:workspace>
<app:workspace>
 <atom:title type="text"
     xmlns:atom="http://www.w3.org/2005/Atom">System Information</atom:title>
 <app:collection href="./system">
  <app:accept/>
 </app:collection>
</app:workspace>
</app:service>
```

The interface is described in the following sections:

- "Workspaces" below

- "Feeds" on the next page

- "Entries" on page 146

- "Category Documents" on page 154

- "Atom REST Operations" on page 155

- "Atom REST ETags" on page 156

- "Atom REST Client" on page 158

Workspaces

The platform service document consists of the following workspaces:

- "SDM Collections Workspace" below

  The System Data Model (SDM) workspace reflects the structure of the SDM and defines feeds for the collections in the HP EM repository (read-only).

- "Publishing Locations Workspace" on the next page

  The locations workspace reflects the structure of attached data content in HP EM created by the publisher.

- "System Collections Workspace" on the next page

  The system workspace contains system information used by HP EM (read-only).

SDM Collections Workspace

The SDM collections workspace contains a collection for each artifact type in the SDM for which an instance can be created within its artifact hierarchy.

> **Note:** Customization Editor can be used to modify the SDM, so your configuration may vary from specific examples in this documentation. For details, see the *HP Enterprise Maps Workbench - Customization Editor Guide.*

Each collection in the workspace consists of the following:

- <app:collection href="./artifact/*artifactType*">

  The reference defines the URL used for the feed for that particular artifact type collection. For details, see "Artifact Collection Feeds" below.

- <app:categories href="./category-documents/*taxonomy*">

  Categories can occur in feed entries and some feed readers can perform filtering according to these categories.

Publishing Locations Workspace

The publishing locations workspace consists of a single collection. This collection is an atom feed made up of entries where the entry can be one of the following types:

- Subcollection
- Resource

The subcollections and resources reflect content uploaded to HP EM using its publication feature.

This location is available as a feed and is accessible with a WebDAV client.

For details, see "Publishing Location Feeds" and "WebDAV Compliant Publishing" on page 130.

System Collections Workspace

The system collections workspace contains a single collection. This collection contains information about the running system.

Feeds

You can access the content of the repository using feeds.

- "Artifact Collection Feeds" below
- "Publishing Location Feeds" on page 143
- "Artifact Relationships Feed" on page 145
- "Artifact History Feed" on page 145
- "Artifact Comments Feed" on page 145
- "Full Text Search" on page 145

Artifact Collection Feeds

Every artifact type collection in the SDM is accessible as a feed.

Use the reference defined in the SDM collections workspace to access a collection feed.

For example, the WSDL collection feed is accessed with URL:

`http://localhost:port/context/platform/rest/artifact/wsdlArtifact`

WSDL Collection Feed

```
<feed xml:base="http://localhost:8180/platform/rest/artifact/wsdlArtifact"
    xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
    xmlns="http://www.w3.org/2005/Atom">
  <id>urn:hp.com:2009:02:systinet:platform:artifacts:sdm:wsdlArtifact</id>
  <updated>2009-06-19T14:54:11.614+02:00</updated>
  <title type="text" xml:lang="en">Collection of WSDLs</title>
  <opensearch:itemsPerPage>50</opensearch:itemsPerPage>
  <opensearch:startIndex>1</opensearch:startIndex>
  <link href="artifactBase" type="application/atom+xml;type=feed"
      rel="urn:hp.com:2009:02:systinet:platform:artifacts:parent"
      title="parent sdm feed"/>
  <link href="wsdlArtifact?start-index=1&amp;page-size=50"
      type="application/atom+xml;type=feed"
      rel="self" title="feed self"/>
  <author>
    <name>system:restadmin</name>
  </author>
  <generator>HP Enterprise Maps</generator>
  <entry>
    <id>urn:hp.com:2009:02:systinet:platform:artifact:4465c1e1-f214-47c5-a958-
d3202ab20dfa</id>
    <updated>2009-06-09T10:06:35.443+02:00</updated>
    <title type="text" xml:lang="en">paymentMethod.wsdl</title>
    ...
  </entry>
  ...
</feed>
```

Each artifact type collection feed consists of the following descriptors:

Descriptors	Description
id	The feed identification.
updated	The last update time.
title	The name of the feed.

link	A set of links with the following link types indicated by the `rel` attribute:
	
	• `urn:hp.com:2009:02:systinet:platform:artifacts:parent`
	  Links to collection feeds for super artifacts in the inheritance category.
	
	• `urn:hp.com:2009:02:systinet:platform:artifacts:child`
	  Links to collection feeds for descendant artifact types.

Descriptors	Description
entry	The set of entries in the feed. For more details, see "Artifact Atom Entries" on page 146.
opensearch:startIndex	Starting point for the feed relative to index entries. The first indexed item is 1.
opensearch:itemsPerPage	Number of items per page.

You can modify the output of the feed as described in the following sections:

- "Filtering Feeds" below
- "Viewing Entry Content in Feeds" on the next page
- "Domains in Feeds" on the next page
- "Property Based Searching" on the next page
- "Feed Ordering" on page 142
- "Feed Paging" on page 142
- "Bulk GETs" on page 143

You can also combine these output methods.

Separate each term with "&".

For example, to get artifacts 10-79 which contain `policy` in the description, ordered primarily by their name in descending order and then by description in ascending order, and displaying properties defined in `artifactBase`, use the following URL:

```
http://host:port/context/platform/rest/artifact/artifactBase?p.description=*policy*
&start-index=10&page-size=70&order-by=name-,description&inline-content
```

Filtering Feeds

Feeds are presented in the REST interface as a set of equivalent collections.

Examples of feeds include:

- `http://localhost:port/context/platform/rest/artifact/implementationArtifact`
- `http://localhost:port/context/platform/rest/artifact/xmlServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/webServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/businessServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/wsdlArtifact`

Viewed in this way, the feeds form a flat structure. However, there are established relationships between feeds in terms of an inheritance hierarchy.

The root of the hierarchy is
`http://localhost:port/context/platform/rest/artifact/artifactBase`.

You can use abstract artifact type feeds to obtain all artifact types lower in the hierarchy. For example, the implementationArtifact feed contains all SOAP service, XML service, and web application artifacts.

The relationships between feeds are realized via
`urn:hp.com:2009:02:systinet:platform:artifacts:parent` and
`urn:hp.com:2009:02:systinet:platform:artifacts:child` links.

Viewing Entry Content in Feeds

You can use feeds to obtain multiple artifact entry content as well.

Add `?inline-content` to the collection feed URL to obtain the full content for each entry in the feed.

> **Note:** The properties displayed in the content for an entry are determined by the artifact type used in the feed URL. Properties specific to an artifact type lower in the hierarchy are not displayed.

Domains in Feeds

The domain can be specified using a domain parameter in the `/artifact/` segment or the feed URL.

For example,
`http://localhost:port/context/platform/rest/artifact;domain=defaultDomain/wsdlArtifact` shows all WSDLs in the Default Domain.

> **Note:** Artifacts may be moved across domains using a PUT operation that specifies the system property `_domainId`.

Property Based Searching

You can search for specific artifacts in a feed with property based filtering. You can filter by any property type regardless of its type and cardinality, but the elementary conditions are always primitive values. The filtering property must be present in the artifact type defining the feed.

The property must be one of the following elementary types:

- text
- integer
- bigInteger
- date
- double
- boolean
- uuid

To view the permitted property names for a particular artifact feed, you can examine the SDM with URL:

`http://host:port/context/platform/rest/system/model`.

If you want to filter by a compound property (for example, category property which has 3 compounds: taxonomyUri, name, value) you must use dot notation. For example to search by compound `val` (value) of property `criticality` on `businessServiceArtifact` use the following URL:

`http://host:port/em/platform/rest/artifact/businessServiceArtifact?p.criticality.val=uddi:systinet.com:soa:model:taxonomies:impactLevel:high`

Only business services artifacts with high criticality are listed.

For text property filtering, operator case-insensitive-equals is used, but can explicitly use wildcards. To find all service artifact with `svc` in their name submit the following URL:

`http://host:port/em/platform/rest/artifact/businessServiceArtifact?p.name=*svc*`

The following wildcards are supported:

- `*` for zero or more arbitrary characters.
- `_` for exactly one arbitrary character.

> **Note:** HP EM does not support explicit boolean operators but there is an implicit AND for conditions on different properties and an implicit OR on conditions on the same property.

The following examples show various ways to use property searching:

- Artifacts with a name starting with `service` and a description containing `assertion`:

  `http://host:port/context/platform/rest/artifact/artifactBase?p.name=service*&p.description=*assertion*`

- Artifacts with a name containing either starting with `service` or containing `assertion`:

  `http://host:port/context/platform/rest/artifact/artifactBase?p.name=service*&p.name=*assertion*`

- Deleted artifacts only.

  `http://host:port/context/platform/rest/artifact/artifactBase?p._deleted=true`

> **Tip:** To view the category values, open the category document, for details, see "Category Documents" on page 154.

Feed Ordering

By default, entries in feeds are ordered by their `atom:updated` element.

Add `?order-by=` to the collection feed URL to change the order.

- Entries ordered by name (ascending):

  `http://host:port/context/platform/rest/artifact/artifactBase?order-by=name`

- Entries ordered by name (descending):

  `http://host:port/context/platform/rest/artifact/artifactBase?order-by=name-`

- Entries ordered by name (descending), then description (ascending):

  `http://host:port/context/platform/rest/artifact/artifactBase?order-by=name-,description`

You can also use properties for ordering with the same conditions as for searching.

For details, see "Property Based Searching" on the previous page.

Feed Paging

You can also control the feed paging.

- The first ten entries:

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?page-size=10
  ```

- Entries 10-19 (inclusive):

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?page-size=10&start-
  index=10
  ```

Note: The default number of entries is 50 and the maximum number of entries is 500.

Bulk GETs

A specific REST use case is a Bulk GET - getting multiple artifacts in a single request/response interaction. This can be handled via a property based search on specific collections, presuming that the UUIDs of the artifacts to retrieve are known.

For example, assume the following business service artifacts with UUIDs, bs1 and bs2. There are 3 web service artifacts with UUIDs ws1, ws2, and ws3. The ATOM GET request to return all 5 artifacts at once is as follows:

```
http://host:8080/em/platform/rest/artifact/artifactBase?p._uuid=bs1&p._uuid=bs2&p._
uuid=ws1&p._uuid=ws2&p._uuid=ws3&inline-content
```

Notice the inline-content flag, it specifies the inclusion of proprietary XML representation into atom entries.

Submitting this URL returns a feed with 5 artifacts, assuming they exist. But inside the atom content there are only properties specific to the artifactBase artifact type. For example: businessServiceArtifact defines the property `criticality`. This property is not present in the atom content because it is not declared at artifactBase level. The properties listed in the atom content are strictly driven by artifact type, specified as one part of the URL (in our case artifactBase).

However, there is one exception, relationship properties are always listed in the atom content regardless of the given artifact type. The business service artifact defines a relationship property `service`. This property is not declared at artifactBase level, however, it is present in the XML representation regardless of the artifact type given in the URI.

If you want to get the full set of properties (even those specific to the given artifact type), you must perform multiple GETs per artifact type. In our example, this requires the following 2 GETs:

```
http://host:8080/em/platform/rest/artifact/businessServiceArtifact?p._uuid=bs1&p._
uuid=bs2&inline-content
```

```
http://host:8080/em/platform/rest/artifact/webServiceArtifact?p._uuid=ws1&p._
uuid=ws2&p._uuid=ws3&inline-content
```

By submitting these two HTTP GETs, you obtain full representation of the 5 artifacts: bs1, bs2, ws1, ws2, and ws3.

Publishing Location Feeds

The location feed enables you to browse the attached data content in the repository.

HP EM adds this content whenever you publish an artifact associated with attached data content. .

The publishing location is accessible using a WebDAV client. For details, see "WebDAV Compliant Publishing" on page 130.

The content feed consists of resources (the data content) organized into collections (folders). Access the feed using the following URL:

```
http://localhost:8080/em/platform/rest/location
```

If you use a browser, this opens a view which enables you to browse the data content and interact with it.

> **Note:** The view of a collection location only displays the resources for which you have permissions.

HP EM publisher creates a collection within the publishing location when you upload data content. .

Open a collection by clicking its name, or download a zip file of its content by clicking **Download as Archive**. At the lowest level, the browser shows the actual data content. For the actual content, click the content name.

Click **Advanced View** to open the detail view of the related artifact in HP EM. For details, see "Artifact View Page" in the *User Guide*.

You can change the output of the location space on your browser using alternative media types:

- `http://hostname:port/context/platform/rest/location`

  The default output as described above.

- `http://hostname:port/context/platform/rest/location?alt=text/html`

  The HTML representation which is the default output for locations. For artifacts with non-HTML content there is no HTML representation.

- `http://hostname:port/context/platform/rest/location/foo?alt=application/zip`

  Output all files from a particular collection (foo) to a zip archive.

  Add the following optional switches to output additional related documentation:

  - `&inline-desc`

    Includes document descriptor files in the archive (files with the .desc suffix in .meta subdirectories).

  - `&inline-acl`

    Includes ACL files in the archive (files with the .acl suffix in .meta directories).

  - `&zip-compat`

    Enable zip compatibility mode (no directory entries are created in the archive).

- `http://hostname:port/context/platform/rest/location/test?alt=application/atom%2bxml`

  View the Atom feed for a collection location.

- `http://hostname:port/context/platform/rest/location/foo?alt=application/json`

  Output a particular collection location as a JSon representation.

By default, the last revision of a resource or collection is shown, but you can request revisions from a particular date using the following pattern:

```
http://hostname:port/context/platform/rest/location;datetime=[datetimeValue]
```

For example, `http://hostname:port/context/platform/rest/location/foo/a.wsdl`, corresponds to the last revision of a the `a.wsdl` resource in the `foo` location.

`http://hostname:port/context/platform/rest/location;datetime=2008-01-01T12:00:00.000Z/foo/a.wsdl`, corresponds to the revision of the `a.wsdl` resource at 12:00 on 1/1/2008.

Specifying a collection location that does not exist returns an exception.

Artifact Relationships Feed

You can view the relationships of an artifact as a feed.

For example, to view the comments feed of a WSDL artifact, use the URL:

```
http://host:port/context/platform/rest/artifact/wsdlArtifact/UUID/relation
```

The feed returns both incoming and outgoing relationships to/from the artifact. The content shows a proprietary representation of the relationship, with the related artifact available by following the 'alternate' link.

If the related artifact is readable by the current client identity, its name is displayed, otherwise only its UUID is shown.

Artifact History Feed

You can view the revision history of an artifact as a feed.

For example, to view the revision history of `my.wsdl`, use the URL:

```
http://host:port/context/platform/rest/artifact/wsdlArtifact/my.wsdl/history
```

Artifact Comments Feed

You can view the comments made about an artifact as a feed.

For example, to view the comments feed of a WSDL artifact, use the URL:

```
http://host:port/context/platform/rest/artifact/wsdlArtifact/UUID/comments
```

Full Text Search

Full text search can be run in an SDM collection feed.

Add `?fulltext=SEACHEDTEXT` to the collection feed URL to perform full text search.

For example, to search for the text "lifecycle" in all artifacts:

```
http://host:port/context/platform/rest/artifact/artifactBase?fulltext=lifecycle
```

`Feed Ordering` and `Feed Paging` can be also applied to the result.

Full text search result can be only ordered by `relevance`, `name` or `timestamp`.

Default ordering is `relevance-,name`.

Note: Full text search must be enabled in the database, for more details see the Installation Guide.

Entries

The detailed information about an artifact in the repository is available as an entry.

Entries are described in the following sections:

- "Artifact Atom Entries" below
- "Artifact History Entries" on page 148
- "Atom Entry Property Descriptors" on page 149
- "Artifact Data" on page 153
- "Resource Identification" on page 153

Artifact Atom Entries

The information about each entry in the collection feed is only a summary. Each entry can be accessed directly using its `self link` as referenced in the artifact feed, which is formed from either its `restName` or `id`.

For example, you can access a particular user profile entry with URL:

`http://localhost:port/context/platform/rest/artifact/personArtifact/admin`

Admin User Profile Entry

```
<entry xml:base=
    "http://localhost:8180/em/platform/restSecure/artifact/personArtifact"
    xmlns="http://www.w3.org/2005/Atom">
  <id>urn:hp.com:2009:02:systinet:platform:artifact:d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a</id>
  <updated>2009-06-01T09:30:23.154+02:00</updated>
  <title type="text" xml:lang="en">HP EM Administrator</title>
  <summary type="text" xml:lang="en">HP EM Administrator.</summary>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fatom%2Bxml"
      type="application/atom+xml" rel="self" title="artifact detail"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fxml"
      type="application/xml" rel="alternate" title="XML representation"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fatom%2Bxml"
      type="application/atom+xml"
      rel="urn:hp.com:2009:02:systinet:platform:artifact:last-revision"
      title="last revision"/>
```

```
<link href="personArtifact" type="application/atom+xml;type=feed"
    rel="urn:hp.com:2009:02:systinet:platform:artifacts:collection"
    title="sdm feed"/>
<link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a/history"
    type="application/atom+xml;type=feed"
    rel="urn:hp.com:2009:02:systinet:platform:artifact:history"
    title="history feed"/>
<link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a/acl"
    type="application/xml"
    rel="urn:hp.com:2009:02:systinet:platform:artifact:acl"
    title="access control list"/>
<link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a?alt=text%2Fhtml"
    type="text/html" rel="alternate" title="UI view page"/>
<author>
  <name>em:admin</name>
</author>
<category label="Active"
    scheme="uddi:systinet.com:soa:model:taxonomies:accountStates:accountState"
    term="S1"/>
<category label="Artifact"
    scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
    term="urn:com:systinet:soa:model:artifacts"/>
<category label="Content"
    scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
    term="urn:com:systinet:soa:model:artifacts:content"
    ext:parent="urn:com:systinet:soa:model:artifacts"
    xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
<category label="Contact"
    scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
    term="urn:com:systinet:soa:model:artifacts:content:contact"
    ext:parent="urn:com:systinet:soa:model:artifacts:content"
    xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
<category label="User Profile"
    scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
    term="urn:com:systinet:soa:model:artifacts:content:contact:person"
    ext:parent="urn:com:systinet:soa:model:artifacts:content:contact"
    xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
<content type="application/xml">
  ...
</content>
</entry>
```

Each artifact entry consists of the following descriptors:

Descriptor	Description
id	A unique id for the artifact (uuid).
updated	The last update time.

Descriptor	Description
`title`	The name of the entry.

`link`	A set of links with the following link types indicated by the `rel` attribute:
	
	• `self`
	  The atom entry details.
	
	• `urn:hp.com:2009:02:systinet:platform:artifacts:collection`
	  The associated artifact collection feed. For details, see "Artifact Collection Feeds" on page 138.
	
	• `urn:hp.com:2009:02:systinet:platform:artifact:last-revision`
	  The last revision of this artifact.
	
	• `edit-media`
	  The associated data content for an artifact.
	
	• `urn:hp.com:2009:02:systinet:platform:artifact:history`
	  The collection feed for revisions of this artifact.
	
	• `alternate`
	  A set of alternate views of the artifact, including:
	
	  • `application/xml` The bare XML representation of the `content` descriptor.
	
	  • `text/html` Points to the HP EM UI view of the artifact.
	
	• `related`
	  Links to related artifacts.
	
	  Note: Related artifacts may also be linked where the link has the `rel` attribute with a specific relationship name. For details, see "Relationship Properties Atom Representation" on page 151.

`category`	A set of taxonomic values from: • Taxonomy property values • categoryBag and identifierBag • sdmTypes taxonomy values
`author`	The creator of this revision of the artifact.
`content`	The bare XML representation of the content descriptor. For details, see "Atom Entry Property Descriptors" on the next page.
`summary`	An artifact description.

Artifact History Entries

By default, entries display the latest revision. You can view older revisions by adding `;rev=X` to the entry URL.

For example, the first revision of a WSDL can be obtained with the URL:

```
https://host:port/context/platform/rest/artifact/wsdlArtifacts/mywsdl;rev=1
```

Atom Entry Property Descriptors

Atom entries contains an XML representation of an artifact in the `content` descriptor.

Admin User Entry Content

```xml
<content type="application/xml">
  <art:artifact name="personArtifact" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:p="http://hp.com/2008/02/em/platform/model/property"
      xmlns:sdm="http://hp.com/2007/10/em/platform/model/propertyType"
      xmlns:art="http://hp.com/2008/02/em/platform/model/artifact">
    <p:primaryGroup xsi:nil="true" sdm:type="text"/>
    <p:accountState name="Active"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:accountStates"
value="S1"
        sdm:type="category"/>
    <p:designTimePolicy xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:documentation xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:_uuid sdm:type="uuid">d82a5dcc-d85c-4766-9967-93eb5dc0bd0a</p:_uuid>
    <p:_revision sdm:type="integer">1</p:_revision>
    <p:_checksum sdm:type="bigInteger">0</p:_checksum>
    <p:_contentType xsi:nil="true" sdm:type="text"/>
    <p:_revisionTimestamp sdm:type="date">2009-06-01T07:30:23.154Z</p:_
revisionTimestamp>
    <p:keyword xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:categoryBag xsi:nil="true" sdm:type="categoryBag"/>
    <p:_revisionCreator sdm:type="text">em:admin</p:_revisionCreator>
    <p:_artifactType name="Artifact"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts" sdm:type="category"
p:multi="true"/>
    <p:_artifactType name="Content"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content" sdm:type="category"
p:multi="true"/>
    <p:_artifactType name="Contact"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content:contact"
sdm:type="category"
        p:multi="true"/>
    <p:_artifactType name="User Profile"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content:contact:person"
```

```
sdm:type="category"
        p:multi="true"/>
    <p:identifierBag xsi:nil="true" sdm:type="identifierBag"/>
    <p:description sdm:type="text">HP EM Administrator.</p:description>
    <p:_owner sdm:type="text">admin</p:_owner>
    <p:_deleted sdm:type="boolean">false</p:_deleted>
    <p:name sdm:type="text">HP SOA Administrator</p:name>
    <p:consumptionContract xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:consumptionRequest xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:r_consumerOwner2contract xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:provides xsi:nil="true" sdm:type="documentRelationship" p:multi="true"/>
    <p:contactRole xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:r_contactClassification xsi:nil="true" sdm:type="category"/>
    <p:geographicalLocation xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:languageCode xsi:nil="true" sdm:type="category"/>
    <p:hpsoaApplicationContact xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:r_memberOf xsi:nil="true" sdm:type="documentRelationship" p:multi="true"/>
    <p:loginName sdm:type="text">admin</p:loginName>
    <p:address xsi:nil="true" sdm:type="address"/>
    <p:email sdm:type="text" p:multi="true">admin@comp.com</p:email>
    <p:phone xsi:nil="true" sdm:type="text" p:multi="true"/>
    <p:instantMessenger xsi:nil="true" sdm:type="text" p:multi="true"/>
    <p:externalDefinition xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
  </art:artifact>
</content>
```

The content is effectively a list of the properties of an artifact.

The property types are described in the following sections:

- "Primitive Properties Atom Representation" below
- "Category Properties Atom Representation" on the next page
- "Relationship Properties Atom Representation" on the next page
- "Special Properties Atom Representation" on page 152

Primitive Properties Atom Representation

Primitive properties are represented as follows:

`<p:NAMEsdm:type="TYPE">VALUE<p:NAME>`

The following primitive property types use this form:

Property Type	xsi:type Correspondance
date	xs:dateTime
boolean	xs:boolean
double	xs:double
integer	xs:int
bigInteger	xs:integer
text	xs:string
uuid	xs:string

For example:

```
<p:phone sdm:type="text">774 789 784</p:phone>
```

Category Properties Atom Representation

Category properties are propagated in two places in the Atom entries.

The `category` descriptor, which also appears in collection feeds, describes the taxonomy and category as follows:

```
<category label="..." scheme="..." term="..."/>
```

- `label` corresponds to the category name.
- `scheme` corresponds to the taxonomy URI combined with the property name.
- `term` corresponds to the category URI.

This is reproduced in the entry `content` as a property:

```
<p:NAME name="..." taxonomyUri="..." value="..." sdm:type="category"/>
```

For example, a web service with Failure Impact set to High is represented as a property in the entry for the web service:

```
<p:criticality name="High"
taxonomyUri="uddi:systinet.com:soa:model:taxonomies:impactLevel"
value="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"
sdm:type="category"/>
```

Note that the property representing this taxonomic category is `criticality`.

The property is propagated to Atom metadata as an `atom:category` element:

```
<atom:category label="High"
scheme="uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality"
term="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"/ >
```

Relationship Properties Atom Representation

Relationship properties are propagated in two places in the Atom entry.

In feeds the `link` exists as metadata.

The `link` descriptor describes the following link types:

- A generic `related` link.
- A specific relationship bound link where the `rel` attribute uses a
  `'urn:hp.com:2009:02:systinet:platform:artifact:relation:`prefix with the relationship
  name.

In entries, relationships are described as a set of `property` atom content descriptors:

Relationship Properties

```
Incoming relationship example:
<p:inBusinessService xlink:href="businessServiceArtifact/1210"
    sdm:type="documentRelationship" p:multi="true">
  <t:source>c519d961-03b3-4303-b61b-8809b945b7ae</t:source>
  <t:exact>false</t:exact>
</p:inBusinessService>

Exact incoming:
<p:inBusinessService xlin:href="businessServiceArtifact/1210"
    sdm:type="documentRelationship" p:multi="true">
  <t:source>c519d961-03b3-4303-b61b-8809b945b7ae</t:source>
  <t:exact>true</t:exact>
</p:inBusinessService>

Outgoing relationship example:
<p:service xlin:href="webServiceArtifact/5"
    sdm:type="documentRelationship" p:multi="true">
  <t:target deleted="false">5a4aeca7-a8f9-4761-b504-82723ab2f417</t:target>
</p:service>

Exact outgoing:
<p:service xlin:href="xmlServiceArtifact/101.xml;rev=1"
    sdm:type="documentRelationship" p:multi="true">
  <t:target revision="1" deleted="false">72ab6f1f-e943-4fd2-a7bc-
5d227e6e134a</t:target>
</p:service>
```

Special Properties Atom Representation

Special properties are defined by an XML schema which determines their structure.

HP EM contains an XML schema which defines the following property types:

- address
- categoryBag
- identifierBag
- dailyInterval
- nameURLPair

- nameValuePair
- parameterList (XQuery parameter)
- scheduled
- selector

Artifact Data

If an artifact has associated data content, then you can directly access the data content.

For example, a WSDL artifact is usually associated with the actual WSDL file.

Access the WSDL entry with the URL:

`https://localhost:8443/em/platform/rest/artifact/wsdlArtifact/mywsdl?alt=atom`

WSDL Entry

```
<entry
xml:base="http://localhost:8180/em/platform/restSecure/artifact/wsdlArtifact"
    xmlns="http://www.w3.org/2005/Atom">
  <id>urn:hp.com:2009:02:systinet:platform:artifact:f5aff3eb-95fd-4791-856b-
3ac551666da2</id>
  <updated>2009-06-08T16:24:55.609+02:00</updated>
  <title type="text" xml:lang="en">mywsdl</title>
  ...
  <link href="../location/wsdls/mywsdl.wsdl" type="application/xml" rel="edit-
media" title="attached data" />
  ...
</entry>
```

The entry contains a link pointing to the locations workspace. The data is also available using a `/data` suffix.

For example,
`https://localhost:8443/em/platform/rest/artifact/wsdlArtifact/mywsdl/data`

You can also access older revisions of the data with the URL:

`https://localhost:8443/em/platform/rest/artifact/wsdlArtifact/mywsdl;rev=1/data`

> **Caution:** Using any relative references in the XML data will probably cause an error because they are resolved relatively to the GET context. Use the `location` context to navigate references instead.

Resource Identification

A web service artifact with uuid `65a2b119-9a6b-491e-8353-3692f4b9e3e5` and name `MyService` is available in the artifacts collection:

`http://localhost:port/context/em/platform/rest/artifact/`

At the following locations:

- artifactBase/65a2b119-9a6b-491e-8353-3692f4b9e3e5

- implementation/65a2b119-9a6b-491e-8353-3692f4b9e3e5

- webServiceArtifact/65a2b119-9a6b-491e-8353-3692f4b9e3e5

These URLs are not user-friendly. For newly created artifacts, HP EMauto-generates a REST name which in most cases is more user-friendly than the uuid.

This REST name can be used instead of the uuid in the URL.

```
http://localhost:port/context/em/platform/rest/artifact/webServiceArtifact/MyServic
e
```

> **Note:** If you migrate or federate resources (for example, with UDDI Registry import/export), the user-friendly URLs are lost.

User-friendly REST names remain the same, even if you change the artifact name.

Category Documents

Atom categories are a way to categorize large amounts of data. The permitted values in Atom categories can be either fixed or unrestricted. Category documents group permitted category values.

An example of a category group with a fixed set of values is the impact level criticality category group.

```
http://host:port/context/platform/rest/category-
document/uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality
```

Impact Criticality Category Document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<app:categories xmlns:app="http://www.w3.org/2007/app"
xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:hp="http://hp.com/2008/02/em/platform/model/taxonomy"
    xmlns:v355tax="http://systinet.com/uddi/taxonomy/v3/5.5"
    xmlns:v350tax="http://systinet.com/uddi/taxonomy/v3/5.0" fixed="yes"
    scheme="uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality">
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"
label="High"/>
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:medium"
label="Medium"/>
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:low"
label="Low"/>
</app:categories>
```

HP EM uses taxonomies, which are an abstraction almost identical to Atom categories. These taxonomies are sometimes transferable to Atom category documents, which can be referenced from the service document.

The categories in the taxonomy then appear as Atom categories, corresponding to the taxonomy values in artifact entries and feeds.

Atom REST Operations

To use the Atom REST interface, applications must map each operation to an HTTP request. For details, see Summary of Atom REST Operations.

Summary of Atom REST Operations

REST Operation	HTTP method	Query Field	Notes
"CREATE" below	POST	create	The path specifies the containing collection and the POST body contains an XML representation of the artifact to create.
GET	GET	None	Obtains the requested resources. For details, see "Feeds" on page 138 and "Entries" on page 146.
"UPDATE" on the next page	PUT	update	Updates the specified resource.
"DELETE" on the next page	DELETE	delete	Deletes the specified resource. GET, UNDELETE, and PURGE operations can be run on deleted resources.
"UNDELETE" on the next page	POST	undelete	Undeletes the deleted resource. It can then be updated again.
"PURGE" on the next page	DELETE	purge	Purge physically removes a resource.

Note: All writable operations use a proprietary XML representation for POST and PUT operations.

CREATE

Implemented by processing a POST request to the artifact type collection space. The POST body contains a valid XML representation of the new artifact.

```
POST http://localhost:8080/em/platform/restSecure/artifact/businessServiceArtifact
```

The content of the XML representation should match an artifact Atom entry. For details, see "Artifact Atom Entries" on page 146.

You can create artifacts conditionally using CREATE with Etags. For details, see "Atom REST ETags" on the next page.

Note: Since this operation requires an HTTP POST request, you cannot simply enter the URL into a browser. Typically the request is coded in an application. It is possible to use Javascript or HTTP command line clients.

UPDATE

Implemented by processing a PUT request to the specified collection and artifact identified with its UUID. The updated content is contained in the XML representation. For details, see "Artifact Atom Entries" on page 146.

```
PUT http://localhost:8080/em/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

> **Note:** Since this operation requires an HTTP PUT request, you cannot simply enter the URL into a browser. Typically the request is coded in an application. It is possible to use Javascript or HTTP command line clients.

DELETE

Implemented by sending a DELETE request to the specified collection and artifact identified using its UUID.

```
DELETE http://localhost:8080/em/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

UNDELETE

Implemented by sending an empty POST request to the specific collection and deleted artifact identified using its UUID. There is no XML representation associated with the POST operation for UNDELETE.

```
POST http://localhost:8080/em/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

PURGE

Implemented by sending a DELETE request to the specific collection and artifact identified by its UUID and its history feed URI.

> **Caution:** This operation cannot be undone.

```
DELETE http://localhost:8080/em/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002/history
```

Atom REST ETags

ETags enable you to perform GET, PUT, and POST operations using conditions. For example, you can use ETags to compare a response to a previously cached response to see if there are any changes to the requested resource.

> **Note:** Using ETags requires a REST client in order to specify the parameters.

You can use both *weak* and *strong* ETags.

Weak ETags are implemented by comparing the last modified time of an artifact in the repository with the time from HTTP header attributes: `If-Modified-Since` and `If-Unmodified-Since`.

Strong ETags are used mainly for caching purposes when weak ETags based on timestamps are not sufficient. For example, when an artifact has not been modified but its representation has. This happens when there is a new, changed, or missing incoming relation. ETags are random hash-generated with every artifact update.

Use ETags as described in the following topics:

- "Conditional GET" below
- "Conditional PUT and POST" below

Conditional GET

You can apply a conditional GET to determine whether a resource has changed, and then only return the representation if there is a change.

You can use a weak ETag specifying a time or a strong ETag specifying the tag attribute used to identify the revision.

Specify the time using the *If-Modified-Since* header parameter in the HTTP request.

This time is compared to the *Last Modified* attribute in the response. The *Last Modified* attribute is always returned and can be stored for future reference.

If cases where timestamps are not sufficient, you can use ETags to compare entry or feed revisions.

Specify the ETag value using the *If-None-Match* header parameter in the HTTP request.

This speoifed ETag is compared to the *ETag* attribute in the response. The *ETag* attribute is always returned and can be stored for future reference.

If the artifact has not changed, then an HTTP standard non-modified response is created with a 304 status code and proper headers are returned.

If a header parameter is not specified the latest representation is always returned.

Conditional PUT and POST

You can apply a conditional PUT or POST to determine whether a resource has changed compared to the revision you are updating, and then only apply your update if there is no change.

You can use a weak ETag specifying a time or a strong ETag specifying the tag attribute used to identify the revision.

Specify the time using the *If-Unmodified-Since* header parameter in the HTTP request.

This time is compared to the *Last Modified* attribute in the response. The *Last Modified* attribute is always returned and can be stored for future reference.

In cases where timestamps are not sufficient, you can use ETags to compare entry or feed revisions to determine whether a resource has changed compared to the revision you are updating, and then only apply your update if there is no change.

Specify the ETag value using the *If-Match* header parameter in the HTTP request.

This speoifed ETag is compared to the *ETag* attribute in the response. The *ETag* attribute is always returned and can be stored for future reference.

If the artifact has changed, then an HTTP standard preconditions-failed response is created with a 412 status code and proper headers are returned.

If a header parameter is not specified your update is applied regardless of any other changes.

Atom REST Client

The Atom REST client is an untyped API to manipulate artifacts in the repository. It is a thin layer above the Atom REST Interface.

The client provides the following features:

Model Introspection

- Enumerate Artifact types
- Enumerate Artifact properties

CRUD

- Local operations:
  - Create Artifact instance

- Server Operations
  - Create Artifact

  - Get Artifact

  - Get Artifact Data

  - Update Artifact

  - Update Artifact Data

  - Delete Artifact

  - Purge Artifact

Search

- Search criteria - name-value pairs, same property names are "ORed".
- Lists Artifacts - initialized properties depend on the given artifact type. For example, ArtifactBase has only name, description, categoryBag.
- Pagination and ordering is supported .

Classpath

JAR files are mixed with others in the installation `client/lib` folder.

- abdera-client-1.0.jar
- abdera-core-1.0.jar
- abdera-i18n-1.0.jar
- abdera-parser-1.0.jar
- axiom-api-1.2.5.jar
- axiom-impl-1.2.5.jar
- common-lang.jar
- commons-codec-1.3.jar
- commons-httpclient-3.1.jar
- commons-lang-2.3.jar
- commons-logging-1.1.jar
- jaxen-full-2.51.jar
- localization-1.0.0-alpha-3.jar
- pl-model-api.jar
- pl-model-impl.jar
- pl-remote-client.jar
- pl-remote-model.jar
- pl-xml-serialization.jar
- pl-xmlbeans-sdmconfig.jar
- pl-xmlbeans-serialization.jar
- saxpath-1.0-FCS.jar
- security.jar
- xmlbeans-2.3.0-patch.hp-3.jar

First Steps

This section provides code extracts that demonstrate working with the API. For more examples, see and the Javadocs at `http://host:port/hp-em-doc/advanced/api/index.html`.

1. Create a new RepositoryClient instance:

```
RepositoryClient repositoryClient =
  RepositoryClientFactory.createRepositoryClient

("http://localhost:8080/em",
  "demouser", "changeit", false, null, 0);
```

2. Create a new webService artifact instance and set its name:

```
ArtifactBase webService =
```

```
  repositoryClient.getArtifactFactory().newArtifact

("webServiceArtifact");
  webService.setName("Demo Webservice Name");
```

3. Store the instance on the server:

```
webService = repositoryClient.createArtifact(webService);
```

4. Get the instance from server:

```
webService = repositoryClient.getArtifact(webService.get_uuid().toString());
```

Important Classes

- **Javadoc** documentation is located at `EM_HOME/doc/api` (`[host]:[port]/hp-em-doc/advanced/api/index.html`).

- **SDM Model** documentation is located at `EM_HOME/doc/sdm` (`[host]:[port]/hp-em-doc/advanced/sdm/index.html`).

- **RepositoryClientFactory**

  - Factory used to create RepositoryClient instances.

  - The factory supports:
    - SDM Model Caching - the parameter means that the factory loads the model from the server if the cached version is older than the passed value.
    - Custom authentication (custom Abdera client factory) - see https://cwiki.apache.org/ABDERA/client.html for more information.
    - Switching off server certificate validation when using HTTPS.

- **RepositoryClient**

  - This interface contains all the important methods and getters for supporting classes.

- **ArtifactBase**

  - To get/set a particular part of an artifact use either the get or set methods.

  - Common abstraction for the untyped view of any artifact in System Data Model (SDM).

- **ArtifactData** - Artifact data holder.

- **ArtifactFactory** - Factory for creating artifact instances.

- **ArtifactRegistry** - Registry of defined artifacts.

  - **ArtifactDescriptor** - Introspective info about an artifact.

  - **PropertyDescriptor** - Introspective info about an artifact's property.

- **ValuesFactory**

  - Able to create MultiplePropertyValues, Uuid, and ArtifactData.

  - Creates instances of single property values from given values.

- **PropertiesUtil**
  - Various static helper functions for manipulating properties.

Demos

The following demo provides more code examples:

- "Atom REST Client Demo" below

Atom REST Client Demo

The purpose of this demo is to introduce the Atom REST Java client and to show how to interact with HP EM using this client. The basic operations CREATE, UPDATE, DELETE, UNDELETE, PURGE, GET, search, and model introspection are demonstrated.

1. Enumerate artifact types and service properties (`enumerateArtifactsAndProperties` method).
2. Create web service artifact and business service artifact with relation to that web service (`createGetUpdateDelete` method).
3. Create service and search that service by criticality (`createSearchDelete` method).

You can find the demo source code in: `EM_HOME\demos\client\rest\src`

To run the REST API demo:

1. Ensure that the demo is properly configured and HP EM is running.
2. Change your working directory to: `EM_HOME\demos\client\rest`
3. To get help, execute: `run`
4. To build the demo, execute: `run make`
5. To run the demo, execute: `run publish`

To rebuild the demo, execute `run clean` to delete the classes directory and `run make` to rebuild the demo classes.

Chapter 11: Lifecycle Remote Client

The Lifecycle Remote Client enables you to remotely manipulate lifecycle processes and manage the governance data of artifacts.

The following topics describe the Lifecycle Remote Client:

- "Process Management" below
- "Artifact Governance" below
- "Classpath" on the next page
- "First Steps" on page 164
- "Important Classes" on page 164

Process Management

Designed for administration of lifecycle processes remotely.

All the functionality is accessible using service `ProcessManagementService`.

An instance of `ProcessManagementService` can be created using method `createProcessManagementService()` on `GovernanceServiceFactory`

For example:

```
ProcessManagementService
service = GovernanceServiceFactory.createProcessManagementService
    ("http://localhost:8080/em","admin","changeit",true)
```

It contains methods for reading, creating, removing, copying, publishing, and editing lifecycle processes.

For more details, see the javadoc for `ProcessManagementService`.

Process information and process editing does not support all features.

Artifact Governance

Designed to manage governance of the artifact remotely.

All the functionality is accessible using service `ArtifactGovernanceService`.

An instance of `ArtifactGovernanceService` can be created using method `createArtifactGovernanceService ()` on `GovernanceServiceFactory`

For example:

```
ArtifactGovernanceService
service = GovernanceServiceFactory.createArtifactGovernanceService
```

```
    ("http://localhost:8080/em","admin","changeit",true)
```

It contains methods for the following:

- Starting governance
- Ending governance
- Changing: process, stage, or process stage and approval
- Approving the requests
- Getting:
  - Governance status for a list of UUIDs or for a governance tree identified by root artifact UUID,
  - Current stage history record
  - Voting status
  - Voting details
  - Artifacts on which voting is enabled
  - Policies and last known validation status
  - Tasks
  - StageHistoryRecords for a single artifact
  - Request approval for a root artifact UUID
- Canceling approval
- Marking a task as complete
- Voting

It also contains support for searching governed artifacts using the following methods:

- By Governance Process UUID
- By Current Stage
- By Last Approved Stage
- By Type
- By Lifecycle Status
- Conditions are always combined together

It always returns governance records. For more details see the javadoc for `ArtifactGovernanceService`.

Classpath

JAR files are mixed with others in client/lib folder.

- lifecycle-remote-api.jar
- hessian-3.1.6-patch.hp-2.jar

First Steps

This section provides code extracts that demonstrate working with the API. For more examples, see the Javadocs at `http://host:port/hp-em-doc/advanced/api/index.html`.

1. Create new Artifact Governance Service instance

```
ArtifactGovernanceService
service=GovernanceServiceFactory.createArtifactGovernanceService
    ("http://localhost:8080/em","admin","changeit",true);
```

2. Get Governance Status of an artifact

```
String artifactUuid=...
GovernanceStatus record = service.getGovernanceStatus(artifactUuid);
```

3. Request approval

```
service.requestApproval(artifactUuid,"Requesting approval.");
```

4. Get Stage History Record and iterate over approvals
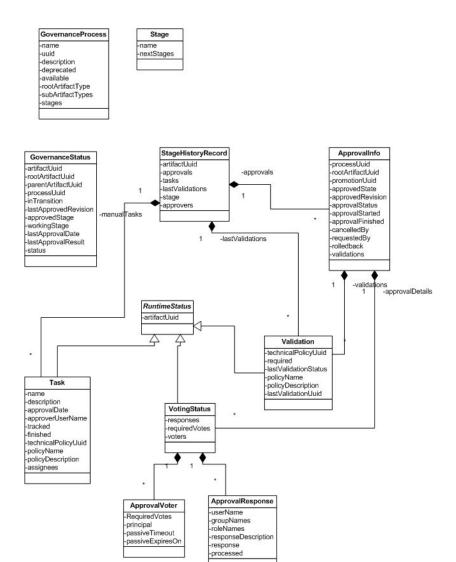
```
StageHistoryRecord
historyRecord = service.getCurrentStageHistoryRecord(artifactUuid);
for (ApprovalInfo ar : historyRecord.getApprovals()) {
  ...
}
```

Important Classes

- **Javadoc** documentation is located at `EM_HOME/doc/api` (`[host]:[port]/hp-em-doc/advanced/api/index.html`).
- **GovernanceServiceFactory** - Factory that creates services.
- **ArtifactGovernanceService** - Service for getting governance details as well as managing governance of artifacts.
- **ProcessManagementService** - Service for managing governance process.

There is a demo available that provides some code examples at `EM_HOME/demos/client/lifecycle`.

Data Structure Diagram

Chapter 12: Validation Client

The Validation Client enables you to remotely manipulate technical policies and policy reports.

The following topics describe the Validation Client:

- "Assertion Demo" below
- "Validation and Report Rendering Demo" below

Assertion Demo

This demo shows you how to develop an assertion to validate a property of EM resources. The demo utilizes the `AssertionValidator` class. See the Javadoc for a full description of this class.

In this demo, you will learn how to:

- Create a custom assertion validator.
- Use the attached project and assertion to create extension in Assertion Editor.
- Apply extension to EM server.

You can find the demo source code in `EM_HOME\demos\policymgr\assertionValidator`

To run the validation demo:

1. Ensure that HP EM is stopped.
2. Open a command prompt at `EM_HOME\demos\policymgr\assertionValidator`.
3. Enter **run make** to create extension.
4. Copy `Demos.jar` to EM_HOME\extensions
5. Enter **run apply** to apply extension to EM server.

Validation and Report Rendering Demo

This demo shows how to use the Policy Manager REST API to validate a resource. The demo utilizes the `ValidationClient` class. See the Javadoc for a full description of this class.

In this demo, you will learn how to:

- Create a service.
- Create a policy report which uses a technical policy.
- Use this policy report to validate a service.
- View the report.

You can find the demo source code in `EM_HOME\demos\policymgr\validation\src`

To run the validation demo:

1. Ensure that HP EM is running.

2. Open a command prompt at `EM_HOME\demos\policymgr\validation`.

3. Enter **run make** to compile the demo source code.

4. Enter **run run** to create the artifacts and run the validation. A link to the HTML report page is printed to the console.

Chapter 13: Report Creation

Administrators are authorized to create new reports and add them in to HP EM. This is a complicated task and requires extensive knowledge of various concepts such as DQL, script editing and data source definition syntax.

This section covers the following reports:
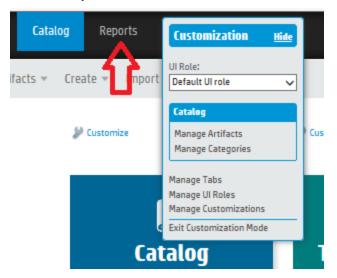
Defining the Query in Artifact Reports
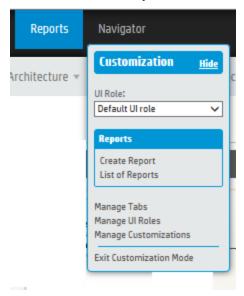
Artifact reports are defined as a part of the HP EM web UI. An administrator can change and customize the web UI directly using web UI. In this example, we will look at how to view or modify the definition of the Projects report.

To define an artifact report:

1. Log in to HP EM as administrator and select the Administration tab, and then select the Customize UI menu item from the Customization menu. The web UI switches to customization mode, in which all pages are editable. A customization box appears on the right side of every page.

2. Click the **Reports** tab.

3. Once you click the **Reports** tab, the Customization box changes to include artifact report-related items. Click **List of Reports**.

A list of all artifact reports is displayed; click the **Projects** link from the list.

4. The artifact report page is shown in the customization mode. It renders the results and displays Customize links at the top of each customizable section. The Customize link above the results table opens a window that shows an XML representation of the UI customization data that defines the report.

```xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <customization xmlns="http://soa.systinet.hp.com/2009/02/ui/customization" xmlns:cust="http://soa.systi
3      <datasource>
4          <query>
5
6  SELECT   :columns
7  FROM     projectArtifact p
8  LEFT JOIN contactArtifact c on bind(p.providedBy) and (p.providedBy.useType = 'projectManager')
9  LEFT JOIN projectFinancialProfileArtifact pfp on bind(p.financialProfile)
10 ORDER BY p.name
11
12          </query>
13     </datasource>
14     <table>
15         <!-- label Projects is SDM label  for name property -->
16         <column id="p_entityHealth_nameId" label="Health">
17             <content queryColumn="pfp.entityHealth.name"/>
18             <property name="width">70</property>
19         </column>
20         <column defaultSortDirection="asc" id="nameColumn" label="Name">
21             <content queryColumn="p.name"/>
22             <property name="width">200</property>
23         </column>
24         <column id="c_projectManager_nameId" label="Manager">
25             <content queryColumn="c.name"/>
26         </column>
27         <column id="p_plannedStartDateId" label="Planned Start">
28             <content queryColumn="p.plannedStartDate"/>
29         </column>
30         <column id="p_plannedFinishDateId" label="Planned Finish">
31             <content queryColumn="p.plannedFinishDate"/>
32         </column>
33         <column id="projectLifecycleStage" label="Lifecycle Stage">
34             <content queryColumn="p._currentStage.name"/>
35         </column>
36         <column id="projectCompletion" label="Completion">
37             <content queryColumn="p.completion"/>
38             <property name="width">80</property>
39         </column>
40         <column id="p_netPresentValueId" label="NPV">
```
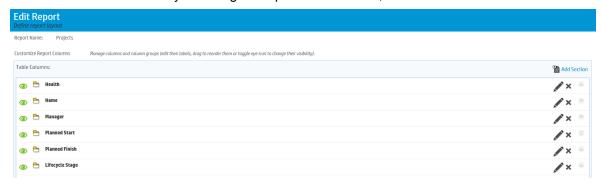
5. Click the **Edit Report** link in the header of Projects report page to access the report definition in Edit Report view.

6. In the Edit Report view, you can define and test the DQL query that creates the resulting table data. You can preview the data as you tune the query. When you finish, click the **Next** button at the bottom of the page below the project list.

7. Specify the layout of columns of the result table. You can edit column names, organize columns in sections, reorder them, or make them visible/invisible by default. Click **Next** and click the **Refresh** button to preview the changes. When you finish, click **Finish**. Alternatively, if you want to have more control over when your changes are published to users, click **Cancel**.

8. If you click **Cancel**, you remain in the customization mode of the web UI. Changes that you made in the web UI are visible only to you as an administrator. If you want to make your changes visible to all users, you can release them in a page that appears when you click the Manage Customizations link in the Customization box. Click the **Exit Customization Mode** link when you are ready to exit customization mode.

Defining Policy Reports

Policy reports are defined as a part of the HP EM web UI. In this example, we will look at how to view or modify the definition of the Archimate Compliance policy report.

To define a policy report:

1. Click the **Definition** tab on the policy report page to see how the report is defined.
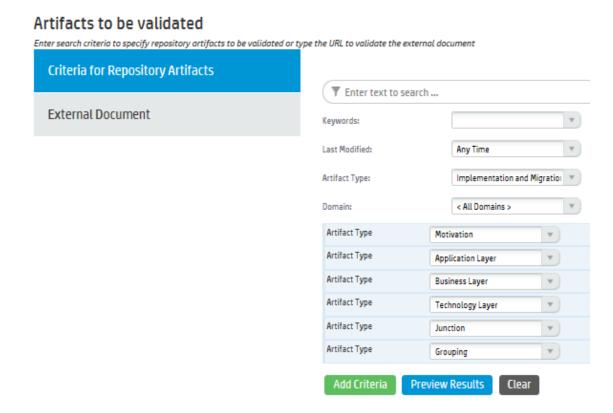
   It shows the Search Criteria that are used to filter artifacts subject to policy validation, and a list of Technical Policies that must be validated against the artifacts. An administrator can add or remove technical policies using a toolbox that appears above the list.

2. Click the pen icon in the toolbox on the right side to edit the report and view other important report definition items. The Edit page lists general fields that specify the report name, description, and target compliance by percentage. The target compliance field is used to indicate the overall goal of the policy report in percent of artifacts that should be compliant.

3. The Artifacts to be Validated section allows you specify a filter on which artifacts you want to validate. You can edit, remove, add, or clear criteria using the controls in this section. By clicking the Preview Results button, you can see artifacts that match the criteria. If required, click the Save button at the bottom of the page to save your changes.

4. You can also create a new policy report by selecting the Reports tab, and then clicking Create > Policy Reports. This will allow you to specify all the details of a new policy report on a single page.

Calculating Policy Report Results

Policy report results are cached in the HP EM repository. Depending on the number of policies and resources being validated, it can take an hour or more to calculate the results. Users who are viewing the policy report are informed that the report might not be up-to-date. Recalculation of the policy report is started in the following ways:

- **Manually**: the user manually recalculates the report. An associated action is available in the Overview tab of the report.
- **By Automatic Task**: HP EM starts a Policy Report Validation task every day around 2:30 AM. The actual time may depend on the time zone of the server on which the HP EM application is running.

To check automatic Policy Report Validation task details:

1. Log in to HP EM as administrator and select the Administration tab, and then select the Tasks menu item on the Configuration menu.

2. Click the **Policy Report Validation Task** link on the Tasks page. If you cannot see it on the page, switch to a table view by using the associated Show Task List toolbar action. The Policy Report Validation Task page shows that the task is scheduled along with a table of the execution history.

3. Click **Edit Schedule** on the toolbar to open the Edit Schedule dialog in which you can view and change the scheduling details of the task.

Creating a Heat Map Portlet

When you create a Heat Map portlet, you first need to create a Data Source Definition script to define a set of data. Then you create a Heat Map Portlet script in **JavaScript** with **Heat map** as the **Execute on** type.

To create a Heat Map portlet:

1. On the **Administration** tab, select **Customization > Manage Scripts**.

2. Create a data source:
   a. Click **Data Sources** in the left panel and click the **Create new data source script** icon. This opens the Data Source Definition editor.

   b. Add a unique name and select **XML** as the **Script language** and **DataSource Definition** as the **Execute on** option. Click **Save**.
      There are two options for creating a data source script: Closure Definition-Based_Data_Sources, or DQL-Based Data_Sources (see "Data Sources" in the Developer Guide for details).

3. Create a Heat Map portlet:
   a. Click **Reports & Portlets** in the left panel and click the **Create new portlet** icon. This opens the Portlet Script editor.

   b. Add a unique name and select **Javascript** as the **Script language** and **Heat map** as the **Execute on** type. Click **Save**.

4. Click **Edit Script** to add the script content in **JavaScript** and click **Save**.
   The following is a content example:

Edit script

```
 1  /* ****************************************************** */
 2  /* ****************************************************** */
 3  /*  The script creates Portlet HeatMap Report.          */
 4  /* ****************************************************** */
 5  /* ****************************************************** */
 6  /* ****************************************************** */
 7  {
 8      id: 'projectHeatmapReport',
 9      dataSource : '/scripts/ProjectCost.xml',   refer to DataSource Script
10      costField: 'costValue',
11      visualizations: [
12          {
13              label : 'Background color',
14              items: [
15                  {
16                      type: 'EA.portlets.visualization.BackgroundColorVisualization',
17                      field: 'density',
18                      name: 'Entity health',
19                      styleSchema: 'entity-health',
20                      showOnTooltip: false
21                  }
22              ]
23          },
24          {
25              label : 'Projects By Cost',
26              items: [
27                  {
28                      type: 'EA.portlets.visualization.Visualization',
29                      field: 'costValue',
30                      name: 'Projects By Cost',
31                      description: ')* Rectangle surface area determines planned project costs'
32                  }
33              ]
34          }
35      ] visualization settings
36  }
```

5. Click **Save**.

   After you save your new Heat Map Portlet script, you can go to the Dashboard and add the Heat Map Portlet to your dashboard view using the Dashboard menu.

Creating a Structure Map

You can use HP EM to create structure maps, an important tool to help manage infrastructure, reduce ongoing costs, and increase areas of investment within your enterprise architecture.

To create a Structure Map portlet, first create an embedded script with Structure Map execution on type.

1. On the **Administration** tab, select **Customization > Manage Scripts**.
2. Create a data source:
   a. Click **Data Sources** in the left panel and click the **Create new data source script** icon. This opens the Data Source Definition editor.

   b. Add a unique name and select **XML** as the **Script language** and **DataSource Definition** as the **Execute on** option. Click **Save**.
      There are two options for creating a data source script: Closure Definition-Based_Data_ Sources or DQL-Based Data_Sources (see "Data Sources" in the Developer Guide for details).
3. Create a Structure map:
   a. Click **Reports & Portlets** in the left panel and click the **Create new portlet** icon. This opens the Portlet Script editor.

   b. Add a unique name and select **Javascript** as the **Script language** and **Structure Map** as the **Execute on** option. Click **Save**.
4. Click **Edit Script** to add the script content in **JavaScript** and click **Save**.
   The following is a content example:

5. After you save your new structure map portlet script, you can go to the Dashboard and add the structure map portlet to your Dashboard view using the Dashboard menu.

Create a Custom Report

Overview

User can create a JavaScript report which extends from `EA.portlets.CustomReport`, and can also use external JavaScript libraries & CSS definitions in their custom report.

1. **Creating first Custom Report:**

   Go to **Administration Tab > Customization > Manage Scripts > Reports & Portlets** and click **Create new portlet** button.

2. **Create your first script:**

- Name: follow the format <your report name>.js

- Script Language: JavaScript

- Execute on: JavaScript Report

See Portlet Script image.

The class name follows format EA.scripts.<your report name>

```
/* **************************************************** */
/* **************************************************** */
/*  My First Custom Report.                             */
/* **************************************************** */
/* ****************************************************  */
/* **************************************************** */

Ext4.define('EA.scripts.MyReport', {
    extend: 'EA.portlets.CustomReport',
    initComponent: function () {
        this.callParent(arguments);
        this.update('<h1>Hello World</h1>');
    }
```

```
    })
```

Click **Save** and then navigate to **Reports** menu. Right click to add your portlet.

In image above, the category is not set. You can set category under 'Functional Category' and 'Default Height'. "See Portlet Script image." on the previous page.

3. **Query data from EM Repository:**

```
/* ****************************************************** */
/* ****************************************************** */
/*  My First Custom Report.                             */
/* ****************************************************** */
/* ****************************************************** */
/* ****************************************************** */

Ext4.define('EA.scripts.MyReport', {
        extend: 'EA.portlets.CustomReport',
        initComponent: function() {
                this.callParent(arguments);
                var self = this;
                var dqlStore = Ext4.create('EA.model.tools.DQLStore', {
```

```
                        query: '<query>SELECT a.name, a._uuid FROM
                        businessFunctionArtifact a</query>',
                        fields: ['name', '_uuid'],
                        pageSize: 10
                });
                dqlStore.on('load', function(store) {
                        self.render(store.data.items);
                });
                dqlStore.load();
        },

        render: function(items) {
                var content = '';
                for (var i = 0; i < items.length; i++) {
                        content += '<li><a href="javascript:void(0)"
                        onclick="showArtifact(\'' + items[i].data._uuid +
                        '\')">' + items[i].data.name + '</a></li>';
                }

                this.update('<div style="padding: 20px"><ul>' + content +
                '</ul></div>');

        }
        })
```
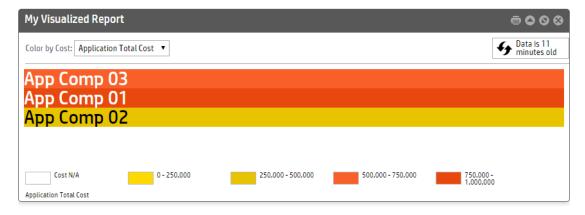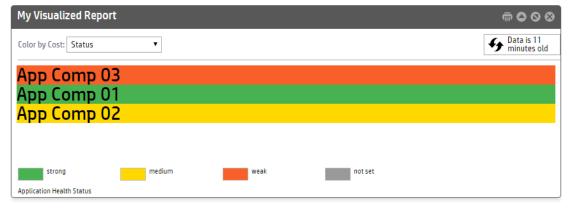
Your Portlet Name

Service Configuration & Activation
Close Customer Order
Group Enterprise Management
Manage Supply Chain Contract Variation
Initiate S/P Requisition Order
Market & Strategy Policy
Manage Customer Bill Inquiries
Track & Manage S/P Requisition
Customer Relationship Management
Business Continuity Management

4. **Visualizable Portlet:**

   To create visual report like Structure Map and Heatmap, refer the script below:

```
/* ***************************************************** */
/* ***************************************************** */
/*  My Visualized Report.                               */
/* ***************************************************** */
/* *****************************************************  */
```

```
/* ******************************************************* */

Ext4.define('EA.scripts.MyVisualizedReport', {
       extend: 'EA.portal.VisualizablePortlet',
       config: {
               id: 'MyVisualizedReport',
               dataSource : '/scripts/ApplicationCost.xml',
               visualizations: [
                       {
                               label : 'Color by Cost',
                               items: [{
                                       type: 'EA.portlets.visualization.
                                       NumberBasedColorVisualization',
                                       field: 'costValue',
                                       name: 'Application Total Cost',
                                       description: 'Application Total Cost',
                                  },{
                                       type: 'EA.portlets.visualization.
                                       BackgroundColorVisualization',
                                       field: 'density',
                                       name: 'Status',
                                       description: 'Application Health Status',
                                       styleSchema: {
                                       'Green' : {style: 'background-color:
                                       #49B250;', desc: 'strong'},
                                       'Yellow': {style: 'background-color:
                                       #FFD800;', desc: 'medium'},
                                       'Red'   : {style: 'background-color:
                                       #F85F29;', desc: 'weak'},
                                       'N/A'       : {style: 'background-color:
                                       #999999;', desc: 'not set'}
                                       }

                               }
                           ]
                       }
               ]
       },
       initComponent: function() {
               this.callParent(arguments);
       this.update('\
                       <div id="' + this.config.id + '-menu"
                       class="em-smap-menu cell-container"
                       style="width: 100%"></div>\
                       <div class = "eam-heat-map" id="' + this.config.id +
                       '-map"></div>\
                       <div class="em-smap-legend" id="' + this.config.id +
```

```
                                '-legend"></div>\
                ');
        this.mapPrepared = false;
        },
        buildMap: function(config) {
         config = config != null ? config : {};
         var self = this;
                if(config.loadData) {
                        if (this.clearCache == true){
                                this.datasource=Ext4.create
                                ('EA.model.tools.DataSourceStore',
                                { dataSource: this.config.dataSource,
                                extraParams:{clearCache: true} });
                        }else{
                                this.datasource=Ext4.create
                                ('EA.model.tools.DataSourceStore',
                                { dataSource: this.config.dataSource });
                        }
                        if (this.config.maxNode != null){
                                this.datasource.pageSize = this.config.maxNode;
                        }
                this.datasource.on('load',
                    function(store) {
                                if(config.buildVisualizationController) {
                                        self.buildVisualization(store);
                                        self.buildMenu();
                                }
                                self.datasource = store;
                                        self.mapPrepared = true;
                                        self.buildCachedMap();
                    },this);
                        this.datasource.maskElement = this.id;
                        this.datasource.load();
                } else {
                        self.buildCachedMap();
                }
        },
        buildMapContent: function() {
                this.mapContainer.html
                (this.buildNodes(this.datasource.data.items));
        },
        buildNodes: function(nodes) {
                var content = '';
                for(var i = 0; i < nodes.length; i++) {
                        content += this.buildNode(nodes[i]);
                }
                return content;
```

```
        },
        buildNode: function(node) {
                return '<div style="' + this.buildVisualStyle(node) + '">'
                + node.data.name + '</div>';
        }
})
```


5. **Load external JavaScript & CSS:**

   Invoke callback function after loading all required external resources.

   ```
   layoutManager.loadStyle(['URL1', 'URL2']);
   layoutManager.loadScript([URL1', 'URL2'], callback);
   ```
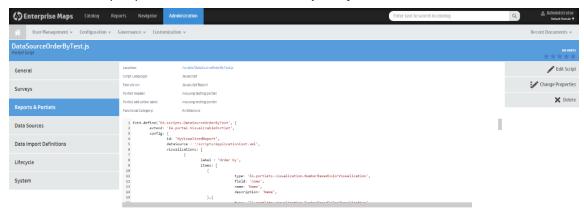
Creating a Custom Report with Ordering

Following is an example of Data Sources:

orderBy

This example provides a guideline on how to use the *orderBy* parameter from a portlet. Follow the steps given below:

- **Portlet Script**

  Create a Javascript report named *DataSourceOrderByTest.js*

  Provide below code as the script:

```javascript
Ext4.define('EA.scripts.DataSourceOrderByTest', {
    extend: 'EA.portal.VisualizablePortlet',
    config: {
        id: 'MyVisualizedReport',
        dataSource : '/scripts/ApplicationCost.xml',
        visualizations: [
            {
                label : 'Order by',
                items: [
                    {
                        type: 'EA.portlets.visualization.NumberBasedColorVisualization',
                        field: 'name',
                        name: 'Name',
                        description: 'Name',
                    },{
                        type: 'EA.portlets.visualization.NumberBasedColorVisualization',
                        field: 'costValue',
                        name: 'Cost value',
                        description: 'Cost value',
                    },{
                        type: 'EA.portlets.visualization.BackgroundColorVisualization',
                        field: 'density',
                        name: 'density',
                        description: 'density'
                    }
                ]
            }
        ]
    },
    initComponent: function() {
        this.callParent(arguments);
        this.update('\
```

```
            <div id="' + this.config.id + '-menu" class="em-smap-menu cell-container"
style="width: 100%"></div>\
            <div class = "eam-heat-map" id="' + this.config.id + '-map"></div>\
            <div class="em-smap-legend" id="' + this.config.id + '-legend"></div>\
        ');
        this.mapPrepared = false;
    },

  buildMenu: function() {
    var self = this;
        this.selectors = [];

        for(var i = 0; i < this.visualizations.length; i++) {
            if (this.visualizations[i].options.length > 1) {
                var selector = this.buildSelector(this.menuContainer,
this.visualizations[i].label, this.visualizations[i].options, this.visualizations
[i].activeIndex);
                selector.data('index',i);
                selector.change(function() {
                    var index = $(this).data('index');
                    self.visualizations[index].activeIndex = this.selectedIndex;
                    self.visualizations[index].options
[this.selectedIndex].visualization.onSelection(self,self.datasource);

                    var sortField = self.visualizations[index].options
[this.selectedIndex].field;
                    self.reload({
                     sort: sortField
                     });
                });
            }
        }
        self.buildDataSourceRecalculate(self.datasource, self.config.id + '-
menu');
  },

    buildMap: function(config, extraParams) {
        config = config != null ? config : {};
        extraParams = !extraParams ? {} : extraParams;
         var self = this;
        if(config.loadData) {
            this.buildStore(extraParams);
            this.datasource.on('load',
                function(store) {
                    if(config.buildVisualizationController) {
                        self.buildVisualization(store);
                        self.buildMenu();
                     }
```

```
                    self.datasource = store;
                    self.mapPrepared = true;
                    self.buildCachedMap();
                },this);
            this.datasource.load();
        } else {
            self.buildCachedMap();
        }
    },

    reload: function(extraParams){
        this.buildMap({loadData:true, buildVisualizationController:false},
extraParams);
    },

    buildStore: function(extraParams) {
        this.datasource=Ext4.create('EA.model.tools.DataSourceStore', {
                    dataSource: this.config.dataSource,
                    extraParams: extraParams
            }
        );
        this.datasource.extraParams.clearCache = true;
        if (this.config.maxNode != null){
            this.datasource.pageSize = this.config.maxNode;
        }
        this.datasource.maskElement = this.id;
    },

    buildMapContent: function() {
        this.mapContainer.html(this.buildNodes(this.datasource.data.items));
    },
    buildNodes: function(nodes) {
        var content = '';
        for(var i = 0; i < nodes.length; i++) {
                content += this.buildNode(nodes[i]);
        }
        return content;
    },
    buildNode: function(node) {
        return '<li><div style="color:black;font-size:16px !important;">' +
node.data.name + '</div></li>';
    }
})
```

- **Data Source Script**

   Create a data source script named *ApplicationCost.xml* with below code:

```
<closure maxDepth="1" maxResults="25" maxProcessingTime="10000"
seedsAsResults="true">
```

```
    <defaultSeedQuery>
       select f._uuid from applicationComponentArtifact f LEFT JOIN
appFinancialProfileArtifact pfp on bind(f.financialProfile) where not exists
        (select 1 from applicationComponentArtifact f1 join
applicationComponentArtifact f2
                using f2.composedOf where f1._uuid=f._uuid) and
pfp.annualCostTotal is not null
    </defaultSeedQuery>

    <resultArtifacts>
        <artifact sdmName="applicationComponentArtifact">
           <field name="costValue" query="SELECT pfp.annualCostTotal as costValue
FROM  applicationComponentArtifact p LEFT JOIN appFinancialProfileArtifact pfp on
bind(p.financialProfile) where p._uuid=:uuid"/>
           <field name="density" query="SELECT pfp.entityHealth.val as density
FROM  applicationComponentArtifact p LEFT JOIN appFinancialProfileArtifact pfp on
bind(p.financialProfile) where p._uuid=:uuid"/>
        </artifact>
    </resultArtifacts>

    <traversableArtifacts>
        <artifact sdmName="applicationComponentArtifact"/>
    </traversableArtifacts>

    <traversableRelations>
        <relation sdmName="composedOf"/>
    </traversableRelations>

</closure>
```

- **Execute**

  Add the created portlet to the Reports tab and test it.