OpenIDM 6.2.1-SNAPSHOT Developer's Guide

MarkCraig
PaulBryan
AndiEgloff
LaszloHordos

,
, ,

Table of Contents

Preface

This guide shows you how to work with OpenIDM APIs to integrate data stores and applications in your organization with OpenIDM services for identity management, provisioning, and compliance.

1 Who Should Use this Guide

This guide is written for Java and web developers who build OpenIDM services into the workflows of their organizations.

This guide starts by explaining identity management with OpenIDM briefly, and describing best practices for integration OpenIDM services into your organization. Then it demonstrates how to connect your applications with OpenIDM services and to build workflows for identity management, provisioning, and compliance.

You do not need to be an OpenIDM wizard to learn something from this guide. You do need some background in writing Java and web applications to get the most out of this guide. You can nevertheless get started with this guide, and then learn more as you go along.

2 Using Samples

See the license for the specific language governing permissions and limitations under the license.

3 Formatting Conventions

Note

Pay attention to notes like this one.

Some items might be formatted differently from other text, like `filenames`, **commands**, and `literal values`.

```
$ echo Terminal sessions are formatted like this.
Terminal sessions are formatted like this.
```

```
class Test
{
    public static void main(String [] args)
    {
        System.out.println("This is a program listing.");

    }
}
```

In many cases, sections pertaining to UNIX, GNU/Linux, Mac OS X, BSD, and so forth are marked (UNIX). Sections pertaining to Microsoft Windows might be marked (Windows). To avoid repetition, however, file system directory names are often given only in UNIX format as in `/path/to/OpenIDM`, even if the text applies to `C:\path\to\OpenIDM` as well.

Warning

Ignore warnings at your own risk.

4 Accessing OpenIDM Documentation Online

Core documentation, such as what you are now reading, aims to be technically accurate and complete with respect to the software documented. Core documentation therefore follows a three-phase review process designed to eliminate errors. The review process should slow authors down enough that documentation you get with a stable release has had time to bake fully.

Fully baked core documentation is available at ....

The OpenIDM Wiki regularly brings you more, fresh content. In addition, you are welcome to sign up and then edit the Wiki if you notice an error, or if you have something to share.

5 Joining the OpenIDM Community

After you sign up at ForgeRock, you can also login to the Wiki and the issue database to follow what is happening with the project.

If you have questions regarding OpenIDM which are not answered by the documentation, there is a mailing list which can be found at https://lists.forgerock.org/mailman/listinfo/openidm where you are likely to find an answer.

The Wiki has information on how to check out OpenIDM source code. There is also a mailing list for OpenIDM development which can be found at https://lists.forgerock.org/mailman/listinfo/openidm-dev Should you want to contribute a patch, test, or feature, or want to author part of the core documentation, first have a look on the ForgeRock site at how to get involved.

Part A. Overview

TODO

Chapter 1
OpenIDM APIs and Protocols

1.1 Development

Pages in this section should be aimed toward developers working on OpenIDM.

1.1.1 Dependencies

This page outlines the external library dependencies that OpenIDM has with a brief description for each. For the exact version dependency consult the maven POM files.

1.1.1.1 Runtime dependencies

These are prerequisites to deploy and run the OpenIDM zip package.

Oracle Java SE JDK 6 Update 24+

The latest stable version of the Sun Java Development Kit and runtime. We may elect to switch to OpenJDK in future, but there are known bugs that are still in it that are known to be fixed in Sun's.

1.1.1.2 Build dependencies

These are dependencies that are required to build OpenIDM from source.

Maven 3.0+

Manages the build of the OpenIDM project from source. Resolves module dependencies, compiles code, executes testing, provides build and test reporting and documentation.

1.1.1.3 Bundled dependencies

These are runtime dependencies that we include in our zip package distribution. These are automatically resolved by Maven during the build.

Apache Felix

The OSGi modularity framework that is bundled with the default packaging. We leverage additional optional bundles, such as the web console.

Jackson

A high-performance JSON processor. In most cases, we will be using this for simple data binding to standard Java data types: Map, List, String, Number, Boolean. Has good integration with Restlet for providing JSON representation of in-memory object structures.

Jetty/Pax Web Bundle

Servlet Container and HTTP service embedded as an OSGi bundle.

OrientDB

An embeddable NoSQL database that is bundled with OpenIDM as its default data store.

Quartz Scheduler

Scheduling service to perform periodic jobs. We currently use a wrapped version with an OSGi manifest created by servicemix.

Restlet

Provides a resource-oriented architecture framework for exposure of objects via a RESTful HTTP API. Allows for more dynamic resource taxonomy and routing than other frameworks such as JAX-RS.

Rhino

Rhino is an open-source implementation of JavaScript written entirely in Java. It is typically embedded into Java applications to provide scripting to end users.

SLF4J

The Simple Logging Facade for Java, a facade for Java logging to defer binding to a particular logging implementation to deployment time. Extremely small, with nice features such as parameterized logging as well as nested and mapped diagnostic contexts (NDC/MDC).

1.1.1.4 Test dependencies

These are dependencies that are required to perform unit and integration testing of OpenIDM. These are automatically resolved by Maven during the unit test phase.

TestNG

A testing framework for unit, functional, end-to-end and integration testing.

FEST-Assert

A library for writing fluent assertions in unit tests.

1.1.2 Guidelines

1.1.2.1 Design guidelines

- Adhere to resource-oriented architecture patterns wherever practical.
- Maintain JavaScript object model for objects with desired interoperability with applications, services, scripts.
- Use OSGi bundles for loose coupling of components.

1.1.2.2 Coding guidelines

All coding conventions should follow the Code Conventions for the Java Programming Language, with the following exceptions:

- Indentation: spaces, no tabs.
- Line length limit: 128 characters.

Chapter 2 Distributed Task

The purpose of the distributed task mechanism is to allow for scaling and fail-over capabilities.

The implementation utilizes the existing shared infrastructure of the cluster nodes in the form of the shared repository (e.g. database) rather than adding another shared mechanism to install and administer. This mechanism is suitable for medium frequency/coarse grained jobs. If a need arises for fine grained/very frequent task distribution, utilizing an existing mechanism such as the http front-end is a possibility.

2.1 Approach Basics

The data store (repository) is used to add new tasks that any node can pick up (poll based model) and claim. Upon claiming a task, the node "leases" the task for a specified amount of time (e.g. 5 minutes). As it processes the task it renews the lease, i.e. regularly asks again for the lease to expire in 5 minutes. If the node for any reason stops processing (e.g. crashed) the lease expires, and another node is free to claim the task. The node can store a result in the task. This mechanism operates similar to a map/reduce mechanism.

2.2 High Availability / HA

The lease mechanism provides for automatic fail-over if a task does not make any progress anymore.

There is no leader and hence no single point of failure or need to re-elect leaders.

2.3 Scaling

A node can split out a task (such as a recon batch) into many pieces (such as 1000 records to recon each), and create individual tasks that can be claimed and processed by many nodes in parallel. By grouping the tasks a subsequent "join" task can be triggered that can process the completion of the whole batch.

2.4 Task Handlers

Nodes in the cluster upon start/up configuration should register all task handlers they are willing to handle. Nodes will only pick up work for task types it has handlers for, this is primarily intended to remove start-up race conditions, but can also be used to disable certain kinds of tasks (e.g. reconciliation) on a given node.

2.5 Implementation Notes

Each node has a scheduled activity that periodically checks for new work, and scans the tasks for abandoned tasks.

2.6 Concurrency/Locking Notes

The underlying store must provide for some way to allow a node to uniquely claim a task, e.g. optimistic concurrency to check that the "claim" update succeeded rather than conflicted with another node already claiming the task.

There is no guaranteed ordering amongst the distributed tasks; they can happen in any order and concurrently.

2.7 Disallowing Concurrent Execution and Serialized Executions in the Cluster

When there is a requirement that a given functionality is only executed by one node at any given time in the cluster, this mechanism can be used as well. If the nodes know of a unique name/id for the functionality (e.g. "check-database-consistency", they can assign that id as the taskId and ask the submit to ignore if the tasks entry already exists in the DB. This way even if multiple nodes ask to submit the same task, only one task will be stored and hence picked up.

Queing serialized execution of tasks is currently not supported, i.e. note that the task is only executed once. Multiple submits are not queued for serialized execution, duplicate submits are discarded/ignored. We could add a submitSerialExecution() capability going forward if required (which would queue the tasks and only serially process each in the cluster).

2.8 QoS / Transactionality

The task is processed on an at-least-once approach; i.e. until the task handler says it completed the task successfully, or it says there is something wrong in the task and it should not be re-tried - triggering an error handler.

2.9 Sub-tasks and Grouping

A group name (in the context of task type) provides for a way to define a "join" on the group - a task that gets triggered when the whole group finished (either successfully or not).

A group task's unique identifier consists of the task type (e.g. "reconcile-set-of-records"), a group name ("reconcile-system-x"), task number in group(10), group total tasks (100). The group task generation/submission should be done in a fashion that re-doing the generation results in the same identifier.

When the total number of task that will be submitted for a group is unknown whilst generating the tasks, only the last record submitted for group has both the task number in group and group total tasks, which must be the same (100/100). This is necessary for the system to detect when the whole group has finished. Previous tasks only have the task number in group set. The generation of tasks within a group should be idempotent, i.e. it should be able to re-do the task generation in a fashion that duplicates are ignored because the taskid is the same again.

A task can submit sub-tasks through the handler provided context. This implies that the spawning task is the parent of these sub-tasks. This allows for an enclosing task to a) ensure only one enclosing task of the same type (controlled through task id) runs at the same time in the cluster, and b) there is a place to handle a "join" (group finished handler) of the overall task, after all the sub-tasks are finished.

Only when the sub-tasks all finished will the parent task's "group finished handler" be triggered. For example, if only one reconciliation for a given system should run at any given time, and it is broken into sub-tasks a Handler for Single task "[recon-parent][recon-system-x] will submit 100 sub-tasks as follows:

- Submit single task with task type [recon-parent], task name "recon-system-x".
- Node 1 picks up "[recon-parent][recon-system-x]", submits 100x grouped tasks with task type "recon-record-set", group name "recon-system-x", record numbers 1-100 / 100
- All nodes start picking up and processing "[recon-record-set][recon-system-x][<batch number>/100]" tasks
- When all "[recon-record-set][recon-system-x][<batch number>/100]" tasks finished, a task gets inserted to trigger the group finished handler for "recon-batch".
- A node picks up and handles the group finished handler 'task' for "recon-record-set"
- Upon completion of the "[recon-record-set][recon-system-x][group-finished-task]" group finished handler, a task gets inserted to trigger the group finished handler on the parent task, "[recon-parent][recon-system-x][group-finished-task]"
- A node picks up and handles the group finished handler task for "[recon-parent][recon-system-x][group-finished-task]"

2.10 Scheduling

Rather than having the scheduler be cluster aware, every node will have its own local scheduler - configured with a homogenous schedule. Where there are tasks that should be triggered only once cluster-wide this can be achieved by inserting a task with a taskId that is the same for all nodes, and for example encodes the configured trigger time. The submit should be configured to ignore if the task already exists. Hence, examples of some variations upon a scheduler triggering:

- Execute something immediately on each node: no need to use the distributed task
- Execute something once for this timer, on only one node: insert a distributed task with taskId "check-database-<configured trigger date/time>'
- Execute something at most once in parallel, even if different timers trigger: insert a distributed task with taskid "active-sync-systemx"
- Note that this would result in a subsequent configured active sync schedule of same type to be ignored if the existing still is in progress
- Execute a group of tasks at most once in parallel, even if different configured timers trigger: insert a distributed task to spawn sub-tasks within a group, i.e.
- Insert a distributed taks with taskid "recon-systemx" (group "recon-parent"), which spawns sub-tasks with taskids "recon-systemx-<batch number>" (group "recon-batch")
- Note that this would result in a subsequent configured active sync schedule of same type to be ignored if the existing still is in progress
- Queueing and Serializing executions not yet supported, i.e. can't queue group y to only start when group x finished.

2.11 API

```
package org.forgerock.openidm.distributedtask;


/**
 * Distribute tasks across the cluster nodes for both scaling and HA/fail-over
 */

public interface DistributedTask {

    /**
     * @param failGroupFast if true a single task failure within the group
     * aborts further processing of the group and eventually triggers the
     * group handler
     * @param taskData the data associated with the task. Content must be
     * either 1. String, 2. JSON based object model and/or 3. Serializable
     * @return Task ID
     */

    String submit(TaskIdentifier taskId, int priority, Object taskData);

    /**
     * Register the implementation for a given task type
     * Will get invoked when a node claims a task and hands it to this handler.
     * The handler is responsible for executing the task, maintaining the
     * lease as it is processing, and for returning results if any
     */

    void registerTaskHandler(String taskType, TaskHandler taskHandler);

    /**
     * Handle if a group is finished processing, either successfully
     * (all tasks) or not.
     */

    void registerGroupFinishedHandler(String taskType, TaskHandler taskHandler);

    /**
     * Optional handler to customize how failed tasks are treated. Default
     * is to log, remove individual task and trigger group complete handling
     * if appropriate.
     */

    void registerInvalidTaskHandler(TaskHandler taskHandler);

}


public interface TaskIdentifier {

    String getStringifiedTaskId();

}



public class SingleTaskId implements TaskIdentifier {

    public SingleTaskId(String taskType, String taskName) {...};

}


public class GroupedTaskId implements TaskIdentifier {

    public GroupedTaskId(String taskType, String groupName,
        boolean failGroupFast, int taskNumber, int totalGroupTasks) {...};

}


public abstract class TaskHandler {

    /**
     * @param leaseSeconds sets the initial number of seconds the task
     * handler should own the task, it should be
     * plenty of time to either process the task, or to renew the lease;
     * but at the same time be reasonably short to achieve the desired
     * fail-over times when a handler/node has a problem and stops
     * processing/renewing the lease.
     */

    public void setInitialLeaseTime(int leaseSeconds) {

    }

    /**
     * @param taskData The data for the task to process. For a regular task
     * this data is what was submitted, for group finished tasks this
     * contains the success/failure and results of all the tasks that ran in
     * the group.
     * @return optional result, which can be used in the group complete
     * handler
     */

    Object process(TaskIdentifier taskId, int priority, Object taskData,
        LeaseManager leaseManager) throws InvalidTaskException,
        RetryLaterException, LeaseExpiredException;

}


public interface LeaseManager {

    /**
     * Renew the lease (the time allotted for the task handler to process the
     * next step and before someone else should assume that this taskHandler
     * has an issue and another node should take over. The lease time should
     * be selected so that during normal operation it is plenty of time for
     * the taskHandler to renew the lease for continous processing.
     * An example would be a lease of 5 minutes, and the taskHandler expects
     * to renew the lease every ~30 seconds.
     * To allow for frequent lease updates without unnecessary performance
     * overhead, the lease manager may batch/time writes to the persistence
     * store, but it must first check continued ownership of the task with
     * sufficient time.
     * @param leaseSeconds how many seconds from current time the lease
     * should expire
     * @throws LeaseExpiredException if the lease renewal did not succeed.
     * The task handler is required to abort the task processing as it now
     * does not own the task anymore.
     */

    boolean renewLease(int leaseSeconds) throws LeaseExpiredException {}

}
```

Chapter 3
Object model

Artifacts handled by OpenIDM are Java object representations of the JavaScript object model as defined by JSON. This supports interoperability and potential integration with a vast number of applications, services and programming languages. As OpenIDM is a Java-based product, these representations are instances of classes: Map, List, String, Number, Boolean and null (known in this documentation by the initialism: MLSNBN). OpenIDM can serialize/deserialize these structures to/from JSON as required.

OpenIDM exposes a set of triggers and functions that system administrators can define in JavaScript (with the ability to expand into other scripting and programming languages in the future), which can natively read and modify these JSON-based object model structures.

Chapter 4
Repository

The repository provides a common abstraction for a pluggable persistence layer. Plugged in repositories could be NoSQL, relational databases, ldap etc.

The API operates with (JSON-based) object model parameters/results, with the same RESTful principles as the other internal APIs.

Aside from get, put, patch, delete it also provides for queries for sets of objects.

4.1 Query

Repository Query

Implementation Phases

1. In a first phase native queries are supported
2. In a second phase a common query expression independent of the underlying technology is planned.

Assumptions

1. We will provide for native queries as a baseline. This way even when we add a common query language doesn't have to cover every possibility

2. A common expression/language to define queries, independent of the underlying data store is desirable in the long run

3. The client API provides for both the ability to execute pre-defined, parameterized queries, and the ability to supply the query as a parameter as well.

Query Configuration

Queries can be configured either on the OrientDB service, i.e. the queries section of the file

conf/org.forgerock.repo.orientdb.json

Or they can be declared in the context of the calling module if it passes the expression to the repository as part of the query.

Query Types

The default is to express the query in the native language of the underlying store.

An optional querytype can be added going foward to explicitly declare the query type, e.g.

- Native query languages(s) - Languages/expressions directly understood by the underlying persistence store

- Common query languages(s) - NOTE: format TBD (*)

examples:

QueryType NATIVE

QueryType COMMON_JSON_PATH (*)

QueryType COMMON_SQL (*)

Common Query Formats

To be decided

Query Tokens

Queries can contain tokens of the format ${token-name}, which will get replaced from parameters passed to the query.

Queries can be of type String or List.

At token of type List typically will be converted into a repository specific string representation to insert into the query, e.g. for an orientdb "query of "select ${_fields} from managed/user" a List of fields with firstname, lastname, email will be expanded to "select firstname,lastname,email from managed/user"

Query API

Object query(String id, Map params)

id:

identifies the resource set to query against, such as a relative uri for all "user" objects

params:

set of parameters that can be substituted as tokens into the query. Reserved property keys are defined in QueryConstants.

1. The reserved property _queryId (QueryConstants.QUERY_ID) can specify a pre-configured parameterized query to execute
2. The reserved properties _queryExpressions ((QueryConstants.QUERY_EXPRESSION) (and optional _query-type) allow to pass add-hoc queries rather than specifying a query by id
3. The optional reserved properties _page-from (*), _page-size (*), _page-direction (*) can be used to page through the results either forwards or backwards. For the first page, the _page-from property can be empty; in the result meta-data the query will return two tokens for subsequent use in the _page-from parameter - a token representing the first record to page backwards, and a token representing the last record to page forward

The system also populates the OrientDB document class name in the property "_resource" (QueryConstants.RESOURCE_NAME), which can be resulved as a query token, e.g.

select * from ${_resource}

return value:

a Map<String, Object> JSON based object model

The actual result set "records" are in the "result" map entry (QueryConstants.QUERY_RESULT) and of format List<Map<String, Object>>. Each list entry is a (potentially parts of) an object matching the query. If not all fields of an object were selected, only the selected parts of the object will be present.

Additional meta-data is available at the top level of the returned map, e.g. QueryConstants.STATISTICS_QUERY_TIME (time to execute the query in ms) or QueryConstants.STATISTICS_CONVERSION_TIME (time to convert the results to the object model in ms)

The metadata in the future can contain more information about the query, such as paging token for the first and last result to page backwards/forwards.

The paging mechanism does not guarantee isolation or consistency across the consecutive paging calls, i.e. changing the underlying data in between/during paging calls(s) will show up in the results.

Query Definition

The query definition happens based on structure of the stored object, e.g.

1. The mapping object which links arbitrary objects (managed and/or system objects)
2. The managed object structure and its associated schema
3. Intermediate objects constructed for recon and sync purposes
4. Objects constructed for reporting purposes

The query can contain tokens (e.g. ${firstname}) that will get substituted at runtime by the value supplied in the params map argument

For example, assuming a managed object "managed/user"

Example 4.1.

```
{
    "_id" : "999",
    "_rev" : "0",
    "firstname" : "John",
    "lastname" : "Doe",
    "address":
    {
         "street":"Some ln",
         "city":"Somewhere",
         "zip":"99999"
    }
}
```

Example associated queries

Example 4.2.

```
queries: {
    "my-query" : "select firstname, lastname from user where address.city = '${city}'"
}
```

The call to Managed Object API can look something equivalent to the following (e.g. from the REST API, or from recon code)

```
Map<String, String> params = new HashMap<String, String>();
params.put(QueryConstants.QUERY_ID, "my-query");
params.put("city", "Some");

Map<String, Object> result = query("managed/user", params);

List<Map<String, Object>> records = (List<Map<String, Object>>) result.get(QueryConstants.QUERY_RESULT);
for (Map<String, Object> entry : records.entrySet()) {
    entry.get("firstname");
    entry.get("lastname");
    entry.get("_id");
    entry.get("_rev");
}
Long queryTimeMs = (String) result.get(QueryConstants.STATISTICS_QUERY_TIME);
Long conversionTimeMs = result.get(QueryConstants.STATISTICS_CONVERSION_TIME);

// Not yet supported
// result.get(QueryConstants.PAGING_TOKEN_FIRST);
// result.get(QueryConstants.PAGING_TOKEN_LAST);
```

It may pass this query directly to the repository service, "as is". The specific repository service implementation, e.g. the repo-orientdb would have read in the query definitions. If necessary it can translate a common query format into something it natively understands, or - as in this example - use the native query directly.

Performance Considerations

Care needs to be taken to create appropriate indexes on queried fields.

When creating queries with tokens, note that OrientDB can only make prepared statements with tokens in the "where" clause. Hence if tokens are used elsewhere, the repository will create queries on demand with the replaced tokens.

Comments

TODO: make sure the API aligns with the query API specified on the managed object

1. Might need to provide less rigid pre-defined queries, e.g. being able to search a user given any number of combinations of fields

2. The common query technology choice is a concern.

3. JSONPath (or JSONQuery) is not a standard and not well known. Although technically it fits nicely into the architecture, this may not appeal that much to folks that might be DB developers/admins.

4. SQL is more familiar to many, but we may have to invent a query syntax that allows querying embedded documents. It also is not terribly JSON related.

5. XPath is another technology that is pretty well known, but obviously is typically thought of as XML related - although JXPath provides queries on arbitrary java objects for example.

6. The common query flexiblity/effort is a concern: Depending on the extent of what needs to be expressable this could become a considerable effort for every persistence option, including concerns about lowest common denominator and other incompatibilities.

7. Rather than simple token substitution from a single dimension map we could support notations to traverse objects, akin to the JSP expression language (EL). We could also then include whole resource "objects" as a paramter map

entry, e.g. "${systemobject.firstname}". This could remove the need for a javascript mapping to extract parameters for simple cases.

Chapter 5
Terminology

5.1 Managed object

An object modeled to represent the identity-related data managed by OpenIDM and potentially across multiple resources.

5.2 Mapping object

An artifact that defines the mappings between the properties of a managed object type and the properties of a resource object type.

5.3 Object

An object, per the object model. Addressable objects have identifiers that can be used to establish a location where the object can be accessed, per OpenIDM's resource oriented architecture.

5.4 Resource

An application, service or data store with which OpenIDM synchronizes data with its own internal managed objects.

5.5 Resource object

An artifact that is stored in a resource, which OpenIDM accesses during provisioning, synchronization and reconciliation.

5.6 Schema object

An artifact that defines the structure and semantics of an object.

Chapter 6

Best Practices For OpenIDM Integration

This chapter presents best practices to keep in mind when integrating and extending OpenIDM services in your organization.

Index