Apache Syncope Reference Guide

Version 2.0.4-SNAPSHOT

Table of Contents

1. Introduction . . . 2
1.1. Identity Technologies . . . 2
1.1.1. Identity Stores . . . 2
1.1.2. Provisioning Engines . . . 4
1.1.3. Access Managers . . . 5
1.1.4. The Complete Picture . . . 5
2. Architecture . . . 7
2.1. Core . . . 7
2.1.1. REST . . . 7
2.1.2. Logic . . . 8
2.1.3. Provisioning . . . 8
2.1.4. Workflow . . . 9
2.1.5. Persistence . . . 9
2.1.6. Security . . . 9
2.2. Admin UI . . . 10
2.3. End-user UI . . . 10
2.4. CLI . . . 10
2.5. Third Party Applications . . . 10
2.5.1. Eclipse IDE Plugin . . . 10
2.5.2. Netbeans IDE Plugin . . . 10
3. Concepts . . . 11
3.1. Users, Groups and Any Objects . . . 11
3.1.1. Password Reset . . . 12
3.2. Type Management . . . 13
3.2.1. Schema . . . 13
Plain . . . 13
Derived . . . 14
Virtual . . . 14
3.2.2. AnyTypeClass . . . 15
3.2.3. AnyType . . . 15
3.2.4. RelationshipType . . . 17
3.2.5. Type Extensions . . . 18
3.3. External Resources . . . 19
3.3.1. Connector Instance details . . . 20
3.3.2. External Resource details . . . 21
3.3.3. Mapping . . . 21
3.4. Realms . . . 23
3.4.1. Realm Provisioning . . . 24

3.4.2. LogicActions . . . 24
3.5. Entitlements . . . 24
3.6. Roles . . . 25
3.6.1. Delegated Administration . . . 25
3.7. Provisioning . . . 26
3.7.1. Overview . . . 26
3.7.2. Propagation . . . 28
PropagationActions . . . 29
3.7.3. Pull . . . 29
PullActions . . . 31
3.7.4. Push . . . 31
PushActions . . . 32
3.8. Policies . . . 32
3.8.1. Account . . . 33
Account Rules . . . 33
Pass-through Authentication . . . 34
3.8.2. Password . . . 34
Password Rules . . . 34
3.8.3. Pull . . . 35
Pull Correlation Rules . . . 36
3.8.4. Push . . . 36
3.9. Workflow . . . 36
3.9.1. Activiti User Workflow Adapter . . . 37
Approval . . . 37
Identity Recertification . . . 39
3.9.2. Flowable User Workflow Adapter . . . 40
3.10. Notifications . . . 41
3.10.1. Notification Events . . . 41
3.10.2. Notification Templates . . . 42
3.11. Tasks . . . 43
3.11.1. Propagation . . . 43
3.11.2. Pull . . . 44
3.11.3. Push . . . 44
3.11.4. Notification . . . 45
3.11.5. Custom . . . 45
3.12. Reports . . . 46
3.12.1. Report Templates . . . 46
3.12.2. Reportlets . . . 46
Static Reportlet . . . 47
User and Group Reportles . . . 47
Reconciliation Reportlet . . . 47

Audit Reportlet . . . 47

3.13. Audit . . . 47

3.13.1. Audit Events . . . 47

3.14. Domains . . . 48

3.15. Extensions . . . 48

3.15.1. Apache Camel Provisioning Manager . . . 49

3.15.2. Swagger . . . 49

3.15.3. SAML 2.0 Service Provider . . . 49

3.15.4. Elasticsearch . . . 50

4. Working with Apache Syncope . . . 51

4.1. Admin Console . . . 51

4.1.1. Pages . . . 51

Dashboard . . . 51

Realms . . . 52

Topology . . . 52

Reports . . . 53

Configuration . . . 53

Approval . . . 54

Extensions . . . 55

4.2. Enduser Application . . . 55

4.3. CLI . . . 56

4.3.1. Commands . . . 57

Schema command . . . 57

Connector command . . . 58

Resource command . . . 58

User command . . . 59

Group command . . . 61

Any command . . . 62

Role command . . . 63

Realm command . . . 63

Question command . . . 64

Configuration command . . . 64

Logger command . . . 65

Task command . . . 66

Notification command . . . 67

Report command . . . 68

Policy command . . . 69

Info command . . . 69

Entitlement command . . . 70

Domain command . . . 71

4.4. RESTful services . . . 71

4.4.1. REST Authentication and Authorization . . . 72
4.4.2. REST Headers . . . 72
X-Syncope-Token . . . 72
X-Syncope-Domain . . . 73
X-Syncope-Key and Location . . . 73
X-Application-Error-Code and X-Application-Error-Info . . . 73
X-Syncope-Null-Priority-Async . . . 73
Prefer and Preference-Applied . . . 73
ETag, If-Match and If-None-Match . . . 73
X-Syncope-Entitlements . . . 74
4.4.3. Bulk Operations . . . 74
4.4.4. Client Library . . . 75
4.5. Customization . . . 79
4.5.1. Core . . . 82
4.5.2. Console . . . 87
4.5.3. Enduser . . . 87
i18n . . . 88
Form customization . . . 89
4.5.4. Extensions . . . 92
4.6. System Administration . . . 92
4.6.1. DBMS . . . 93
PostgreSQL . . . 93
MySQL . . . 94
MariaDB . . . 95
Oracle Database . . . 95
MS SQL Server . . . 96
4.6.2. Database Connection Pool . . . 96
4.6.3. JavaEE Container . . . 97
Apache Tomcat 8 and 8.5 . . . 97
Glassfish 4.1 and Payara Server . . . 97
Wildfly 9 and 10 . . . 98
4.6.4. High-Availability . . . 100
4.6.5. Domains Management . . . 101
4.6.6. Set admin credentials . . . 102
4.6.7. Configure ConnId locations . . . 102
4.6.8. Deal with internal storage export - import . . . 104
Export . . . 104
Import . . . 105
4.6.9. Enable the Activiti Modeler . . . 105
4.6.10. Install connector bundles . . . 105
Local sources . . . 105

Run-time 107
4.6.11. E-mail Configuration 107
4.6.12. Keystore 108
4.6.13. Configuration Parameters 110
4.7. Migration from Apache Syncope 1.2 111
4.7.1. Preparation 111
4.7.2. Migrate configuration 111
4.7.3. Migrate users and roles 113

ℹ *This document is under active development and discussion!*

If you find errors or omissions in this document, please don’t hesitate to submit an issue or open a pull request with a fix. We also encourage you to ask questions and discuss any aspects of the project on the mailing lists or IRC. New contributors are always welcome!

Preface

This reference guide covers Apache Syncope services for identity management, provisioning, and compliance.

Chapter 1. Introduction

Often, *Identity Management* and *Access Management* are jointly referred, mainly because their two management worlds likely coexist in the same project or in the same environment.

The two topics are however completely different: each one has its own context, its own rules, its own best practices.

On the other hand, some products provide unorthodox implementations so it is indeed possible to do the same thing with both of them.

Identity Management
: Tools and practices to keep identity data consistent and synchronized across repositories, data formats and models.

Access Management
: Systems, protocols and technologies supporting user authentication (how Users are let accessing a given system) and authorization (which capabilities each user owns on a given system).

From the definitions above, Identity Management and Access Management can be seen as complementary: very often, the data synchronized by the former are then used by the latter to provide its features - e.g. authentication and authorization.

ℹ Functionally, **Apache Syncope** implements **Identity Management** features.

1.1. Identity Technologies

Identity and Access Management (IAM) is not implemented by a single technology; it is instead a composition of heterogeneous technologies - differing by maturity, scope, applicability and feature coverage - which require some 'glue' to fit together.

As with other application domains, it can be observed that tools that appeared earlier tend to partially overlap with more recent, targeted products.

1.1.1. Identity Stores

Identity Stores are the places where identity-related information is stored.

An Identity Store can be shared among several systems: as a result, there is a single place where account data is managed by administrators, and the same password can be used for the same user for accessing different applications.

Various Identity Store types are available:

- Flat files (XML, CSV, …)

- LDAP
- Relational databases (MySQL, Oracle, Microsoft SQL Server, PostgreSQL, …)
- Plaform-specific (Microsoft Active Directory, FreeIPA, PowerShell, …)
- Web services (REST, SOAP, …)
- Cloud providers
- …and much more.

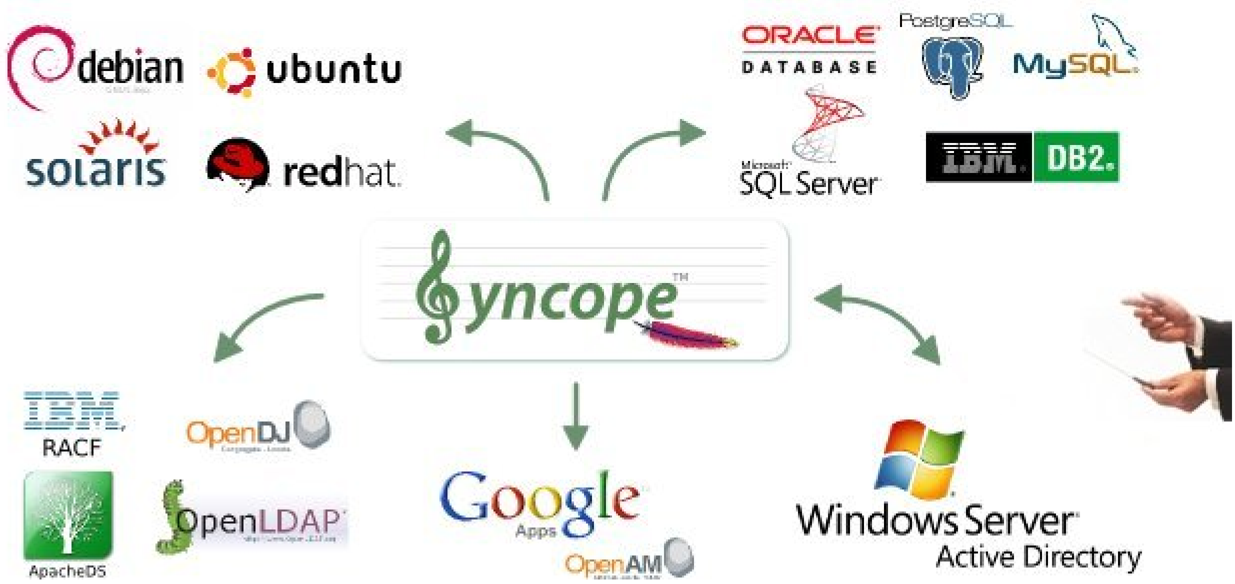


Figure 1. Apache Syncope and the external world

ConnId

Apache Syncope relies on ConnId for communication with Identity Stores; ConnId is designed to separate the implementation of an application from the dependencies of the system that the application is attempting to connect to.

ConnId is the continuation of The Identity Connectors Framework (Sun ICF), a project that used to be part of market leader Sun IdM and has since been released by Sun Microsystems as an Open Source project. This makes the connectors layer particularly reliable because most connectors have already been implemented in the framework and widely tested.

The new ConnId project, featuring contributors from several companies, provides all that is required nowadays for a modern Open Source project, including an Apache Maven driven build, artifacts and mailing lists. Additional connectors – such as for SOAP, CSV, PowerShell and Active Directory – are also provided.

Aren't Identity Stores enough?

One might suppose that a single Identity Store can solve all the identity needs inside an organization, but there are a few drawbacks with this approach:

1. Heterogeneity of systems
2. Lack of a single source of information (HR for corporate id, Groupware for mail address, …)
3. Often applications require a local user database
4. Inconsistent policies across the infrastructure
5. Lack of workflow management
6. Hidden infrastructure management cost, growing with the size of the organization

1.1.2. Provisioning Engines

The main role of *Provisioning Engines* is to keep Identity Stores synchronized as much as possible.

Some other characteristics and features provided:

- Adapt to Identity Store data and application models
- Do not require changes in Identity Stores or applications
- Build virtual unified view of identity data distributed across several Identity Stores
- Allow to define and enforce security policies
- Permit workflow definition, with transitions subject to approval
- Focused on application back-end

In brief, provisioning engines take heterogeneous Identity Stores (and business requirements) as input and build up high-level identity data management throughout what is called the **Identity Lifecycle**.

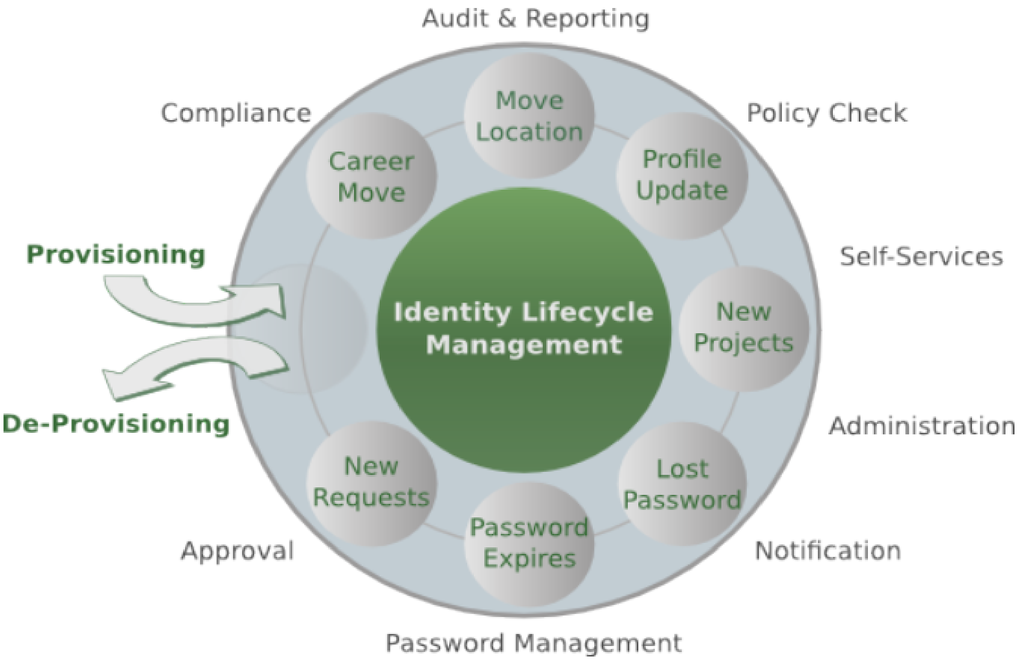


Figure 2. Identity Lifecycle

From a technology point of view, **Apache Syncope** is primarily a **Provisioning Engine**.

1.1.3. Access Managers

Access Managers focus on the application front-end, enforcing application access via authentication (how Users are let access a given system) and authorization (which capabilities each user owns on a given system).

Several practices and standards can be implemented by Access Managers:

- Single Sign-On
- OAuth
- XACML
- SAML
- OpenID Connect

1.1.4. The Complete Picture

The picture below shows a typical scenario where an organization's infrastructure is helped by identity technologies in providing secure and trusted application access to end-Users, while keeping different levels of data and processes under control for business owners, help-desk operators and system administrators.

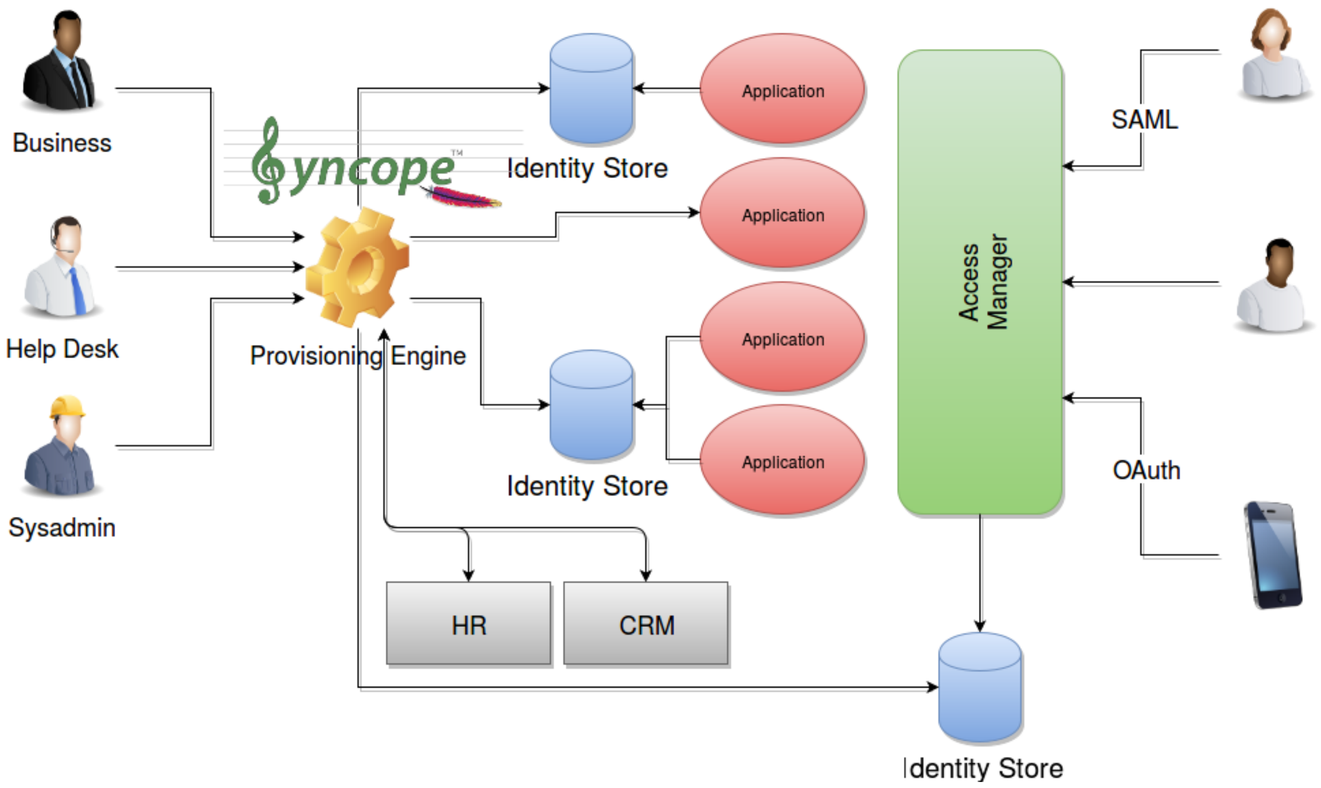


Figure 3. Identity Technologies - The Complete Picture

Chapter 2. Architecture

Apache Syncope is made of several components, which are logically summarized in the picture below.

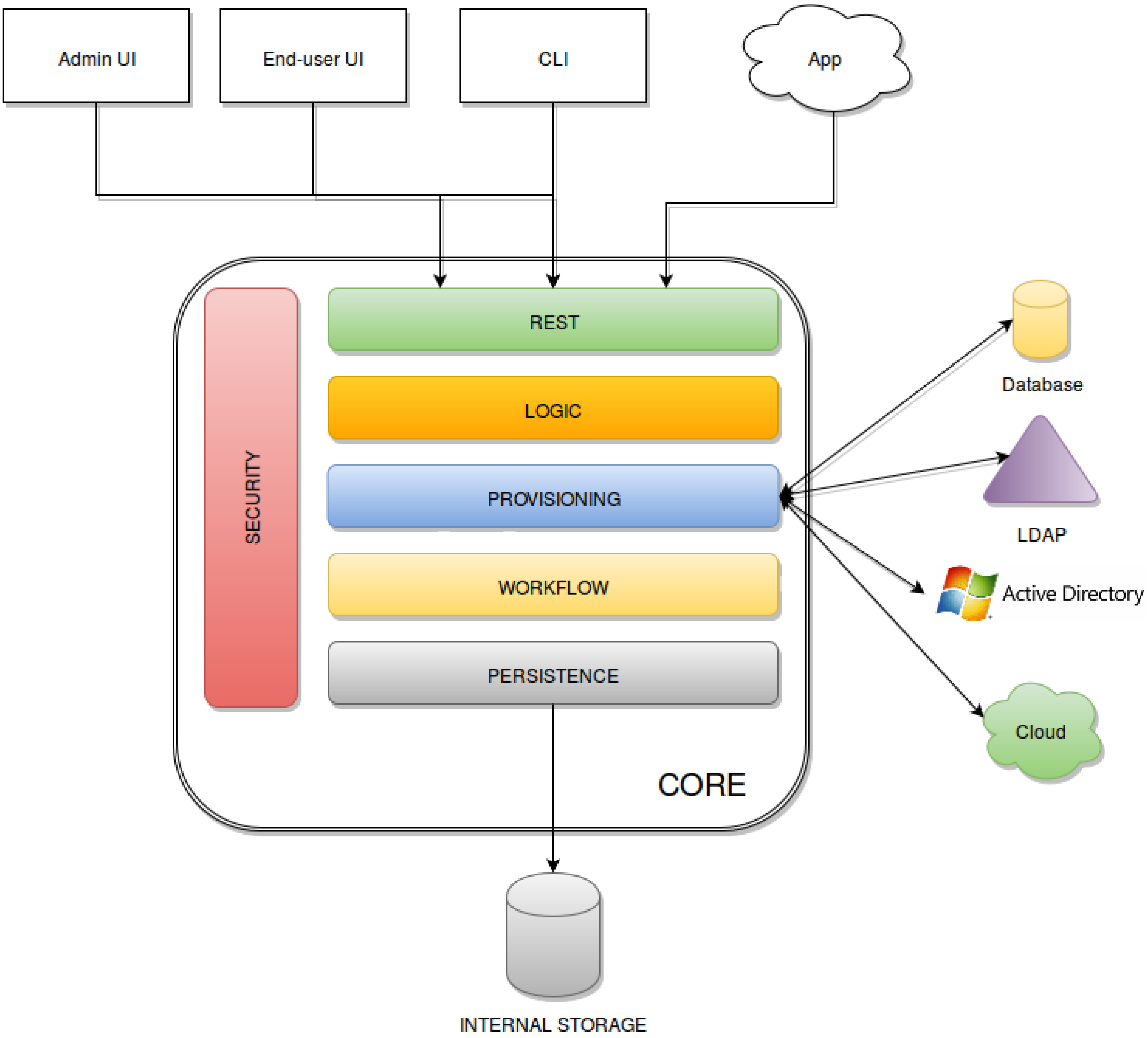


Figure 4. Architecture

2.1. Core

All the services provided by Apache Syncope are defined, elaborated and served by the ***Core***.

The Core is internally further structured into several layers, each one taking care of specific aspects of the identity management services.

2.1.1. REST

The primary way to consume Core services is the RESTful interface, which enables full access to all the features provided. This interface enables third-party applications, written in any programming language, to consume IdM services.

The rich pre-defined set of endpoints can be extended by adding new ones, which might be needed on a given Apache Syncope deployment to complement the native features with domain-specific operations.

An extension is also available, providing full Swagger features, which enables in-browser access to all the REST endpoints defined.

At a technical level, the RESTful interface is a fully-compliant JAX-RS 2.0 implementation based on Apache CXF, natively dealing both with JSON and XML payloads.

More details are available in the dedicated usage section.

2.1.2. Logic

Right below the external interface level, the overall business logic is responsible for orchestrating the other layers, by implementing the operations that can be triggered via REST services. It is also responsible for controling some additional features (notifications, reports and auditing).

2.1.3. Provisioning

The Provisioning layer is involved with managing the internal (via workflow) and external (via specific connectors) representation of Users, Groups and Any Objects.

One of the most important features provided is the mapping definition: internal data (Users, for example) representation is correlated with information available on the available Identity Stores.
Such definitions constitute the pillars of inbound (pull) and outbound (propagation / push) provisioning.

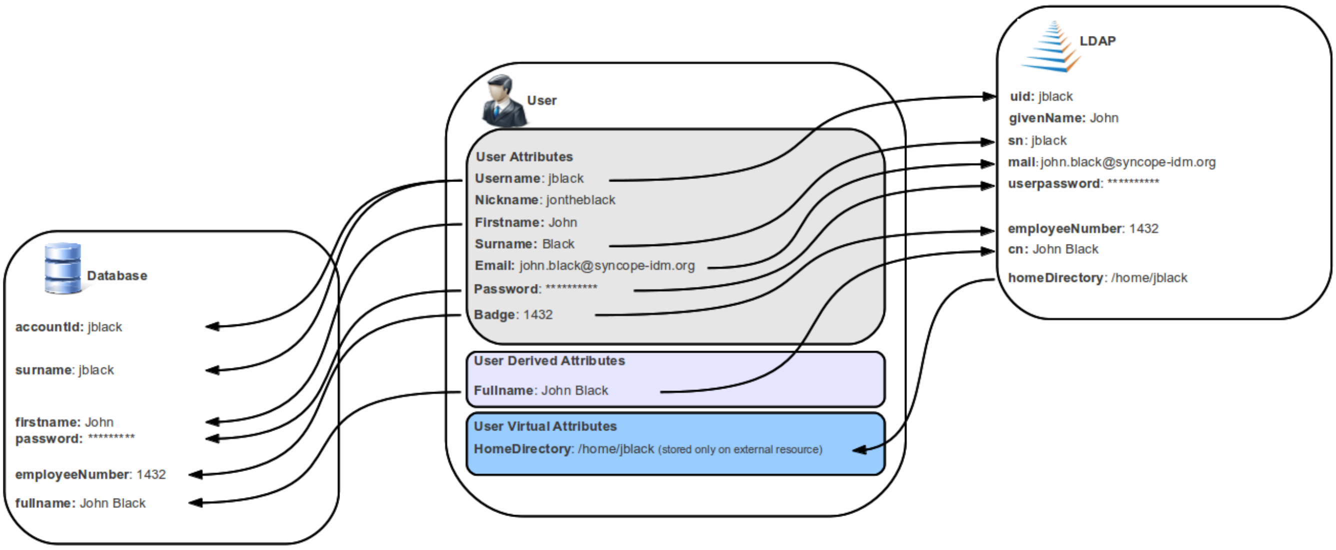


Figure 5. Internal / External Mapping

The default implementation can be sometimes tailored to meet the requirements of a specific deployment, as it is the crucial decision point for defining and enforcing the consistency and transformations between internal and external data.

In addition, an Apache Camel-based implementation is also available as an extension, which brings all the power of runtime changes and adaptation.

2.1.4. Workflow

The Workflow layer is responsible for managing the internal lifecycle of Users, Groups and Any Objects.

Besides the default engine, another engine is available based on Activiti BPM, the reference open source BPMN 2.0 implementation. It enables advanced features such as approval management and new statuses definitions. An optional web-based GUI editor is also available.

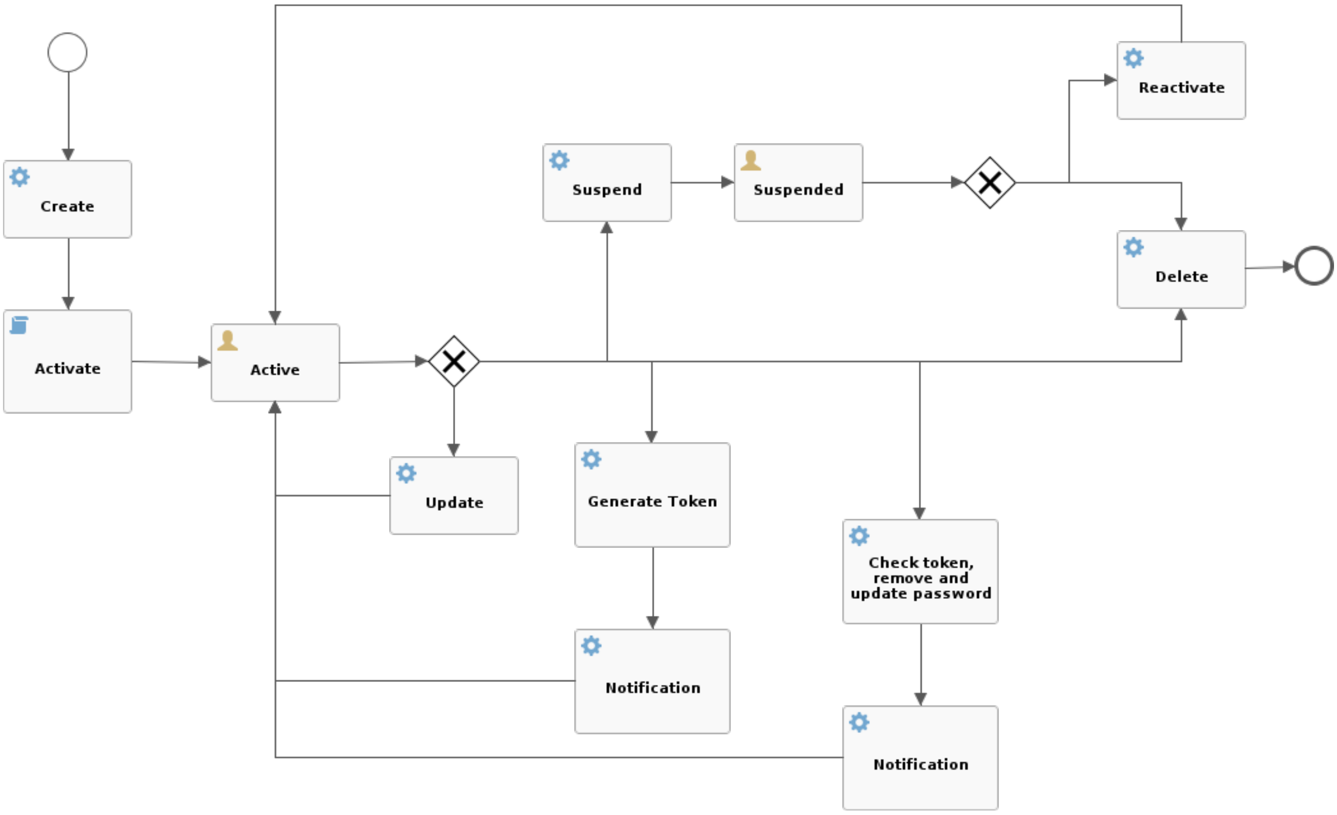


Figure 6. Default Activiti user workflow

Besides Activiti, new workflow engines - possibly integrating with third-party tools as Camunda or jBPM, can be written and plugged into specific deployments.

2.1.5. Persistence

All data (users, groups, attributes, resources, …) is internally managed at a high level using a standard JPA 2.0 approach. The data is persisted into an underlying database, referred to as ***Internal Storage***. Consistency is ensured via the comprehensive transaction management provided by the Spring Framework.

Globally, this offers the ability to easily scale up to a million entities and at the same time allows great portability with no code changes: MySQL, MariaDB, PostgreSQL, Oracle and MS SQL Server are fully supported deployment options.

Domains allow to manage data beloging to different tenants into separate database instances.

2.1.6. Security

Rather than being a separate layer, Security features are triggered throughout incoming request processing.

A fine-grained set of entitlements is defined which can be granted to administrators, thus enabling the implementation of delegated administration scenarios.

2.2. Admin UI

The Admin UI is the web-based console for configuring and administering running deployments, with full support for delegated administration.

The communication between Admin UI and Core is exclusively REST-based.

More details are available in the dedicated usage section.

2.3. End-user UI

The End-user UI is the web-based application for self-registration, self-service and password reset.

The communication between End-user UI and Core is exclusively REST-based.

2.4. CLI

The command-line interface (CLI) client is an utility tool meant for interacting with Apache Syncope deployments from shell scripts.

The communication between CLI and Core is exclusively REST-based.

More details are available in the dedicated usage section.

2.5. Third Party Applications

Third-party applications are provided full access to IdM services by leveraging the REST interface, either via the Java Client Library (the basis of Admin UI, End-user UI and CLI) or plain HTTP calls.

2.5.1. Eclipse IDE Plugin

The Eclipse IDE plugin allows remote management of notification e-mail and report templates, and constitutes an example of a Java application relying on the Client Library for interacting with the Core via REST.

2.5.2. Netbeans IDE Plugin

The Netbeans IDE plugin allows remote management of notification e-mail and report templates, and constitutes an example of a Java application relying on the Client Library for interacting with the Core via REST.

Chapter 3. Concepts

3.1. Users, Groups and Any Objects

Users, Groups and Any Objects are definitely the key entities to manage: as explained above in fact, the whole identity management concept is literally about managing identity data.

Starting with Apache Syncope 2.0, the following identities are supported:

- **Users** represent the virtual identities build up of account information fragmented across the associated external resources
- **Groups** have the dual purpose of representing entities on external resources supporting this concept (say LDAP or Active Directory) and putting together Users or Any Objects for implementing group-based provisioning, e.g. to dynamically associate Users or Any Objects to external resources
- **Any Objects** actually cover very different entities that can be modeled: printers, services, sensors, …

For each of the identities above, Apache Syncope is capable of maintaining:

1. `name` (`username`, for Users) - string value uniquely identifying a specific user, group or any object instance;
2. `password` (Users only) - hashed or encrypted value, depending on the selected `password.cipher.algorithm` - see below for details - which can be used for authentication;
3. set of attributes, with each attribute being a `(key,values)` pair where
   - `key` is a string label (e.g. `surname`);
   - `values` is a (possibly singleton) collection of data (e.g. `[Doe]` but also `[john.doe@syncope.apache.org, jdoe@gmail.com]`) ; the type of values that can be assigned to each attribute is defined via the schema matching the `key` value (e.g. *plain*, *derived* and *virtual*);
4. associations with external resources, for provisioning.

> *Which schemas can be populated for a given user / group / any object?*
>
> Each user / group / any object will be able to hold values for all schemas:
>
> 1. defined in the Any Type classes associated to their Any Type;
> 2. defined in the Any Type classes configured as ***auxiliary*** for the specific instance.

Moreover, Users and Any Objects can be part of Groups, or associated to other any objects.

Memberships and Relationships

When an user or an any object is assigned to a group, a ***membership*** is defined; the (static) members of a group benefit from type extensions.

When an user or an any object is associated to another any object, a ***relationship*** is defined, of one of available relationship types.

Static and Dynamic Memberships

Users and Any Objects are *statically* assigned to Groups when memberships are explicitly set.

With group definition, however, a condition can be expressed so that all matching Users and Any Objects are *dynamic* members of the group.
Dynamic memberships have some limitations: for example, type extensions do not apply; group-based provisioning is still effective.

3.1.1. Password Reset

When users lost their password, a feature is available to help gaining back access to Apache Syncope: password reset.

The process can be outlined as follows:

1. user asks for password reset, typically via end-user
2. user is asked to provide an answer to the security question that was selected during self-registration or self-update
3. if the expected answer is provided, a unique token with time-constrained validity is internally generated and an e-mail is sent to the configured address for the user with a link - again, typically to the end-user - containing such token value
4. user clicks on the received link and provides new password value, typically via end-user
5. user receives confirmation via e-mail

The outlined procedure requires a working e-mail configuration.

In particular:

- the first e-mail is generated from the `requestPasswordReset` notification template: hence, the token-based access link to the end-user is managed there;
- the second e-mail is generated from the `confirmPasswordReset` notification template.

The process above requires the availability of security questions that users can pick up and provide answers for.

The usage of security questions can be however disabled by setting the `passwordReset.securityQuestion` value - see below for details.

⚠ Once provided via Enduser Application, the answers to security questions are **never** reported, neither via REST or Admin UI to administrators, nor to end-users via Enduser Application.

This to avoid any information disclosure which can potentially lead attackers to reset other users' passwords.

ℹ In addition to the password reset feature, administrators can set a flag on a given user so that he / she is forced to update their password value at next login.

3.2. Type Management

In order to manage which attributes can be owned by Users, Groups and any object, and which values can be provided, Apache Syncope defines a simple yet powerful type management system, vaguely inspired by the LDAP/X.500 information model.

3.2.1. Schema

A schema instance describes the values that attributes with that schema will hold; it can be defined plain, derived or virtual.

Plain

Values for attributes with such schema types are provided during user, group or any object create / update.

When defining a plain schema, the following information must be provided:

- Type
  - `String`
  - `Long` - allows to specify a *conversion pattern* to / from string, according to DecimalFormat
  - `Double` - allows to specify a *conversion pattern* to / from string, according to DecimalFormat
  - `Boolean`
  - `Date` - allows to specify a *conversion pattern* to / from string, according to DateFormat
  - `Enum`
    - enumeration values (mandatory)
    - enumeration labels (optional, values will be used alternatively)
  - `Encrypted`
    - secret key
    - cipher algorithm
  - `Binary` - it is required to provide the declared mime type
- Validator class - (optional) Java class validating the value(s) provided for attributes, see EmailAddressValidator for reference

- Mandatory condition - JEXL expression indicating whether values for this schema must be necessarily provided or not; compared to simple boolean value, such condition allows to express complex statements like 'be mandatory only if this other attribute value is above 14', and so on
- Unique constraint - make sure that no duplicate value(s) for this schema are found
- Multivalue flag - whether single or multiple values are supported
- Read-only flag - whether value(s) for this schema are modifiable only via internal code (say workflow tasks) or can be instead provided during ordinary provisioning

Derived

Sometimes it is useful to obtain values as arbitrary combinations of other attributes' values: for example, with `firstname` and `surname` plain schemas, it is natural to think that `fullname` could be somehow defined as the concatenation of `firstname` 's and `surname` 's values, separated by a blank space.

Derived schemas are always read-only and require a JEXL expression to be specified that references plain schema types.
For the sample above, it would be

```
firstname + ' ' + surname
```

With derived attributes, values are not stored into the internal storage but calculated on request, by evaluating the related JEXL expression

Virtual

Virtual attributes are somehow linked from Identity Stores rather than stored internally.

The typical use case is when attribute values can change in the Identity Store without notice, and it is required to always have access to the most recent values that are available.

It can also be said that virtual schemas are for attributes whose ownership is not that of Syncope but of an Identity Store; the external resources for such Identity Stores are said to be the *linking resources*.

> 💡 As best practice, only attributes for which Apache Syncope retains ownership should be modeled as plain attributes; attributes for which Apache Syncope does not retain ownership should be modeled as virtual instead.

When defining a virtual schema, the following information must be provided:

- External resource - linking resource
- External attribute - attribute to be linked on the external resource
- Any Type - reference Any Type on the external resource
- Read-only flag - whether the external attribute value(s) for this schema can only be read, or

whether they can be written to as well

Virtual Attribute Cache

For performance optimization, virtual attributes are managed by an internal cache to control the actual access to the linked Identity Stores.

The internal cache implements the VirAttrCache interface, and is configurable.

Some implementations are provided by default - see table below - custom ones can be provided.

MemoryVirAttrCache	Simple fixed-size in-memory cache, with configurable time-to-live.
DisabledVirAttrCache	Pass-through cache which actually does not provide any caching: use when direct access to the Identity Store is required.

3.2.2. AnyTypeClass

Any type classes are aggregations of plain, derived and virtual schemas, provided with unique identifiers.

Classes can be assigned to Any Types and are also available as auxiliary (hence to be specified on a given user / group / any object instance) and for type extensions.

3.2.3. AnyType

Any types represent the type of identities that Apache Syncope is able to manage; besides the predefined `USER` and `GROUP`, more types can be created to model workstations, printers, folders, sensors, services, …

For all Any Types that are defined, a set of classes can be selected so that instances af a given Any Type will be enabled to populate attributes for schemas in those classes.

Example 1. Any types and attributes allowed for Users, Groups and Any Objects

Assuming that the following schemas are available:

1. plain: `firstname`, `surname`, `email`
2. derived: `fullname`
3. virtual: `enrollment`

and that the following Any Type classes are defined:

1. `minimal` - containing `firstname`, `surname` and `fullname`
2. `member` - containing `email` and `enrollment`

and that the `USER` Any Type has only `minimal` assigned, then the following Users are valid (details are simplified to increase readability):

```
{
  "key": "74cd8ece-715a-44a4-a736-e17b46c4e7e6",
  "type": "USER",
  "realm": "/",
  "username": "verdi",
  "plainAttrs": [
    {
      "schema": "surname",
      "values": [
        "Verdi"
      ]
    },
    {
      "schema": "firstname",
      "values": [
        "Giuseppe"
      ]
    }
  ],
  "derAttrs": [
    {
      "schema": "fullname",
      "values": [
        "Giuseppe Verdi"
      ]
    }
  ]
}

{
  "key": "1417acbe-cbf6-4277-9372-e75e04f97000",
  "type": "USER",
  "realm": "/",
  "username": "rossini",
  "auxClasses": [ "member" ],
  "plainAttrs": [
    {
      "schema": "surname",
      "values": [
        "Rossini"
      ]
    },
    {
      "schema": "firstname",
      "values": [
        "Gioacchino"
      ]
    },
    {
      "schema": "email",
      "values": [
        "gioacchino.rossini@syncope.apache.org"
      ]
    }
  ],
  "derAttrs": [
    {
      "schema": "fullname",
      "values": [
        "Gioacchino Rossini"
      ]
    }
  ],
  "virAttrs": [
    {
      "schema": "enrollment",
      "values": [
        "154322"
      ]
    }
  ]
}
```

3.2.4. RelationshipType

Relationships allow the creation of a link between a user and an any object, or between two Any Objects; relationship types define the available link types.

Example 2. Relationship between Any Objects (printers)

The following any object of type PRINTER contains a relationship of type neighborhood with another PRINTER (details are simplified to increase readability):

```
{
  "key": "fc6dbc3a-6c07-4965-8781-921e7401a4a5",
  "type": "PRINTER",
  "realm": "/",
  "name": "HP LJ 1300n",
  "auxClasses": [],
  "plainAttrs": [
    {
      "schema": "model",
      "values": [
        "Canon MFC8030"
      ]
    },
    {
      "schema": "location",
      "values": [
        "1st floor"
      ]
    }
  ],
  "relationships": [
    {
      "type": "neighborhood",
      "rightType": "PRINTER",
      "rightKey": "8559d14d-58c2-46eb-a2d4-a7d35161e8f8"
    }
  ]
}
```

3.2.5. Type Extensions

When a user (or an any object) is part of a group, a *membership* is defined.

It is sometimes useful to define attributes which are bound to a particular membership: if, for example, the University A and University B Groups are available, a student might have different e-mail addresses for each university. How can this be modeled?

Type extensions define a set of classes associated to a group, that can be automatically assigned to a given user (or any object) when becoming a member of such group.

Example 3. Membership with type extension

With reference to the sample above (details are simplified to increase readability):

```
{
  "key": "c9b2dec2-00a7-4855-97c0-d854842b4b24",
  "type": "USER",
  "realm": "/",
  "username": "bellini",
  "memberships": [
    {
      "type": "Membership",
      "rightType": "GROUP",
      "rightKey": "bf825fe1-7320-4a54-bd64-143b5c18ab97",
      "groupName": "University A",
      "plainAttrs": [
        {
          "schema": "email",
          "values": [
            "bellini@university_a.net"
          ]
        }
      ]
    },
    {
      "type": "Membership",
      "rightType": "GROUP",
      "rightKey": "bf825fe1-7320-4a54-bd64-143b5c18ab96",
      "groupName": "University B",
      "plainAttrs": [
        {
          "schema": "email",
          "values": [
            "bellini@university_b.net"
          ]
        }
      ]
    }
  ]
}
```

3.3. External Resources

Connector Bundles

The components able to connect to Identity Stores; not specifically bound to Apache Syncope, as they are part of the ConnId project.

Connector Instances

Instances of connector bundles, obtained by assigning values to the defined configuration properties. For instance, there is only a single `DatabaseTable` (the bundle) that can be instantiated several times, for example if there is a need to connect to different databases.

External Resources

Meant to encapsulate all information about how Apache Syncope will use connector instances for provisioning. For each entity supported by the related connector bundle (user, group, printer, services, …), mapping information can be specified.

3.3.1. Connector Instance details

When defining a connector instance, the following information must be provided:

- connector bundle - one of the several already available, or some to be made from scratch, in order to fulfill specific requirements
- pooling information
- configuration - dependening on the selected bundle, these are properties with configuration values: for example, with LDAP this means host, port, bind DN, object classes while with DBMS it would be JDBC URL, table name, etc.
- capabilities - define what operations are allowed on this connector: during provisioning, if a certain operation is invoked but the corresponding capability is not set on the related connector instance, no actual action is performed on the underlying connector; the capabilities are:
  - `AUTHENTICATE` - consent to pass-through authentication
  - `CREATE` - create objects on the underlying connector
  - `UPDATE` - update objects on the underlying connector
  - `DELETE` - delete objects on the underlying connector
  - `SEARCH` - search / read objects from the underlying connector; used during pull with `FULL RECONCILIATION` or `FILTERED RECONCILIATION` mode
  - `SYNC` - synchronize objects from the underlying connector; used during pull with `INCREMENTAL` mode

Configuration and capability override

Capabilities and individual configuration properties can be set for *override*: in this case, all the external resources using the given connector instance will have the chance to override some configuration values, or the capabilities set.

This can be useful when the same connector instance is shared among different resources, with little difference in the required configuration or capabilities.

3.3.2. External Resource details

Given a selected connector instance, the following information is required to define an external resource:

- priority - integer value, in use by the default propagation task executor
- generate random password flag - under some circumstances, a password might be mandatory but no actual value could be available: with this flag set, a random value will be generated, compliant with the defined password policy (if set)
- propagation actions - which actions shall be executed during propagation
- trace levels - control how much tracing (including logs and execution details) shall be carried over during propagation, pull and push
- configuration - see above
- capabilities - see above
- account policy - which account policy to enforce on Users, Groups and Any Objects assigned to this external resource
- password policy - which password policy to enforce on Users, Groups and Any Objects assigned to this external resource
- pull policy - which pull policy to apply during pull on this external resource
- push policy - which push policy to apply during push on this external resource

3.3.3. Mapping

The mapping between internal and external data is of crucial importance when configuring an external resource. Such information, in fact, plays a key role for provisioning.

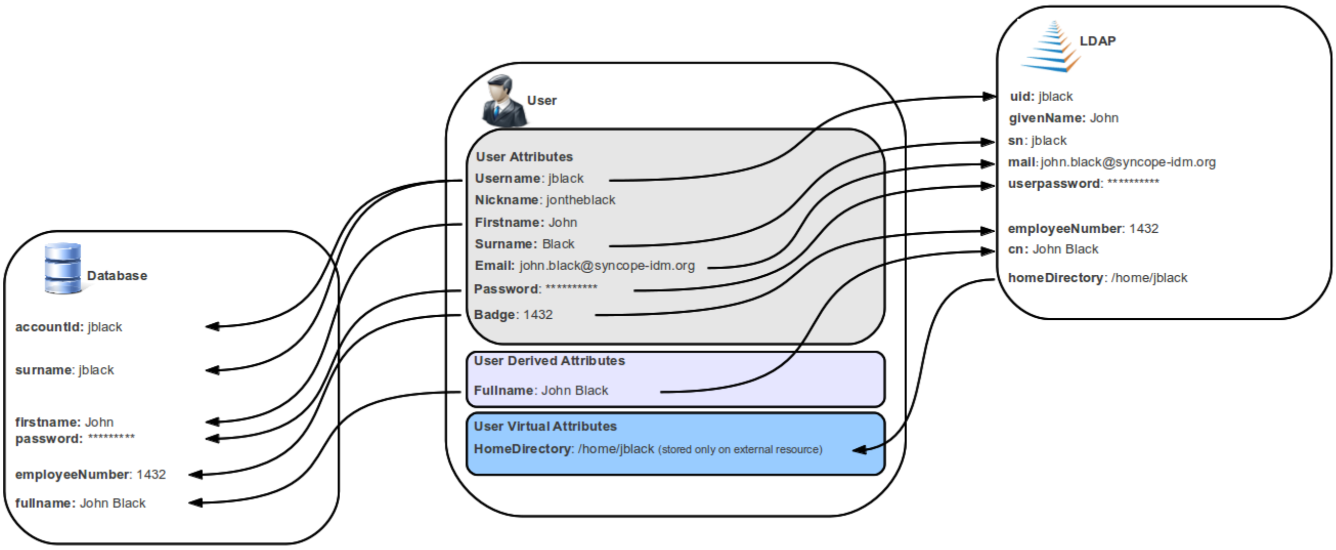


Figure 7. Sample mapping

For each of the Any Types supported by the underlying connector, a different mapping is provided.

A mapping is essentially a collection of *mapping items* describing the correspondance between an user / group / any object attribute and its counterpart on the Identity Store represented by the current external resource. Each item specifies:

- internal attribute - the schema acting as the source or destination of provisioning operations; it must be specified by an expression matching one of the following models:
  - `schema` - resolves to the attribute for the given `schema`, owned by the mapped entity (user, group, any object)
  - `groups[groupName].schema` - resolves to the attribute for the given `schema`, owned by the group with name `groupName`, if a membership for the mapped entity exists
  - `anyObjects[anyObjectName].schema` - resolves to the attribute for the given `schema`, owned by the any object with name `anyObjectName`, if a relationship with the mapped entity exists
  - `memberships[groupName].schema` - resolves to the attribute for the given `schema`, owned by the membership for group `groupName` of the mapped entity (user, any object), if such a membership exists
- external attribute - the name of the attribute on the Identity Store
- transformers - JEXL expression or Java class implementing MappingItemTransformer ; the purpose is to transform values before they are sent to or received from the underlying connector
- mandatory condition - JEXL expression indicating whether values for this mapping item must be necessarily available or not; compared to a simple boolean value, such condition allows complex statements to be expressed such as 'be mandatory only if this other attribute value is above 14', and so on
- remote key flag - should this item be considered as the key value on the Identity Store?
- password flag (Users only) - should this item be treated as the password value?
- purpose - should this item be considered for propagation / push, pull, both or none?

Besides the items documented above, some more data needs to be specified for a complete mapping:

- ConnId `objectClass` - which object class shall be used during communication with the Identity Store; predefined are `__ACCOUNT__` for Users and `__GROUP__` for Groups
- Object link - only required by some connector bundles as LDAP and Active Directory, generally specifies the model for generating the DN (distinguished name) values

Example 4. Mapping items

The following mapping item binds the mandatory internal `name` schema with the external attribute `cn` for both propagation / push and pull.

```
{
  "key": "a2bf43c8-74cb-4250-92cf-fb8889409ac1",
  "intAttrName": "name",
  "extAttrName": "cn",
  "connObjectKey": true,
  "password": false,
  "mandatoryCondition": "true",
  "purpose": "BOTH"
}
```

The following mapping item binds the optional internal `aLong` schema for the membership of the `additional` group with the external attribute `age` for propagation / push only; in addition, it specifies a JEXL expression which appends `.0` to the selected `aLong` value before sending it out to the underlying connector.

```
{
  "key": "9dde8bd5-f158-499e-9d81-3d7fcf9ea1e8",
  "intAttrName": "memberships[additional].aLong",
  "extAttrName": "age",
  "connObjectKey": false,
  "password": false,
  "mandatoryCondition": "false",
  "purpose": "PROPAGATION",
  "propagationJEXLTransformer": "value + '.0'"
}
```

3.4. Realms

Realms define a hierarchical security domain tree, primarily meant for containing Users, Groups and Any Objects.

Each realm:

1. has a unique name and a parent realm - except for the pre-defined *root realm*, which is named `/`;
2. is either a leaf or root of a sub-tree of realms;
3. is uniquely identified by the path from the root realm, e.g. `/a/b/c` identifies the sub-realm `c` in the sub-tree rooted at `b`, having in turn `a` as parent realm, directly under the root realm;
4. optionally refers to account and password policies: such policies are enforced on all Users, Groups and Any Objects in the given realm and sub-realms, unless some sub-realms define their

own policies.

If Users, Groups and Any Objects are members of a realm then they are also members of the parent realm: as a result, the root realm contains everything, and other realms can be seen as containers that split up the total number of entities into smaller pools.

This has consequences on memberships and relationships:

- A User or an Any Object can be members of Groups in the same realm or in one of the sub-realms.
- A User or an Any object can be in a relation with Any Objects in the same realm or in one of the sub-realms.

Moreover, this partition allows fine-grained control over policy enforcement and, alongside with entitlements and roles, helps to implement delegated administration.

> *Logic Templates*
>
> As with pull it is also possible to add templates to a realm.
>
> The values specified in the template are applied to entities belonging to that realm, hence this can be used as a mechanism for setting default values for attributes or external resources on entities.

3.4.1. Realm Provisioning

Provisioning can be enabled for realms: mapping information can be provided so that realms are considered during propagation, pull and push execution.

A typical use case for realm provisioning is to model an organization-like structure on Identity Stores, as with LDAP and Active Directory.

3.4.2. LogicActions

When Users, Groups or Any Objects get created, updated or deleted in a realm, custom logic can be invoked by associating the given realm with one or more implementations of the LogicActions interface.

3.5. Entitlements

Entitlements are basically strings describing the right to perform an operation.

The components in the logic layer are annotated with Spring Security to implement declarative security; in the following code snippet taken from RealmLogic , the `hasRole` expression is used together with one of the standard entitlements to restrict access only to Users owning the `REALM_LIST` entitlement.

```
@PreAuthorize("hasRole('" + StandardEntitlement.REALM_LIST + "')")
public List<RealmTO> list(final String fullPath) {
```

Entitlements are granted via roles to Users, scoped under certain realms, thus allowing delegated administration.

The set of available entitlements is statically defined - even though extensions have the ability to enlarge the initial list : this is because entitlements are the pillars of the internal security model and are not meant for external usage.

3.6. Roles

Roles map a set of entitlements to a set of realms.

Static and Dynamic Memberships

Users are *statically* assigned to roles when assignments are explicitly set.

However, a condition can be expressed in the role definition so that all matching Users are *dynamic* members of the role.

3.6.1. Delegated Administration

The idea is that any user U assigned to a role R, which provides entitlements E_1…E_n for realms Re_1…Re_k, can exercise E_i on entities (Users, Groups, Any Objects of given types, depending on E_i) under any Re_j or related sub-realms.

Example 5. Authorization

Let’s suppose that we want to implement the following scenario:

> Administrator A can create Users under realm R_5 but not under realm R_7, administrator B can update users under realm R_6 and R_8, administrator C can update Groups under realm R_8.

As by default, Apache Syncope will have defined the following entitlements, among others:

- `USER_CREATE`
- `USER_UPDATE`
- `GROUP_UPDATE`

Hence, here is how entitlements should be assigned (via roles) to administrators in order to implement the scenario above:

- Administrator A: `USER_CREATE` on R_5
- Administrator B: `USER_UPDATE` on R_6 and R_8
- Administrator C: `GROUP_UPDATE` on R_8

Group Ownership

Groups can designate a user or another group as *owner*.

The practical consequence of this setting is that Users owning a Group (either because they are directly set as owners or members of the owning group) is that they are entitled to perform all operations (create, update, delete, …) on the owned group, regardless of the Realm.

3.7. Provisioning

As described above, provisioning is actually *the* core feature provided by Apache Syncope.

Essentially, it can be seen as the process of keeping the identity data synchronized between Syncope and related external resources, according to the specifications provided by the mapping. It does this by performing create, update and delete operations onto the internal storage or external resources via connectors.

3.7.1. Overview

The picture below contains an expanded view of the core architecture, with particular reference to the components involved in the provisioning process.

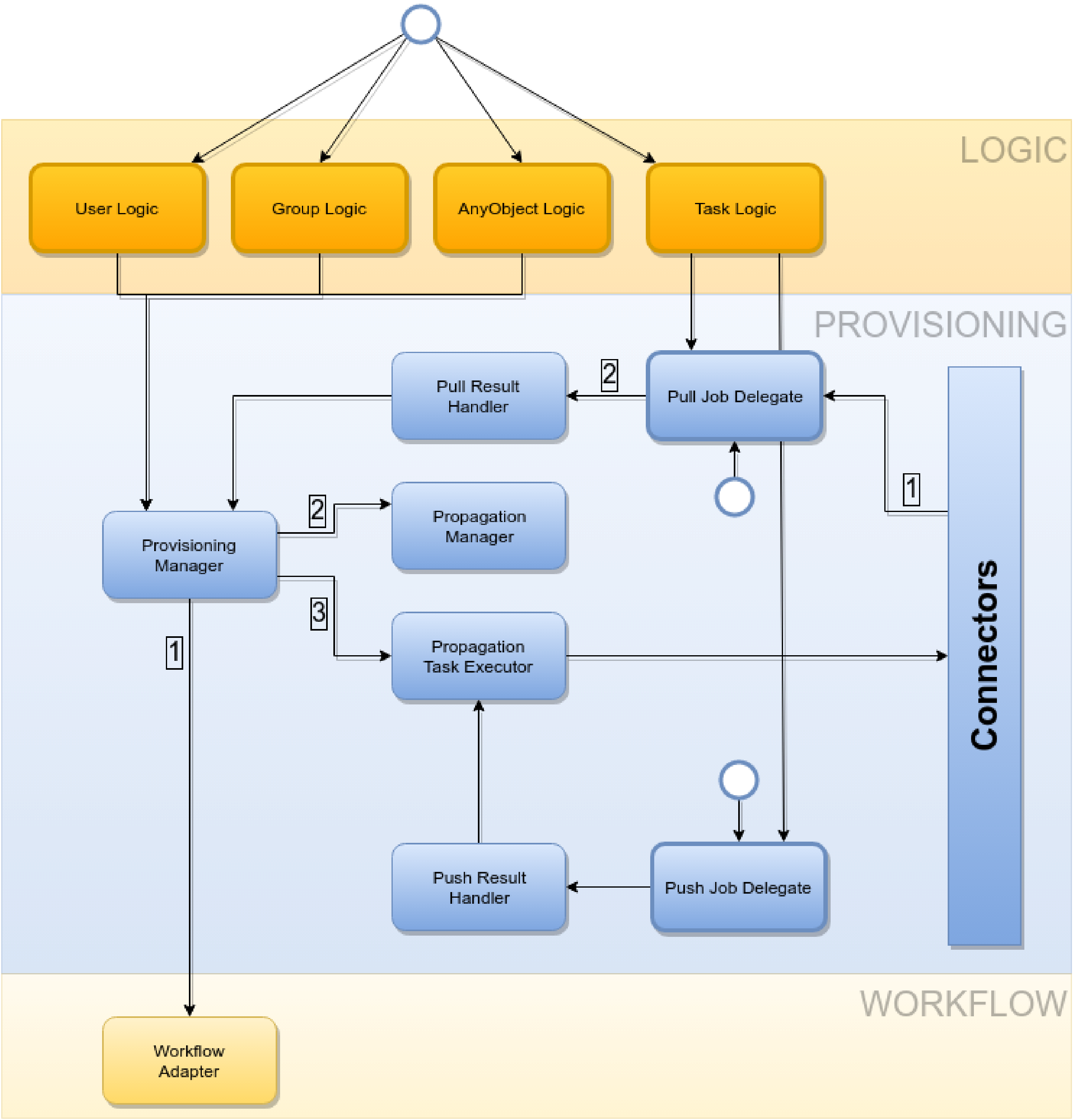


Figure 8. Provisioning flow

The provisioning operations can be initiated in several different ways:

- by creating, updating or deleting Users, Groups or Any Objects via REST (thus involving the underlying logic layer)
- by requesting execution of pull or push tasks via REST
- by triggering periodic pull or push task executions

Provisioning Managers

The provisioning operations are defined by the provisioning manager interfaces:

- UserProvisioningManager
- GroupProvisioningManager
- AnyObjectProvisioningManager

Default implementations are available:

- DefaultUserProvisioningManager
- DefaultGroupProvisioningManager
- DefaultAnyObjectProvisioningManager

An Apache Camel-based implementation is available as extension.

3.7.2. Propagation

Whenever a change is performed via REST on Users, Groups or Any Objects:

1. a set of propagation tasks is generated, one for each associated external resource;
2. the generated propagation tasks are executed, e.g. the corresponding operations (create, update or delete) are sent out, via connectors, to the configured Identity Stores; the tasks can be saved for later re-execution.

Which external resources?

Depending on the entity being created / updated / deleted, different external resources are taken into account by the propagation process:

- **Group**: only the external resources directly assigned
- **User**: the external resources directly assigned plus the ones assigned to Groups configured for the User
- **Any Object**: the external resources directly assigned plus the ones assigned to Groups configured for the Any Object

By default, the propagation process is controlled by the PriorityPropagationTaskExecutor, which implements the following logic:

- sort the tasks according to the related resource's *priority*, then execute sequentially
- tasks for resources with no priority are executed afterwards, concurrently
- the execution of a given set of tasks is halted (and global failure is reported) whenever the first sequential task fails
- status and eventual error message (in case of no resource priority) can be saved for reporting, in the case where the related external resource was configured with adequate tracing
- minimize the set of operations to be actually performed onto the Identity Store by attempting to

read the external object corresponding to the internal entity and comparing with the modifications provided

Different implementations of the PropagationTaskExecutor interface can be provided, in case the required behavior does not fit into the provided implementation.

PropagationActions

The propagation process can be decorated with custom logic to be invoked around task execution, by associating external resources to one or more implementations of the PropagationActions interface.

Some examples are included by default, see table below.

LDAPMembershipPropagationActions	If a User is associated with a Group in Syncope, keep the corresponding User as a amember of the corresponding Group in LDAP.
LDAPPasswordPropagationActions	If no password value was already provided in the propagation task, sends out the internal password hash value to LDAP; the cipher algorithm associated with the password must match the value of `passwordHashAlgorithm` for the LDAP connector bundle.
DBPasswordPropagationActions	If no password value was already provided in the propagation task, sends out the internal password hash value to DBMS; the cipher algorithm associated with the password must match the value of `Password cipher algorithm` for the DatabaseTable connector bundle.

3.7.3. Pull

Pull is the mechanism used to acquire identity data from Identity Stores; for each external resource, one or more pull tasks can be defined, run and scheduled for period execution.

Pull task execution involves querying the external resource and then processing each entity in an isolated transaction; a retrieved entity can be:

1. *matching* if a corresponding internal entity was found, according to the pull policy set for the enclosing external resource;
2. *unmatching* otherwise.

Once this has been assessed, entities are processed according to the matching / unmatching rules specified for the pull task: by default, unmatching entities get created internally, and matching entities are updated.

Matching Rules

- `IGNORE`: do not perform any action;
- `UPDATE`: update matching entity;
- `DEPROVISION`: delete external entity;
- `UNLINK`: remove association with external resource, without performing any (de-)provisioning operation;
- `LINK`: associate with external resource, without performing any (de-)provisioning operation;
- `UNASSIGN`: unlink and delete.

Unmatching Rules

- `IGNORE`: do not perform any action;
- `UNLINK`: do not perform any action;
- `ASSIGN`: create internally, assign the external resource;
- `PROVISION`: create internally, do not assign the external resource.

Pull Mode

The Identity Store can be queried in different ways, depending on the *pull mode* that is specified:

FULL RECONCILIATION

The complete list of entities available is processed.

FILTERED RECONCILIATION

The subset matching the filter (provided by the selected implementation of ReconciliationFilterBuilder) of all available entities is processed.

INCREMENTAL

Only the actual modifications performed since the last pull task execution are considered. This mode requires the underlying connector bundle to implement the ConnId `SYNC` operation - only some of the available bundles match this condition.
This is the only mode which allows pulling delete events, which may end up causing the removal of internal entities.

Pull Templates

With every pull task it is possible to add a template for each defined Any Type.

As the values specified in the template are applied to pulled entities, this can be used as mechanism for setting default values for attributes or external resources on entities.

A typical use case is, when pulling Users from the external resource `R`, to automatically assign `R` so that every further modification in Apache Syncope to such Users will be propagated back to `R`.

PullActions

The pull process can be decorated with custom logic to be invoked around task execution, by associating pull tasks to one or more implementations of the PullActions interface.

Some examples are included by default, see the table below.

LDAPMembershipPullActions	If a User is associated with a Group in LDAP, keep the corresponding User as a member of the corresponding Group in Syncope.
LDAPPasswordPullActions	Import hashed password values from LDAP; the cipher algorithm associated with the password must match the value of `passwordHashAlgorithm` for the LDAP connector bundle.
DBPasswordPullActions	Import hashed password values from DBMS; the cipher algorithm associated with the password must match the value of `Password cipher algorithm` for the DatabaseTable connector bundle.

3.7.4. Push

With push, the matching set of internal entities can be sent to Identity Stores - mainly for (re)initialization purposes; for each external resource, one or more push tasks can be defined, run and scheduled for period execution.

Push task execution involves querying the internal storage and then processing each entity in an isolated transaction; an internal entity can be:

1. *matching* if a corresponding remote entity was found, according to the push policy set for the enclosing external resource;
2. *unmatching* otherwise.

Once this has been assessed, entities are processed according to the matching / unmatching rules specified for the push task: by default, unmatching entities are pushed to Identity Stores, and matching entities are updated.

Matching Rules

- `IGNORE`: do not perform any action;
- `UPDATE`: update matching entity;
- `DEPROVISION`: delete internal entity;
- `UNLINK`: remove association with external resource, without performing any (de-)provisioning operation;
- `LINK`: associate with external resource, without performing any (de-)provisioning operation;
- `UNASSIGN`: unlink and delete.

Unmatching Rules

- `IGNORE`: do not perform any action;
- `UNLINK`: remove association with external resource, without performing any (de-)provisioning operation;
- `ASSIGN`: create externally, assign the external resource;
- `PROVISION`: create externally, do not assign the external resource.

PushActions

The push process can be decorated with custom logic to be invoked around task execution, by associating push tasks to one or more implementations of the PushActions interface.

3.8. Policies

Policies control different aspects of the provisioning process. They can be used to fine-tune and adapt the overall mechanism to the particularities of the specific domain in which a given Apache Syncope deployment is running.

Policy Composition

When defining policies and associating them with different realms and resources, it is common to observe that several policies of the same type have to be enforced on the same user, group or any object.

In such cases, Apache Syncope transparently composes all of the candidate policies and obtains a single applicable policy which contains all the conditions of the composing policies; this process, however, is not guaranteed to be successful, as different policies of the same type might provide conflicting clauses.

3.8.1. Account

Account policies allow the imposition of constraints on username values, and are involved in the authentication process.

> When set for realm R, an account policy is enforced on all Users of R and sub-realms.
>
> When set for resource R, an account policy is enforced on all Users that have R assigned.

When defining an account policy, the following information must be provided:

- max authentication attempts - how many times Users are allowed to fail authentication before getting suspended
- propagate suspension - when suspended as a consequence of too many authentication failures, should Users also be suspended on associated resources or not?
- pass-through resources - which external resources are involved with pass-through authentication
- rules - set of account rules to evaluate with the current policy

Account Rules

Account rules define constraints to apply to username values.

The default account rule (enforced by DefaultAccountRule and configurable via DefaultAccountRuleConf) contains the following controls:

- maximum length - the maximum length to allow; `0` means no limit set;
- minimum length - the minimum length to allow; `0` means no limit set;
- pattern - Java regular expression pattern to match; `NULL` means no match is attempted;
- all uppercase - are lowercase characters allowed?
- all lowercase - are uppercase characters allowed?
- words not permitted - list of words that cannot be present, even as a substring;
- schemas not permitted - list of schemas whose values cannot be present, even as a substring;
- prefixes not permitted - list of strings that cannot be present as a prefix;
- suffixes not permitted - list of strings that cannot be present as a suffix.

> Writing custom account rules means:
>
> 1. providing configuration parameters in an implementation of AccountRuleConf
> 2. implementing enforcement in an implementation of AccountRule annotated via @AccountRuleConfClass referring to the configuration class

Pass-through Authentication

During user authentication, if the resulting applicable account policy defines pass-through resources, the provided credentials are verified first against the internal storage, then against each configured external resource (provided that the underlying connector instance has the `AUTHENTICATE` capability set): the first check that succeeds will successfully authenticate the user.

This feature allows, for example, to reuse credentials contained in Identity Stores (without extracting them), instead of storing password values in the internal storage. It also facilitates implementing authentication chains.

3.8.2. Password

Password policies allow the imposition of constraints on password values.

> When set for realm R, a password policy is enforced on all Users of R and sub-realms.
>
> When set for resource R, a password policy is enforced on all Users that have R assigned.

When defining a password policy, the following information must be provided:

- allow null password - whether a password is mandatory for Users or not
- history length - how many values shall be considered in the history
- rules - set of password rules to evaluate with the current policy

Password Rules

Password rules define constraints to apply to password values.

The default password rule (enforced by DefaultPasswordRule and configurable via DefaultPasswordRuleConf) contains the following controls:

- maximum length - the maximum length to allow; `0` means no limit set;
- minimum length - the minimum length to allow; `0` means no limit set;
- non-alphanumeric required
- alphanumeric required
- digit required
- lowercase required
- uppercase required
- must start with digit
- must not start with digit
- must end with digit
- must not end with digit

- must start with alphanumeric
- must start with non-alphanumeric
- must not start with alphanumeric
- must not start with non-alphanumeric
- must end with alphanumeric
- must end with non-alphanumeric
- must not end with alphanumeric
- must not end with non-alphanumeric
- username allowed - whether a username value can be used
- words not permitted - list of words that cannot be present, even as a substring;
- schemas not permitted - list of schemas whose values cannot be present, even as a substring;
- prefixes not permitted - list of strings that cannot be present as a prefix;
- suffixes not permitted - list of strings that cannot be present as a suffix.

Writing custom password rules means:

1. providing configuration parameters in an implementation of PasswordRuleConf
2. implementing enforcement in an implementation of PasswordRule annotated via @PasswordRuleConfClass referring to the configuration class

3.8.3. Pull

Pull policies are evaluated during the execution of pull tasks and are meant to:

1. help match existing Users, Groups and Any Objects during pull, thus generating update events (rather than create)
2. determine which action shall be taken in case such match is not unique (e.g. what to do if the same external account can be mapped to two distinct Users in Apache Syncope?)

When set for resource R, a pull policy is enforced on all Users, Groups and Any Objects pulled from R.

When defining a pull policy, the following information must be provided:

- conflict resolution action
  - `IGNORE` - do nothing
  - `FIRSTMATCH` - pull first matching object only
  - `LASTMATCH` - pull last matching object only
  - `ALL` - pull all matching objects
- rules - set of correlation rules to evaluate with the current policy; for each defined Any Type, a

different rule is required

Pull Correlation Rules

Pull correlation rules define how to match objects received from connector instances with existing Users, Groups or Any Objects.

The default implementation attempts to match entities on the basis of the values of the provided plain attributes, according to the available mapping.

> Custom pull correlation rules can be provided by implementing the PullCorrelationRule interface.

3.8.4. Push

Push policies are evaluated during the execution of push tasks.

> When set for resource R, a push policy is enforced on all Users, Groups and Any Objects pushed to R.

3.9. Workflow

Workflow manages the internal identity lifecycle by defining statuses and transitions that every user, group or any object in Apache Syncope will traverse. A workflow instance is started once identities get created, and shut down when they are removed.

Workflow is triggered during the provisioning process as the first step in creating, updating or deleting identities into the internal storage.

> *Workflow Adapters*
>
> The workflow features are defined by the workflow adapter interfaces:
>
> - UserWorkflowAdapter
> - GroupWorkflowAdapter
> - AnyObjectWorkflowAdapter
>
> Default implementations are available:
>
> - DefaultUserWorkflowAdapter
> - DefaultGroupWorkflowAdapter
> - DefaultAnyObjectWorkflowAdapter
>
> Custom adapters can be provided by implementing the related interfaces, also as bridges towards third-party tools as Camunda or jBPM.

3.9.1. Activiti User Workflow Adapter

An advanced adapter is provided for Users, based on Activiti BPM, one of reference open source BPMN 2.0 implementations.

The ActivitiUserWorkflowAdapter is bootstrapped from userWorkflow.bpmn20.xml and presents several advantages and more features, if compared to the default user adapter:

1. Besides mandatory statuses, which are modeled as BPMN `userTask` instances, more can be freely added at runtime, provided that adequate transitions and conditions are also inserted; more details about available BPMN constructs are available in the Activiti User Guide.
   Additional statuses and transitions allow the internal processes of Apache Syncope to better adapt to suit organizational flows.
2. Custom logic can be injected into the workflow process by providing BPMN `serviceTask` instances.
3. Activiti forms are used for implementing approval.
4. The Activiti Modeler can be enabled in the admin console, thus allowing web-based graphical modeling of the workflow definition.

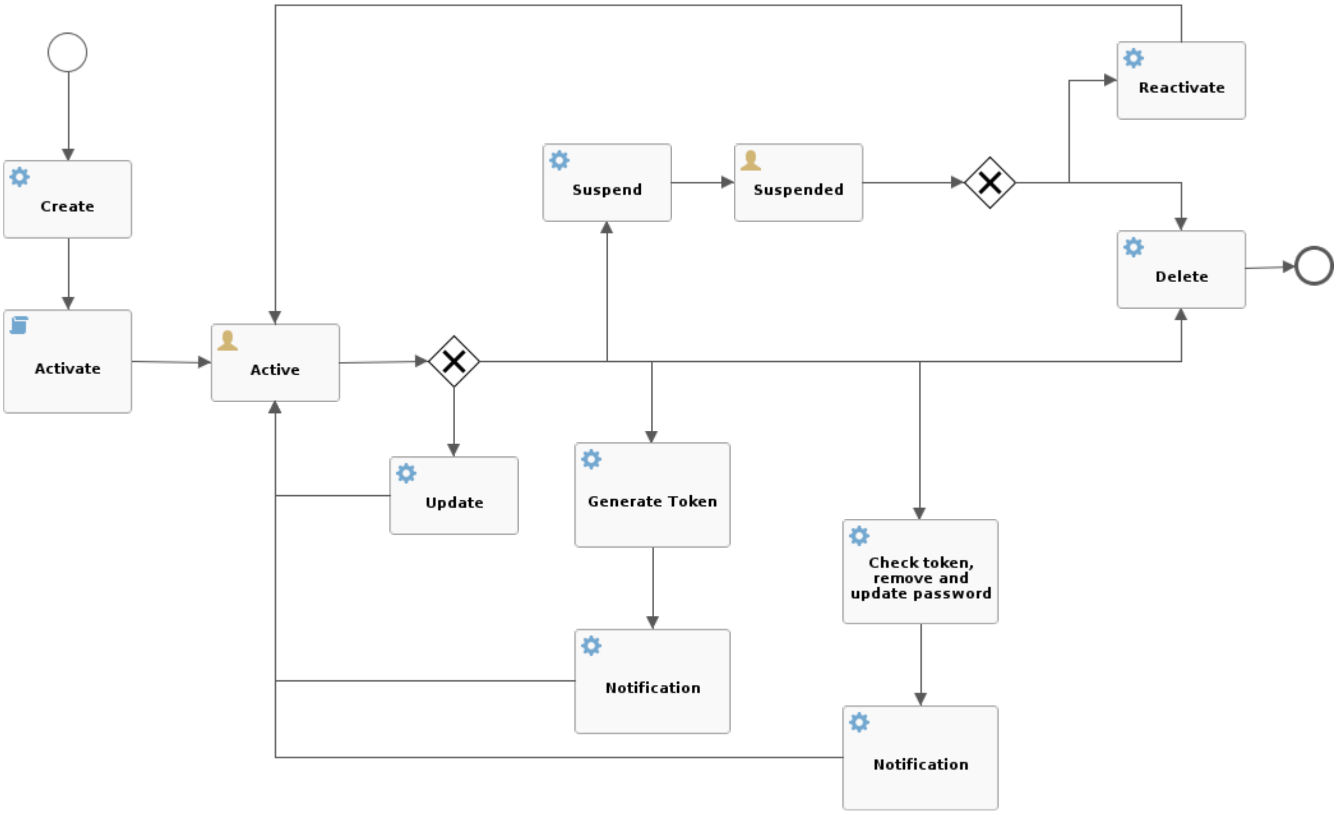


Figure 9. Default Activiti user workflow

Approval

Every transition in the Activiti user workflow definition can be subjected to approval.

The underlying idea is that some kind of self-modifications (group memberships, external resource assignments, …) might not be allowed to 'plain' Users, as there could be conditions which require management approval. Managers could also be asked to complete the information provided before the requested operation is finished.

In order to define an approval form, a dedicated BPMN `userTask` needs to be defined, following the rules established for Activiti forms.

> *What is required for administrators to manage approval?*
>
> The following conditions must be met, for an User `U` to act as administrator for approval:
>
> 1. `U` must own the following entitlements, for all the required realms:
>   a. `WORKFLOW_FORM_CLAIM`
>   b. `WORKFLOW_FORM_LIST`
>   c. `WORKFLOW_FORM_READ`
>   d. `WORKFLOW_FORM_SUBMIT`
>   e. `USER_READ`
> 2. The BPMN `userTask` must either indicate `U` among `candidateUsers` or at least one of the groups assigned to `U` among `candidateGroups`, as required by Activiti's task assignment rules
>
> The special super-user `admin` is entitled to manage all approvals, even those not specifying any `candidateUsers` or `candidateGroups`.

Example 6. Approving self-registration

The snippet below shows how to define an approval form in XML; the same operation can be performed via the Activiti Modeler.

```
<userTask id="createApproval" name="Create approval"
          activiti:candidateGroups="managingDirector"
activiti:formKey="createApproval"> ①
  <extensionElements>
    <activiti:formProperty id="username" name="Username" type="string"
                           expression="${user.username}" writable="false"/> ②
    <activiti:formProperty id="approve" name="Approve?" type="boolean"
                           variable="approve" required="true"/> ③
    <activiti:formProperty id="rejectReason" name="Reason for rejecting"
type="string"
                           variable="rejectReason"/>
  </extensionElements>
</userTask>
```

① `formKey` and `id` must be unique across the workflow definition, `name` is displayed by the admin console; `candidateGroups` and `candidateUsers` might be defined, even both, to indicate which Groups or Users should be managing these approvals; if none are specified, only `admin` is entitled to manage such approval

② `expression` will be evaluated against the current requesting `user` (as workflow variable) and related properties; read-only form input can be defined by setting `writable="false"`

③ exporting approval inputs into workflow variables is possible via the `variable` attribute; required form input can be defined by setting `required="true"`

Once the form is defined, any modification subject to that approval will be manageable via the admin console, according to the following flow (the actual operations on the admin console for the sample above are reported below):

1. administrator A sees the new approval notifications
2. administrator A claims the approval and is then allowed to manage it
3. administrator A reviews the updated user, with ongoing modification applied (no actual modification performed yet)
4. administrator A can approve or reject such modification

Identity Recertification

Recertification, also referred to as Attestation, is the process of periodically reviewing all the identities and granted accesses, within a given organization.

> Recertification is the traditional and most effective countermeasure against *privilege creep*, that is the rentention of accounts during time even when the user change his role, or even his job.
>
> Recertification is a good practice, recommended and often required by international or organizational standards.

Typically, every entity within an Identity Manager needs to be periodically recertified:

- **Identities**: Is the user still valid for the organization?
- **Accounts**: Is the user still using this account?
- **Group Memberships / Entitlements**: Does the user require this functionality?

A basic custom task implementing identity recertification is provided with the IdentityRecertification class, which can be used as base to implement more complex recertification processes.

Essentially, this task goes through all the available users and checks whether they where already recertified in the past `identity.recertification.day.interval` days - see below for details; when needed, it triggers a specific approval.

The delegated administrator for approving recertifications can be configured by modifying the "Recertification Request" task in the workflow definition.

Example 7. Certifiers are members of the `managingDirector` group

```
<userTask id="recertificationRequest" name="Recertification Request"
          activiti:formKey="recertify"
          activiti:candidateGroups="managingDirector">
```

Example 8. Certifier is the manager defined in the user's `lastRecertificator` attribute

```
<userTask id="recertificationRequest" name="Recertification Request"
          activiti:formKey="recertify"
          activiti:candidateUser="${user.lastRecertificator}">
```

3.9.2. Flowable User Workflow Adapter

Starting with Apache Syncope 2.0.3, another advanced adapter is provided for Users, based on Flowable, one of reference open source BPMN 2.0 implementations.

Since Flowable was forked from Activiti BPM, everything stated above about Activiti BPM can be applied.

3.10. Notifications

Apache Syncope can be instructed to send out notification e-mails when certain events occur.

Every notification generates one or more notification tasks, holding the actual e-mails to be sent. The tasks are ordinarily scheduled for execution according to the value provided for `notificationjob.cronExpression` - see below for details - and can be saved for later re-execution.

When defining a notification, the following information must be provided:

- notification template - template for e-mail generation
- sender - e-mail address appearing in the `From` field of the generated e-mail(s)
- subject - text used as e-mail subject
- recipient e-mail attribute - which user attribute shall be considered as e-mail address for delivery (as users might in principle have different e-mail attributes)
- recipient(s) - the actual e-mail recipient(s) which can be specified either as:
  - list of static e-mail addresses
  - matching condition to be applied to available users
  - Java class implementing the NotificationRecipientsProvider interface
- notification event(s) - event(s) triggering the enclosing notification
- about - the condition matching Users, Groups or Any Objects which are evaluated for the specified events; for users, the matching entities can be also considered as additional recipients
- trace level - control how much tracing (including logs and execution details) shall be carried over during execution of the generated notification tasks

3.10.1. Notification Events

Notification (and Audit) events are essentially a means of identifying the invocation of specific methods within the Core, in line with *join points* in the Aspect Oriented Programming (AOP).

An event is identified by the following five coordinates:

1. type - which can be one of
   - `LOGIC`
   - `TASK`
   - `PROPAGATION`
   - `PULL`
   - `PUSH`
   - `CUSTOM`
2. category - the possible values depend on the selected type: for `LOGIC` the Logic components available, for `TASK` the various Custom Tasks configured, for `PROPAGATION`, `PULL` and `PUSH` the defined Any Types
3. subcategory - completes category with external resource name, when selecting `PROPAGATION`, `PULL`

or `PUSH`

4. event type - the final identification of the event; depends on the other coordinates
5. success or failure - whether the current event shall be considered in case of success or failure

The admin console provides tooling to assist with the specification of valid events.

An event is uniquely identified by a string of the following form:

```
[type]:[category]:[subcategory]:[event type]:[SUCCESS|FAILURE]
```

Some samples:

- `[PushTask]:[group]:[resource-db-scripted]:[matchingrule_deprovision]:[SUCCESS]`
successful Group push to the external resource `resource-db-scripted`, when deprovisioning matching entities
- `[LOGIC]:[RealmLogic]:[]:[create]:[FAILURE]`
unsuccessful Realm creation
- `[CUSTOM]:[]:[]:[unexpected identification]:[SUCCESS]`
successful execution of the event identified by the `unexpected identification` string

Custom events can be used to trigger notifications from non-predefined joint points, as BPMN `userTask` instances within the Activiti User Workflow Adapter, PropagationActions, PushActions, PullActions or other custom code.

3.10.2. Notification Templates

A notification template is defined as a pair of JEXL expressions, to be used respectively for plaintext and HTML e-mails, and is available for selection in the notification specification.

Notification templates can be easily managed either via the admin console, the Eclipse IDE Plugin or the Netbeans IDE Plugin.

The full power of JEXL expressions - see reference and some examples - is available.
For example, the `user` variable, an instance of UserTO with actual value matching the *about* condition as introduced above, can be used.

Example 9. Plaintext notification template

```
Hi ${user.plainAttrMap["firstname"].values[0]}
${user.plainAttrMap["surname"].values[0]},
  welcome to Syncope!

Your username is ${user.username}.
Your email address is ${user.plainAttrMap["email"].values[0]}.

Best regards.
```

Example 10. HTML notification template

```
<html>
  <body>
    <h3>Hi ${user.plainAttrMap["firstname"].values[0]}
${user.plainAttrMap["surname"].values[0]},
      welcome to Syncope!</h3>
    <p>Your username is ${user.username}.<br/>
    Your email address is ${user.plainAttrMap["email"].values[0]}.</p>
    <p>Best regards.</p>
  </body>
</html>
```

3.11. Tasks

Tasks control the effective operations that are ongoing in the Core.

Whilst tasks define what and how to perform, they are supposed to be run by some entity (depending on the actual task type, see below for details); their execution result can be saved for later examination.

3.11.1. Propagation

A propagation task encapsulates all the information that is required - according to the defined mapping - to create, update or delete a given User, Group or Any Object, to / from a certain Identity Store:

- operation - `CREATE`, `UPDATE` or `DELETE`
- connObjectKey - value for ConnId unique identifier on the Identity Store
- oldConnObjectKey - the former unique identifier on the Identity Store: bears value only during updates involving the unique identifier
- attributes - set of ConnId attributes built upon internal identity data and configured mapping
- resource - related external resource

- objectClass - ConnId object class
- entity - reference to the internal identity: User, Group or Any Object

> Propagation tasks are automatically generated via the PropagationManager, executed (by default) via the PriorityPropagationTaskExecutor during the propagation process, and are permanently saved - for later re-execution or for examining the execution details - depending on the trace levels set on the related external resource.

3.11.2. Pull

Pull tasks are required to define and trigger the pull process from Identity Stores.

When defining a pull task, the following information must be provided:

- related external resource
- chosen pull mode
- destination Realm - where entities selected for creation are going to be placed
- whether creation, update or deletion on internal storage are allowed or not
- whether to synchronize the status information from the related identity store
- selected matching and unmatching rules
- optional pull action(s)
- entity templates
- scheduling information:
  - when to start
  - cron expression

> Pull tasks are executed, either upon request or due to a schedule, via the PullJobDelegate during the pull process, and are permanently saved - for later re-execution or for examining the execution details - depending on the trace level set on the related external resource.

> *DryRun*
>
> It is possible to simulate the execution of a pull (or push) task without performing any actual modification by selecting the *DryRun* option. The execution results will be still available for examination.

3.11.3. Push

Push tasks are required to define and trigger the push process to Identity Stores.

When defining a push task, the following information must be provided:

- related external resource

- filter information for selecting which internal entities will be pushed onto the identity store
- whether creation, update or deletion on the identity store are allowed or not
- whether to synchronize the status information with internal storage
- selected matching and unmatching rules
- optional push action(s)
- scheduling information:
  - when to start
  - cron expression

Push tasks are executed, either upon request or due to a schedule, via the PushJobDelegate during the push process, and are permanently saved - for later re-execution or for examining the execution details - depending on the trace level set on the related external resource.

3.11.4. Notification

A notification task encapsulates all the information that is required to send out a notification e-mail, according to the specification provided in a given notification:

- entity - reference to the internal identity - User, Group or Any Object - the notification task refers to
- sender e-mail address
- e-mail subject
- effective e-mail recipient(s)
- e-mail body as plaintext and / or HTML

Notification tasks are automatically generated via the NotificationManager, executed via the NotificationJob and are permanently saved - for later re-execution or for examining the execution details - depending on the trace level set on the related notification.

3.11.5. Custom

Custom tasks allow for the injection of logic into the Core in the area of execution and scheduling.

When defining a custom task, the following information must be provided:

- job delegate class: Java class extending AbstractSchedTaskJobDelegate providing the custom logic to execute
- scheduling information:
  - when to start
  - cron expression

Custom tasks are ideal for implementing periodic checks or clean-up operations, possibly in coordination with other components; some examples:

- move users from "pending delete" to "deleted" status 15 days after they reached the "pending delete" status (requires interaction with Activiti User Workflow Adapter)
- send out notification e-mails to users whose password is about to expire on an Identity Store
- disable all users not logging into the system for the past 6 months

3.12. Reports

Reports are a powerful tool to extract, filter and format relevant information from a running Apache Syncope deployment, for a wide range of purposes: from business to DevOps.

A report is essentially defined by a template and a sequence of reportlets, where the latter is responsible for extracting the required information and the former defines how execution results will be presented, in the various available formats.

Reports can be executed upon request or scheduled: execution results can be downloaded as:

- XML
- HTML
- PDF
- RTF
- CSV

3.12.1. Report Templates

A report template is defined as a triple of XSLT documents, distinguished by their target format:

- `FO` - transforms the given report result as XSL-FO, which will be then made available as PDF and RTF
- `HTML` - outputs the given report result as HTML
- `CSV` - outputs the given report result as CSV

Report templates can be easily managed either via the admin console, the Eclipse IDE Plugin or the Netbeans IDE Plugin.

3.12.2. Reportlets

Reportlets are the building blocks of reports.

Each reportlet is composed by:

- a Java class extending AbstractReportlet and implementing the information extraction logic and generating an XML stream as result

- a Java class extending AbstractReportletConf and embedding the configuration options that can be tuned when incorporating a given reportlet into a report; when properly annotated, such options are manageable via the admin console

Some reportlets are available by default (and briefly presented below) either for direct usage or for acting as a reference when building new reportlets for specific Apache Syncope deployments.

Static Reportlet

Defined by StaticReportlet and StaticReportletConf, it is essentially a handy way to inject static values (of various types) into a report.

User and Group Reportles

Defined by UserReportlet and UserReportletConf, it can be used to report various information about Users available in the internal storage, their attributes, memberships and relationships, external resources and so on.

A similar reportlet is also available for Groups, defined by GroupReportlet and GroupReportletConf.

Reconciliation Reportlet

Defined by ReconciliationReportlet and ReconciliationReportletConf, it provides the global reconciliation status for all Users, Groups and Any Objects available in the internal storage, e.g. whether such entities are available on all Identity Stores matching the assigned external resources and, if so, whether the mapped attributes feature the expected values.

An instance of reconciliation reportlet is run by default from the admin console's dashboard, and results are available as a widget.

Audit Reportlet

Defined by AuditReportlet and AuditReportletConf, it is mostly a sample reportlet showing how to extract data produced by Audit.

3.13. Audit

The audit feature allows to capture events occurring within the Core and to log relevant information about them as entries into the `SYNCOPEAUDIT` table of the internal storage.

Once events are reported in the table above, they can be used as input for external tools.

An example of how audit entries can be extracted for reporting is shown by the Audit Reportlet.

3.13.1. Audit Events

The information provided for notification events is also valid for audit events, including examples - except for the admin console tooling, which is naturally distinct.

3.14. Domains

Domains are built to facilitate multitenancy.

Domains allow the physical separation of all data managed by Apache Syncope, by storing the data for different domains into different database instances. Therefore, Apache Syncope can facilitate Users, Groups, Any Objects, External Resources, Policies, Tasks, etc. from different domains (e.g. tenants) in a single core instance.

By default, a single `Master` domain is defined, which also bears the configuration for additional domains.

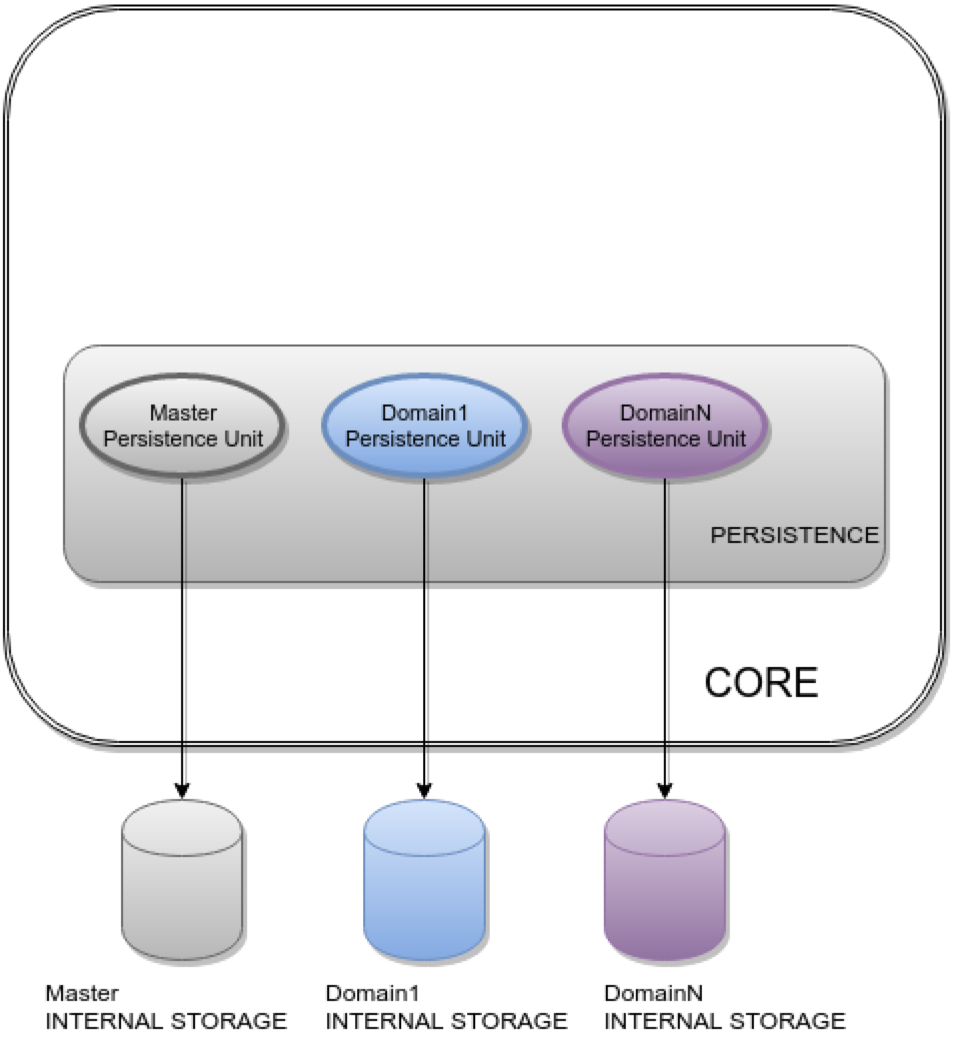


Figure 10. Domains

Each domain's persistence unit can be configured to work with one of the supported DBMSes: `Master` can be on MySQL, `Domain1` on PostgreSQL, `DomainN` on Oracle and so on.

3.15. Extensions

The *vanilla* Apache Syncope deployment can be optional enriched with useful features via an Extension, instead of bloating every single deployment with unneeded libraries and configurations.

With reference to architecture, an extension might add a REST endpoint, manage the persistence of additional entities, extend the security mechanisms, tweak the provisioning layer, add features to the Admin UI or the End-user UI, or even bring all such things together.

Extensions are available from different sources:

1. as Maven artifacts published from the Apache Syncope codebase, part of the official releases - this is the case of the ones detailed below;
2. as Maven artifacts published by third parties;
3. as part of a given deployment source code, as explained in the following.

3.15.1. Apache Camel Provisioning Manager

This extension delegates the provisioning process execution to a set of Apache Camel routes.

The pre-loaded routes can be dynamically changed at runtime via REST or admin console, and modifications are immediately made available for processing.

For example, on creating a new user, you may wish to send an email to an administrator; or if a user is reactivated, you may wish to reactivate the user's home page on a web server.
All these things and more are possible using the myriad of components that are available to be used in Apache Camel routes.

> *Extension Sources*
>
> The source code of this extension is available from the Apache Syncope source tree .

3.15.2. Swagger

This extension enables Swagger UI as web interface for dealing with Apache Syncope RESTful services.

Once installed, Swagger UI is available at

```
protocol://host:port/syncope/swagger/
```

where `protocol`, `host` and `port` reflect your Java EE container installation.

> *Extension Sources*
>
> The source code of this extension is available from the Apache Syncope source tree .

3.15.3. SAML 2.0 Service Provider

This extension can be leveraged to provide SAML 2.0-based Single Sign-On access to the Admin UI, the End-user UI or any other Java EE application dealing with the Core.

Once installed, one or more Identity Providers can be imported from their metadata. For each Identity Provider, it is to configure which one of the attributes - returned as part of the assertion containing the attribute statements - is going to be used by Syncope to match the internal users.

Extension Sources

The source code of this extension is available from the Apache Syncope source tree .

This extension adds features to all components and layers that are available, and can be taken as reference when creating new extensions.

3.15.4. Elasticsearch

This extension requires the latest JDK 8 that is available.

This extension provides an alternate internal search engine for Users, Groups and Any Objects, requiring an external Elasticsearch cluster.

As search operations are central for different aspects of the provisioning process, the global performances are expected to improve when using this extension.

Extension Sources

The source code of this extension is available from the Apache Syncope source tree .

Chapter 4. Working with Apache Syncope

Before proceeding, please ensure that you have access to a running Apache Syncope deployment. You can take a look at the Apache Syncope Getting Started Guide to check system requirements and to choose among the various options for obtaining Apache Syncope.

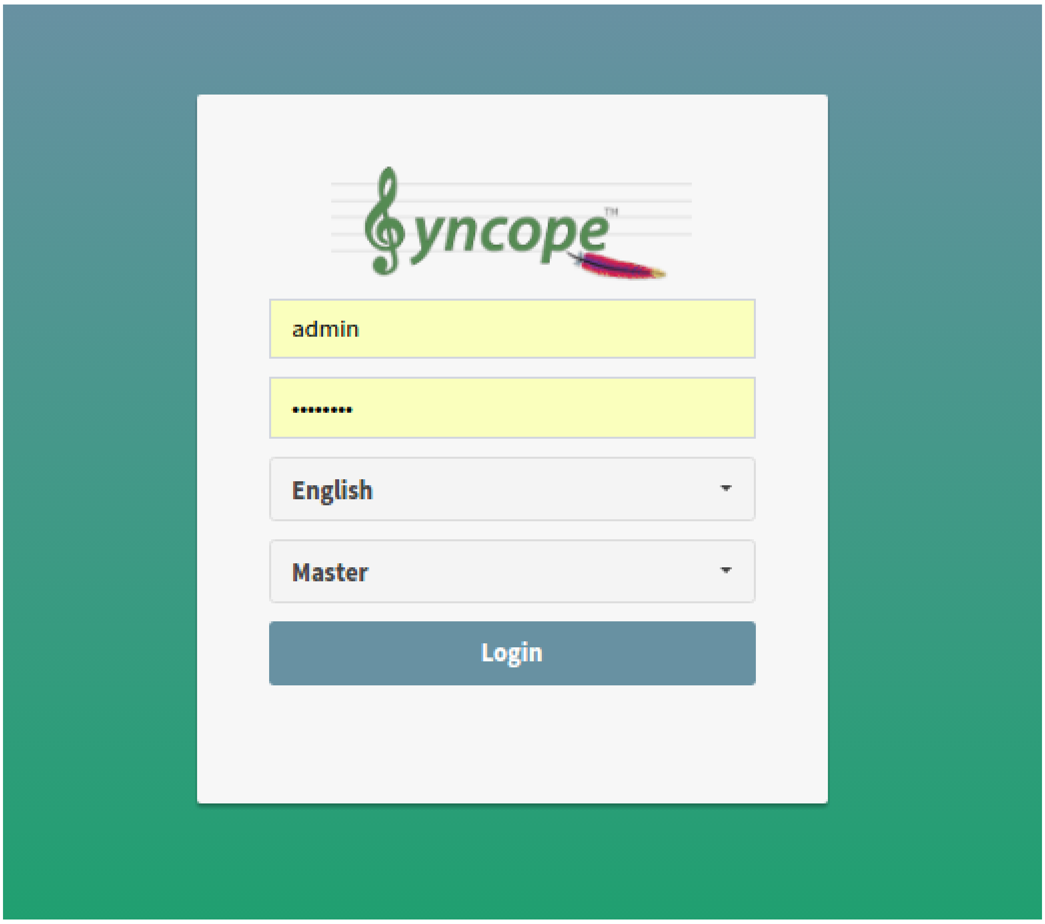
4.1. Admin Console

Once the Java EE container has initialized, the admin console can be accessed at:

```
protocol://host:port/syncope-console/
```

where `protocol`, `host` and `port` reflect your Java EE container installation.

You should be greeted by the following web page.



admin

••••••••

English

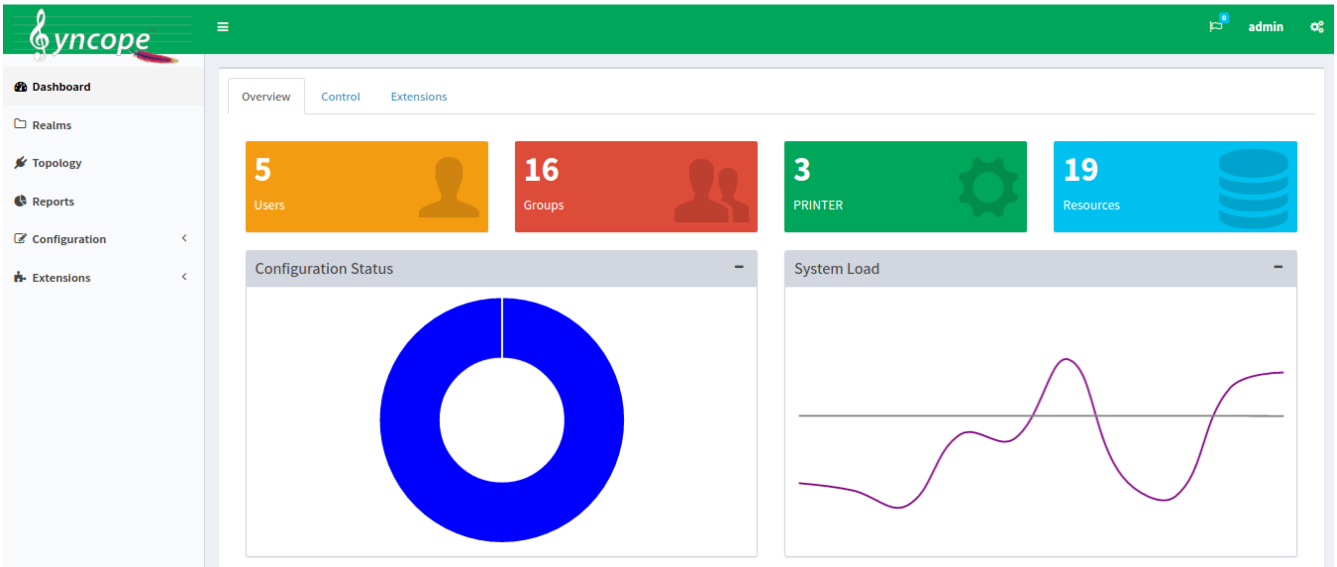
Master

You can use the default admin credentials to login.

4.1.1. Pages

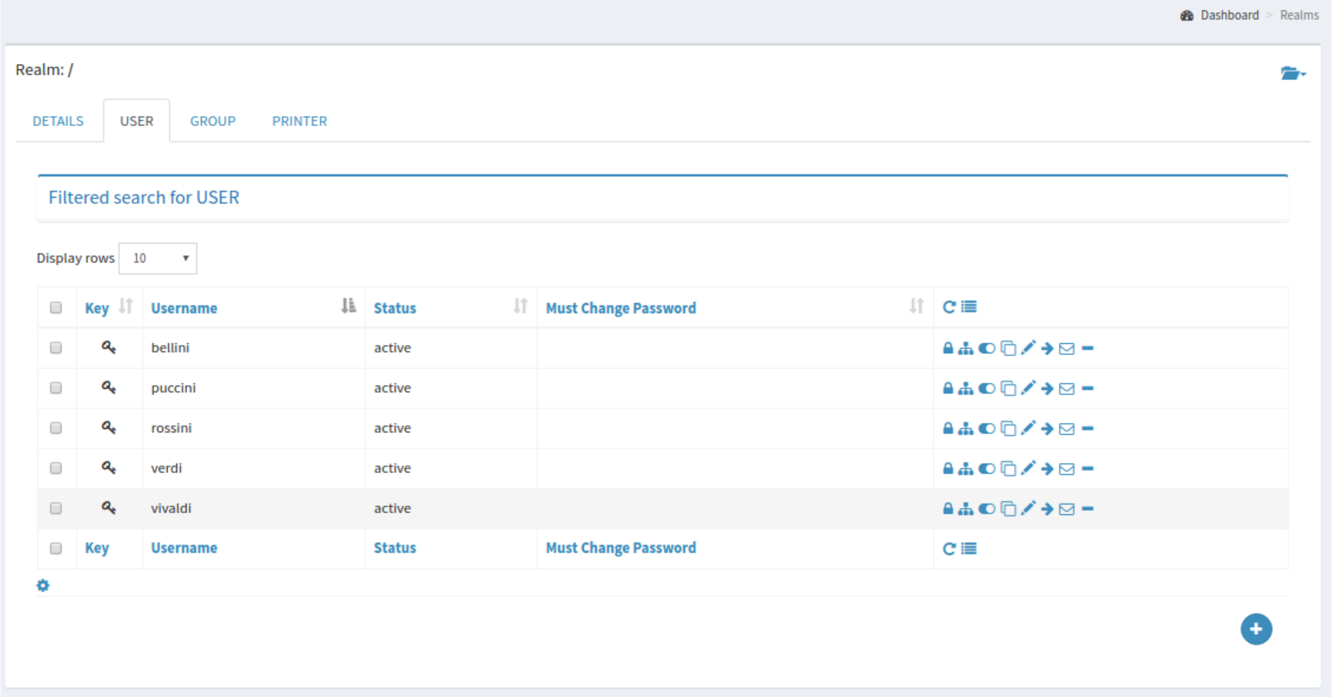
Dashboard

The dashboard provides an overall view of the current state of the Apache Syncope deployment. It consists of various widgets and tabs that show the different metrics and details of each component that is available.



Realms

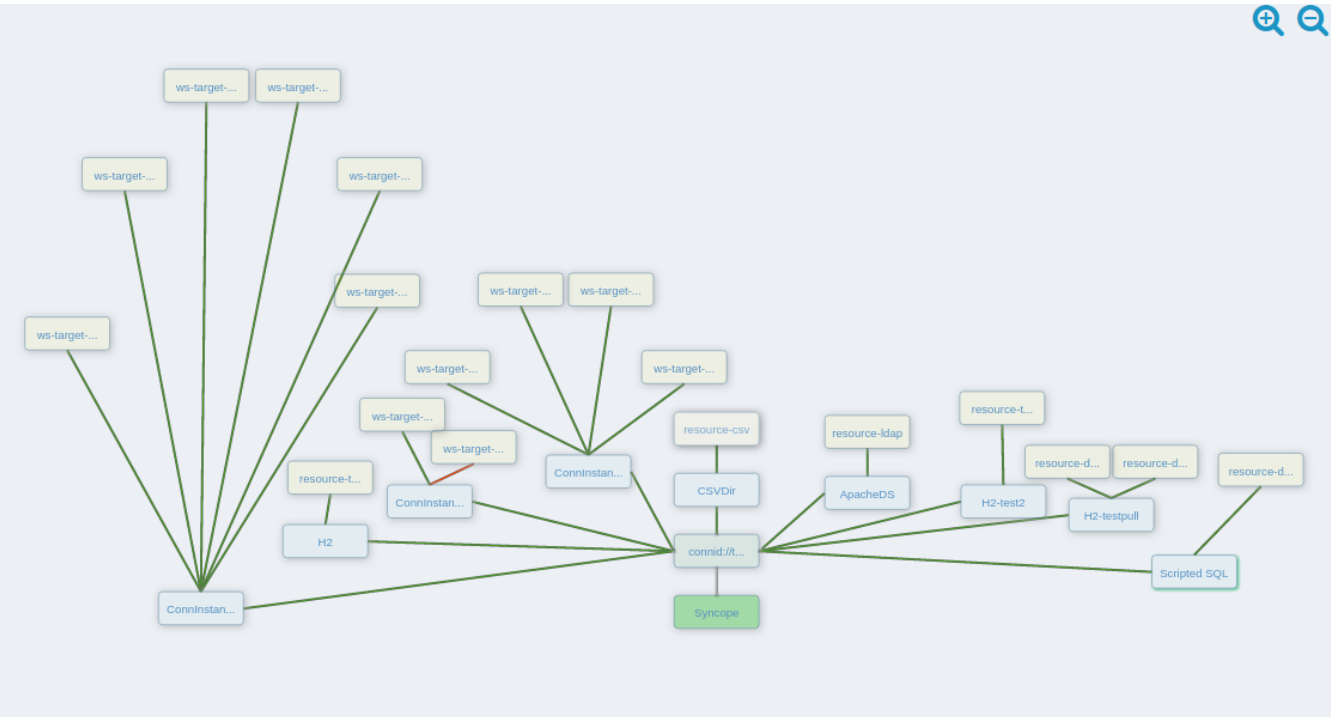
The realms page provides the designated administators with the power to manage Realms as well as Users, Groups and Any Objects, for all Any Types that are defined.



□	Key	Username	Status	Must Change Password
□	🔑	bellini	active	
□	🔑	puccini	active	
□	🔑	rossini	active	
□	🔑	verdi	active	
□	🔑	vivaldi	active	
□	Key	Username	Status	Must Change Password

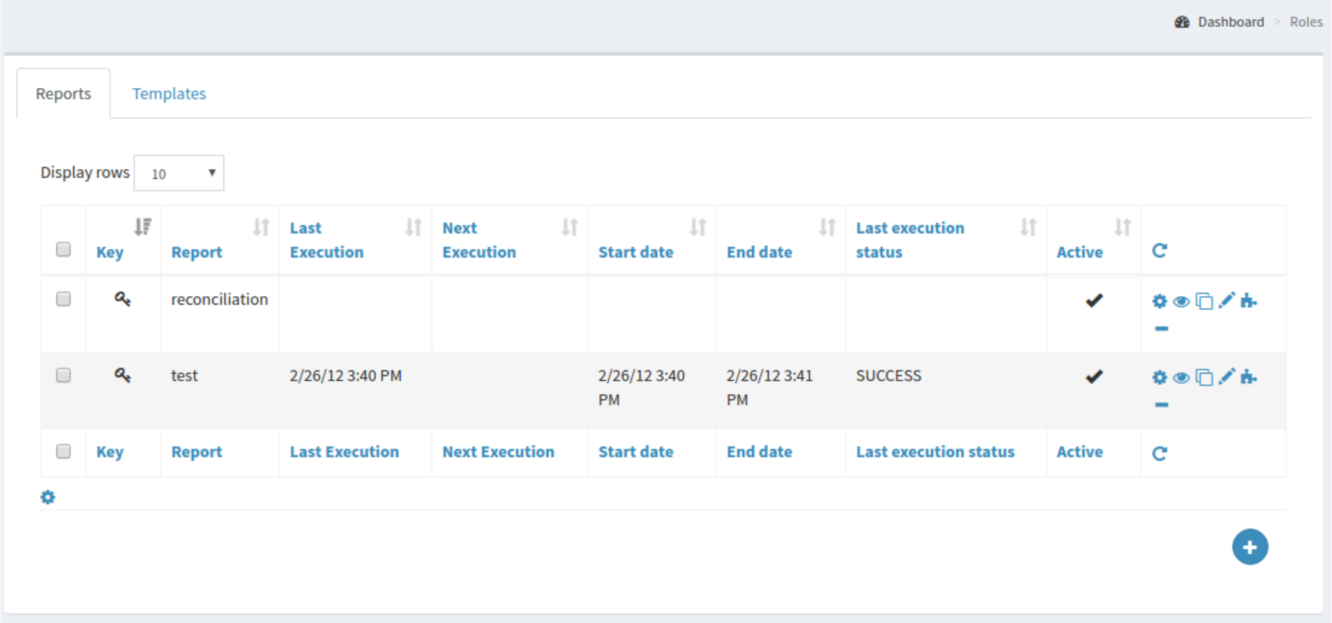
Topology

The topology page provides a mapped view of the connectors and external resources that are available and configured in the given deployment.
Different actions are available when clicking on the various nodes.



Reports

The reports page presents the designated administators with the list of reports configured on the given deployment.
This page also allows the administators to create and edit report templates.



☐	Key	Report	Last Execution	Next Execution	Start date	End date	Last execution status	Active	
☐		reconciliation						✓	
☐		test	2/26/12 3:40 PM		2/26/12 3:40 PM	2/26/12 3:41 PM	SUCCESS	✓	
☐	Key	Report	Last Execution	Next Execution	Start date	End date	Last execution status	Active	

Configuration

The configuration pages allow the designated administators to customize the given deployment to fit the needs of the organization.

Audit

Controls the configuration of the auditing features.

Logs

The logging levels available can be dynamically adjusted; for example, the admin can set it to display only the errors of `io.swagger`, in which case the warning and information logs will not be reported.

Notifications

Gives access to the notification management.
This page also allows the administators to create and edit notification templates.

Parameters

Presents the administrators with the list of defined configuration parameters used in the given deployment such as `token.expireTime` and `password.cipher.algorithm`. These can be edited to further customize the deployment.
New parameters can also be added, for use with custom code.

Policies

Allows the administrators to manage account, password and pull policies.

Roles

Displays and provides editing functionality for roles.

Security Questions

The administrators can use this page to define a set of security questions which the users can choose from when managing their own profile, to allow them to recover their account in case of a forgotten password.

Approval

The images below refer to the self-registration approval sample and to the typical approval flow as explained above.

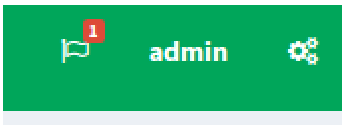
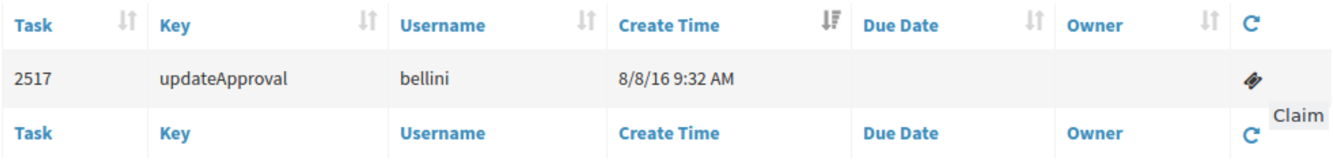
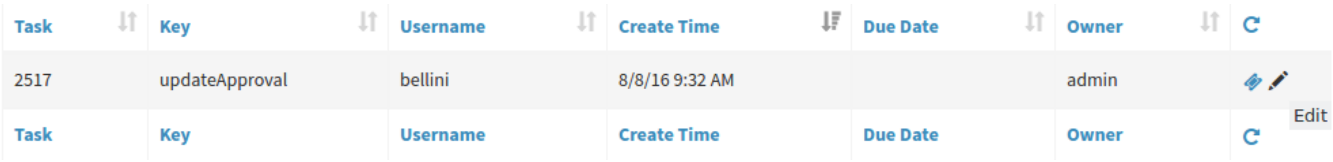


Figure 11. Approval notification



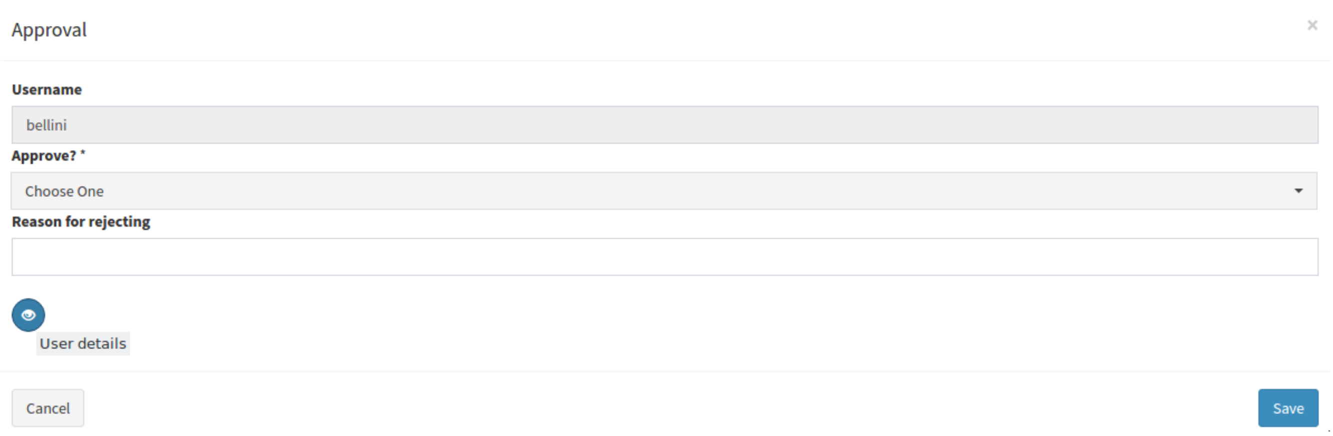
Task	Key	Username	Create Time	Due Date	Owner	
2517	updateApproval	bellini	8/8/16 9:32 AM			Claim
Task	Key	Username	Create Time	Due Date	Owner	

Figure 12. Claiming an approval



Task	Key	Username	Create Time	Due Date	Owner	
2517	updateApproval	bellini	8/8/16 9:32 AM		admin	Edit
Task	Key	Username	Create Time	Due Date	Owner	

Figure 13. Managing an approval



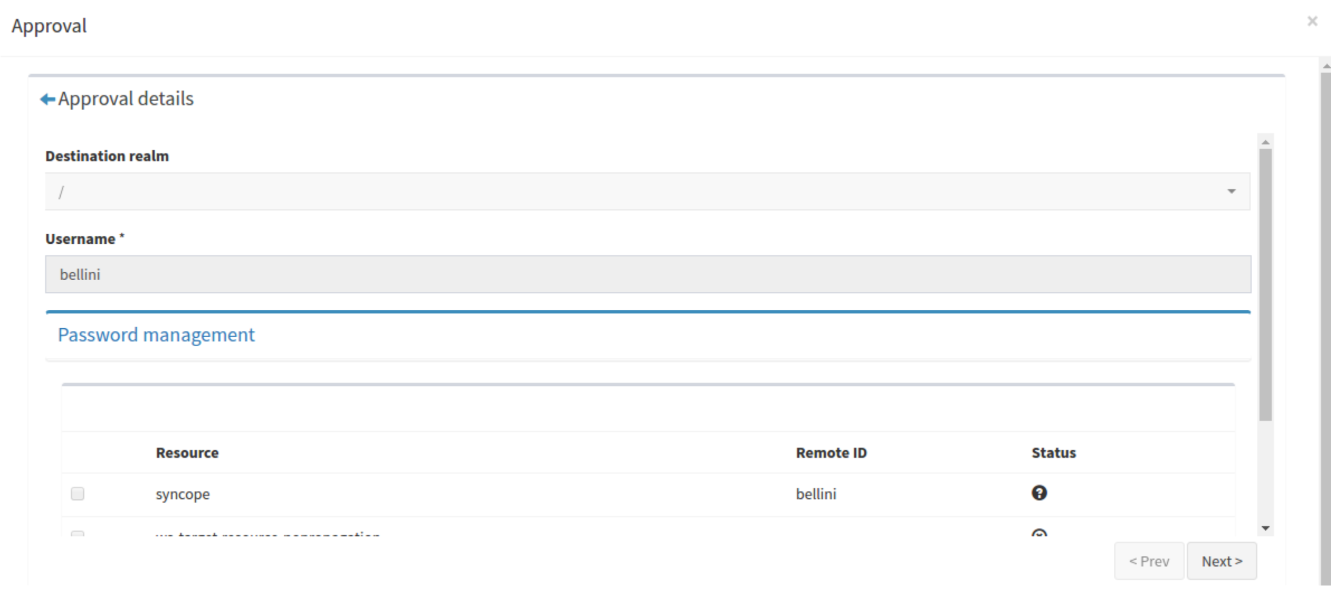
Approval

Username: bellini

Approve? *: Choose One

Reason for rejecting:

Figure 14. Approval form



Approval

Approval details

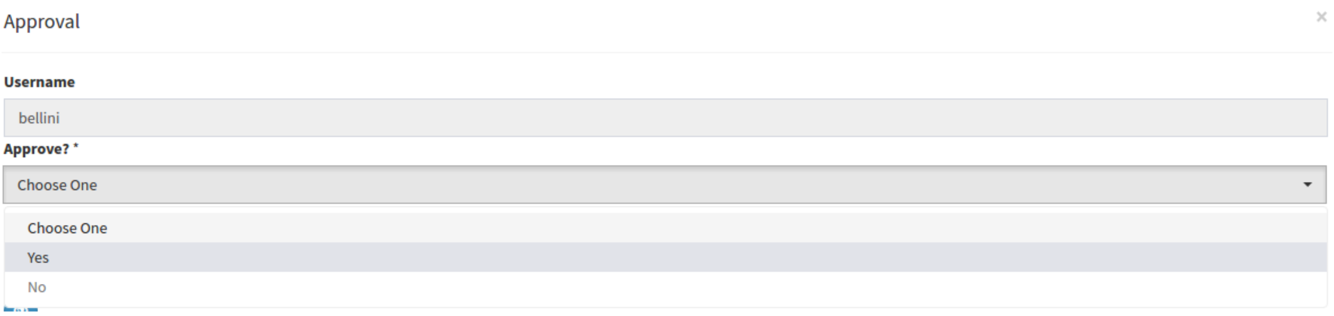
Destination realm: /

Username *: bellini

Password management

	Resource	Remote ID	Status
□	syncope	bellini	

Figure 15. Reviewing modifications



Approval

Username: bellini

Approve? *: Choose One

Figure 16. Approving modiications

Extensions

The extensions configured for the given deployment are dynamically reported in the navigation menu: each extension generally produces one or more pages and makes one or more widgets available in the dashboard.

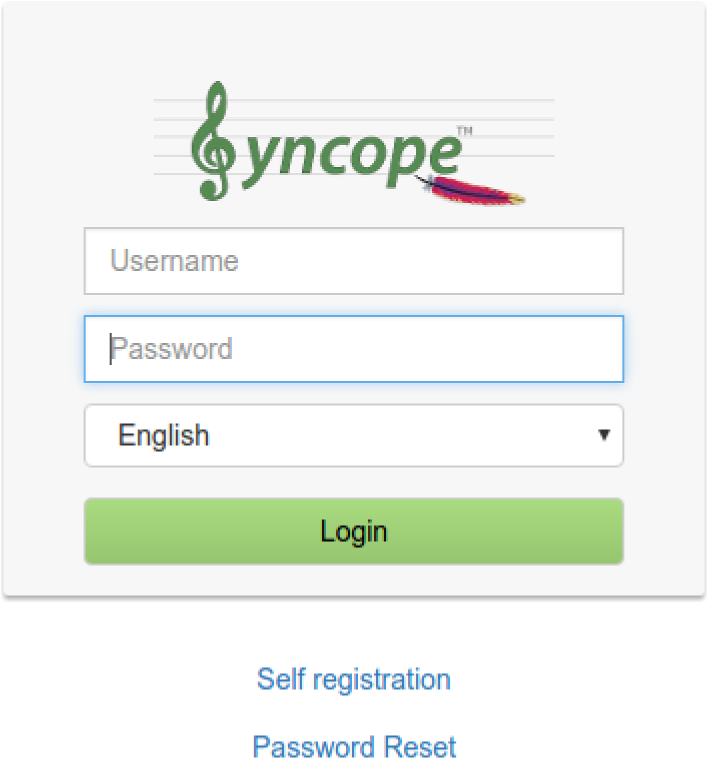
4.2. Enduser Application

Once the Java EE container has initialized, the enduser application can be accessed at:

```
protocol://host:port/syncope-enduser/
```

where `protocol`, `host` and `port` reflect your Java EE container installation.

The scope of the enduser application is primarily to provide a dedicated web-based entry-point for self-registration, self-service and password reset.



Usually, organizations thend to require deep customizations not only in the appearance but often also in the actual mechanisms behind, in order to best suit their processes and flows.
This is the main reason why the enduser application is composed of an AngularJS frontend - which eases extension and full customization - featured by an Apache Wicket backend - which proxies access to the Core, thus skipping several security concerns at a glance.

Nonetheless, the introduction of a client-side technology as AngularJS brought some important security issues to attention; above all, CRSF / XSRF and BOT attacks.
The enduser application offers protection mechanisms against all of them, and optionally consent to embed external features as Google re-Captcha.

While full-fledged front-end features are provided, it is important to highlight how these are primarily meant for customization on a given deployment.

4.3. CLI

Examples in this document are executed on GNU / Linux with *debug environment*.

4.3.1. Commands

Schema command

This command works with schemas.

Help message

```
Usage: schema [options]
  Options:
    --help
    --details
    --list-all
    --list-plain
    --list-derived
    --list-virtual
    --read {SCHEMA-TYPE} {SCHEMA-KEY}
        Schema type: PLAIN / DERIVED / VIRTUAL
    --delete {SCHEMA-TYPE} {SCHEMA-KEY}
        Schema type: PLAIN / DERIVED / VIRTUAL
```

Options

--details

This option shows a table with some details about the schemas and their categories.

--list-all

Running the command with this option you will see the list of all (PLAIN, DERIVED, VIRTUAL) schemas configured.

--list-plain

Running the command with this option you will see the list of the plain schemas available in Syncope.

--list-derived

Running the command with this option you will see the list of the derived schemas available in Syncope with their expressions.

--list-virtual

Running the command with this option you will see the list of the virtual schemas available in Syncope.

--read

The option to read all the information of a specified schema.

--delete

The option to delete a specified schema.

Connector command

This command works with connectors.

Help message

```
Usage: connector [options]
  Options:
    --help
    --details
    --list
    --list-bundles
    --list-configuration-properties
       Syntax: --list-configuration-properties {CONNECTOR-KEY} {CONNECTOR-KEY} [...]
    --read
       Syntax: --read {CONNECTOR-KEY} {CONNECTOR-KEY} [...]
    --delete
       Syntax: --delete {CONNECTOR-KEY} {CONNECTOR-KEY} [...]
```

Options

--details

This option shows a table with some details about connectors and bundles.

--list

Running the command with this option you will see the list of connectors with their configuration.

--list-bundles

Running the command with this option you will see the list of the bundles available in Syncope.

--list-configuration-properties

This option lists the configuration of specified connectors.

--read

The option to read all the information of specified connectors.

--delete

The option to delete a specified connector.

Resource command

This command works with external resources.

Help message

```
Usage: resource [options]
  Options:
    --help
    --details
    --list
    --read
       Syntax: --read {RESOURCE-KEY} {RESOURCE-KEY} [...]
    --delete
       Syntax: --delete {RESOURCE-KEY} {RESOURCE-KEY} [...]
```

Options

--details

This option shows a table with the amount of available resources.

--list

Running the command with this option you will see the list of resources.

--read

The option to read all the information of a specified resource.

--delete

The option to delete a specified resource.

User command

This command works with users.

Help message

```
Usage: user [options]
  Options:
    --help
    --list
    --details
    --get-user-key
       Syntax: --get-user-key {USERNAME} {USERNAME} [...]
    --get-username
       Syntax: --get-username {USER-KEY} {USER-KEY} [...]
    --read-by-username
       Syntax: --read-by-username {USERNAME} {USERNAME} [...]
    --read-by-userkey
       Syntax: --read-by-userkey {USER-KEY} {USER-KEY} [...]
    --search-by-attribute
       Syntax: --search-by-attribute {REALM} {ATTR-NAME}={ATTR-VALUE}
    --search-by-role
       Syntax: --search-by-role {REALM} {ROLE-KEY}
    --search-by-resource
       Syntax: --search-by-resource {REALM} {RESOURCE-KEY}
    --delete
       Syntax: --delete {USER-KEY} {USER-KEY} [...]
    --delete-all
       Syntax: --delete-all {REALM}
    --delete-by-attribute
       Syntax: --delete-by-attribute {REALM} {ATTR-NAME}={ATTR-VALUE}
```

Options

--details

This option shows a table with some details about the Users.

--list

Running the command with this option you will see the list of all Users in the environment. However, the system will ask you a confirmation before execution, because as you can imagine this operation might produce a very large output.

--get-user-key

The option to get the user key starting from a username.

--get-username

The option to get the username starting from a user key.

--read-by-userkey

The option to read user information by their user key.

--read-by-usernam

The option to read user information by their username.

--search-by-attribute

The option to search a list of Users with a common attribute.

--search-by-role

The option to search a list of Users with a specified role.

--search-by-resource

The option to search a list of Users with a specified resource.

--delete

The option to delete a specified user.

--delete-by-attribute

The option to delete the Users with a common attribute.

--delete-all

The option to delete all Users of the realm passed as input.

Group command

This command works with groups.

Help message

```
Usage: group [options]
  Options:
    --help
    --details
    --list
    --read
       Syntax: --read {GROUP-KEY} {GROUP-KEY} [...]
    --read-attr-by-schema-type {GROUP-KEY} {SCHEMA-TYPE}
       Schema type: PLAIN / DERIVED / VIRTUAL
    --read-attr-by-schema {GROUP-KEY} {SCHEMA-TYPE} {SCHEMA-NAME}
       Schema type: PLAIN / DERIVED / VIRTUAL
    --delete
       Syntax: --delete {GROUP-KEY} {GROUP-KEY} [...]
```

Options

--details

This option shows a table with the amount of available Groups and some additional information.

--list

Running the command with this option you will see the list of the Groups.

--read

The option to read the group passed as input.

--read-attr-by-schema-type

The option to read the specified attribute type of the group passed as input.

--read-attr-by-schema

The option to read the specified attribute name of the group passed as input.

--delete

The option to delete a specified group.

Any command

This command works with any objects.

Help message

```
Usage: any [options]
  Options:
    --help
    --details
    --list
    --read
       Syntax: --read {ANY_OBJECT-KEY} {ANY_OBJECT-KEY} [...]
    --read-attr-by-schema-type {ANY_OBJECT-KEY} {SCHEMA-TYPE}
       Schema type: PLAIN / DERIVED / VIRTUAL
    --read-attr-by-schema {ANY_OBJECT-KEY} {SCHEMA-TYPE} {SCHEMA-NAME}
       Schema type: PLAIN / DERIVED / VIRTUAL
    --delete
       Syntax: --delete {ANY_OBJECT-KEY} {ANY_OBJECT-KEY} [...]
```

Options

--details

This option shows a table with the amount of available Any Objects and some additional information.

--list

Running the command with this option you will see the list of the Any Objects.

--read

The option to read the any object passed as input.

--read-attr-by-schema-type

The option to read the specified attribute type of the any object passed as input.

--read-attr-by-schema

The option to read the specified attribute name of the any object passed as input.

--delete

The option to delete a specified any object.

Role command

This command works with roles.

Help message

```
Usage: role [options]
  Options:
    --help
    --details
    --list
    --read
       Syntax: --read {ROLE-KEY} {ROLE-KEY} [...]
    --delete
       Syntax: --delete {ROLE-KEY} {ROLE-KEY} [...]
```

Options

--details

This option shows a table with some details about the roles.

--list

Running the command with this option you will see the list of roles with the realm where they are configured and their entitlements.

--read

The option to read all the information of specified roles.

--delete

The option to delete specified roles.

Realm command

This command works with realms.

Help message

```
Usage: realm [options]
  Options:
    --help
    --details
    --list
```

Options

--details

This option shows a table with the amount of the available realms.

--list

Running the command with this option you will see the list of the realms.

Question command

This command works with security questions for use with password reset.

Help message

```
Usage: question [options]
  Options:
    --help
    --list
    --read
       Syntax: --read {QUESTION-KEY} {QUESTION-KEY} [...]
    --delete
       Syntax: --delete {QUESTION-KEY} {QUESTION-KEY} [...]
```

Options

--list

Running the command with this option you will see the list of questions with their content.

--read

The option to read all the information of specified questions.

--delete

The option to delete a specified question.

Configuration command

This command works with configuration parameters.

Help message

```
Usage: configuration [options]
  Options:
    --help
    --get
    --read
       Syntax: --read {CONF-NAME} {CONF-NAME} [...]
    --update
       Syntax: --update {CONF-NAME}={CONF-VALUE} {CONF-NAME}={CONF-VALUE} [...]
    --delete
       Syntax: --delete {CONF-NAME} {CONF-NAME} [...]
    --export
       Syntax: --export {WHERE-DIR}
```

Options

--get

This get option shows a table with the Syncope configuration.

--read

The option to read the value of specified configuration attributes.

--update

The option to update a value of specified configuration attributes.

--delete

The option to delete specified configuration attributes.

--export

The option to export the Syncope configuration to a specified directory.

Logger command

This command is meant for tweaking runtime logger configuration.

Help message

```
Usage: logger [options]
  Options:
    --help
    --details
    --list-memory-appenders
    --last-statements
       Syntax: --last-statements {APPENDER-NAME}
    --list
    --read
       Syntax: --read {LOG-NAME} {LOG-NAME} [...]
    --update
       Syntax: --update {LOG-NAME}={LOG-LEVEL} {LOG-NAME}={LOG-LEVEL} [...]
    --update-all
       Syntax: --update-all {LOG-LEVEL}
    --create
       Syntax: --create {LOG-NAME}={LOG-LEVEL} {LOG-NAME}={LOG-LEVEL} [...]
    --delete
       Syntax: --delete {LOG-NAME} {LOG-NAME} [...]
```

Options

--details

This option shows a table with some details about logger configuration.

--list

Running the command with this option you will see the table of the loggers configuration. --list -memory-appenders Running the command with this option you will see the table of the memory appenders, whose last statements can be inspected --last-statements The option to get the last statements available for the passed memory appender

--read

The option to read all the information of specified loggers.

--update

The option to change the value of the logger passed as input.

--update-all

This option is especially helpful in production environment when every log is disabled and you need to change them for a while in DEBUG mode.

--create

The option to add a new logger configuration.

--delete

The option to delete a specified logger.

Task command

This command works with tasks.

Help message

```
Usage: task [options]
  Options:
    --help
    --details
    --list
       Syntax: --list {TASK-TYPE}
          Task type: NOTIFICATION / PROPAGATION / PUSH / SCHEDULED / PULL
    --list-running-jobs
    --list-scheduled-jobs
    --read
       Syntax: --read {TASK-KEY} {TASK-KEY} [...]
    --read-execution
       Syntax: --read-execution {TASK-EXEC-KEY} {TASK-EXEC-KEY} [...]
    --delete
       Syntax: --delete {TASK-KEY} {TASK-KEY} [...]
    --delete-execution
       Syntax: --delete-execution {TASK-EXEC-KEY} {TASK-EXEC-KEY} [...]
    --execute
       Syntax: --execute {TASK-KEY} {DRY-RUN}
          Dry run: true / false
```

Options

--details

This option shows a table with some details about tasks and jobs.

--list

Running the command with this option you will see the list of selected tasks type with their information.

--list-scheduled-jobs

Running the command with this option you will see the list of the actual scheduled jobs.

--list-running-jobs

Running the command with this option you will see the list of the actual running jobs.

--read

The option to read all the information of a task.

--read-execution

The option to read all the information of the specified task execution(s).

--delete

The option to delete specified tasks.

--delete-execution

The option to delete the specified task execution(s).

--execute

The option to execute the specified task.

Notification command

This command works with notifications.

Help message

```
Usage: notification [options]
  Options:
    --help
    --list
    --read
       Syntax: --read {NOTIFICATION-KEY}
    --delete
       Syntax: --delete {NOTIFICATION-KEY}
```

Options

--list

Running the command with this option you will see the list of notifications with their configuration.

--read

The option to read all the information of the specified notifications.

--delete

The option to delete a specified notification.

Report command

This command works with reports.

Help message

```
Usage: report [options]
  Options:
    --help
    --details
    --list
    --list-jobs
    --read
       Syntax: --read {REPORT-KEY} {REPORT-KEY} [...]
    --delete
       Syntax: --delete {REPORT-KEY} {REPORT-KEY} [...]
    --execute
       Syntax: --execute {REPORT-KEY}
    --read-execution
       Syntax: --read-execution {EXECUTION-KEY} {EXECUTION-KEY} [...]
    --delete-execution
       Syntax: --delete-execution {EXECUTION-KEY} {EXECUTION-KEY} [...]
    --export-execution-result
       Syntax: --export-execution-result {EXECUTION-KEY} {EXECUTION-KEY} [...]
{FORMAT}
          Format: CSV / HTML / PDF / XML / RTF
```

Options

--details

This option shows a table with some details about the reports and their executions.

--list

Running the command with this option you will see the list of configured reports.

--list-jobs

Running the command with this option you will see the list of the report executions.

--read

The option to read all the information of a specified report.

--read-execution

The option to read all the information of a specified report execution.

--delete

The option to delete a specified report.

--delete-execution

The option to delete a specified report execution.

--execute

The option to run a report.

--export-execution-result

The option to export an execution in a certain format to see the results.

Policy command

This command works with policies.

Help message

```
Usage: policy [options]
  Options:
    --help
    --details
    --list
       Syntax: --list-policy {POLICY-TYPE}
          Policy type: ACCOUNT / PASSWORD / PULL / PUSH
    --read
       Syntax: --read {POLICY-KEY} {POLICY-KEY} [...]
    --delete
       Syntax: --delete {POLICY-KEY} {POLICY-KEY} [...]
```

Options

--details

This option shows a table with the amount of policies for each type.

--list

Running the command with this option you will see the list of the policies.

--read

The option to read all the information of a specified policy.

--delete

The option to delete a specified policy.

Info command

This command reports general information about the current Apache Syncope deployment.

Help message

```
Usage: info [options]
  Options:
    --version
    --pwd-reset-allowed
    --pwd-reset-with-question
    --self-reg-allowed
    --provisioning-manager-classes
    --workflow-adapter-classes
    --account-rules-classes
    --connid-locations
    --reconciliation-filter-builders
    --logic-actions
    --mail-templates
    --mapping-item-transformers
    --password-rules
    --propagation-actions
    --push-actions
    --push-correlation-actions
    --reportlets
    --sync-actions
    --sync-correlation-rules
    --task-jobs
    --validators
    --password-generator
    --vir-attr-cache
    --help
```

Entitlement command

This command works with entitlements.

Help message

```
Usage: entitlement [options]
  Options:
    --help
    --list
    --list-role
       Syntax: --list-role {ENTITLEMENT-NAME}
    --read-by-username
       Syntax: --read-by-username {USERNAME}
    --read-by-userkey
       Syntax: --read-by-userkey {USER-KEY}
    --search-by-role
       Syntax: --search-by-role {ROLE-KEY}
```

Options

--list

Running the command with this option you will see the list of the entitlements.

--list-role

Running the command with this option you will see the list of the roles with a certain entitlement.

--read-by-username

The option to read the entitlements of the username passed as input.

--read-by-userkey

The option to read the entitlements of the user key passed as input.

--search-by-role

The option to read the entitlements of a certain role.

Domain command

This command works with domains.

Help message

```
Usage: domain [options]
  Options:
    --help
    --details
    --list
    --delete
       Syntax: --delete {DOMAIN-KEY} {DOMAIN-KEY} [...]
```

Options

--details

This option shows a table with domain amount.

--list

Running the command with this option you will see the list of the domains.

--delete

The option to delete a specified domain.

4.4. RESTful services

All the features provided by the Core are available as RESTful services.

The base URL for invoking such services is normally set as

```
protocol://host:port/syncope/rest/
```

where protocol, host and port reflect your Java EE container installation.

> *REST Reference*
>
> A complete REST reference generated from WADL is published as well as made available with each deployment at
>
> ```
> protocol://host:port/syncope/
> ```
>
> where protocol, host and port reflect your Java EE container installation.

> The Swagger extension might also help greatly when working with RESTful services.

4.4.1. REST Authentication and Authorization

The Core authentication and authorization is based on Spring Security.

As an initial step, authentication is required to obtain, in the X-Syncope-Token HTTP header, the unique signed JSON Web Token to include in all subsequent requests.

By providing the token received in the initial exchange, the requester can be identified and checked for authorization, based on owned entitlements.

> Users can examine their own entitlements looking at the X-Syncope-Entitlements header value.

> The relevant security configuration lies in securityContext.xml; while normally not needed, this configuration can be anyway customized via the override behavior.
>
> HTTP Basic Authentication is set for use by default.

4.4.2. REST Headers

Apache Syncope supports a number of HTTP headers as detailed below, in addition to the common HTTP headers such as Accept, Content-Type, etc.

> It is possible to deal with the headers below when using the Client Library via the SyncopeClient class methods.

X-Syncope-Token

X-Syncope-Token is returned on response to successful authentication, and contains the unique signed JSON Web Token identifying the authenticated user.

The same header with provided value must be included in all subsequent requests, in order for the requester to be checked for authorization.

The token duration can be configured via the `jwt.lifetime.minutes` property - see below for details.

X-Syncope-Domain

`X-Syncope-Domain` can be optionally set for requests (when not set, `Master` is assumed) to select the target domain.
The value for this header is provided in all responses.

X-Syncope-Key and Location

When creating an entity (User, Group, Schema, External Resource, …) these two headers are populated respectively with the entity key (which may be auto-generated) and the absolute URI identifying the new REST resource.

X-Application-Error-Code and X-Application-Error-Info

If the requested operation is in error, `X-Application-Error-Code` will contain the error code (mostly from ClientExceptionType) and `X-Application-Error-Info` might be optionally populated with more details, if available.

X-Syncope-Null-Priority-Async

When set to `true`, this request header instructs the propagation process not to wait for completion when communicating with External Resources with no priority set.

Prefer and Preference-Applied

Some REST endpoints - typically for creating, updating or deleting Users, Groups or Any Objects - return the entity in the response payload by default.
If this is not required, the `Prefer` request header can be set to `return-no-content` (`return-content` will instead keep the default behavior).

When `Prefer` is specified in the request, the response will feature the `Preference-Applied` header, with value set to the effective preference applied.

> Use `Prefer` in scenarios where it is important to avoid unnecessary data in the response payload.

ETag, If-Match and If-None-Match

For each response containing Users, Groups or Any Objects, the ETag header is generated, which contains the latest modification date.

This value can be passed, during subsequent requests to modify the same entity, via the `If-Match` or `If-None-Match` headers.
When the provided `If-Match` value does not match the latest modification date of the entity, an error is reported and the requested operation is not performed.

The combined usage of `ETag` and `If-Match` can be enforced to implement optimistic concurrency control over Users, Groups and Any Objects operations.

X-Syncope-Entitlements

When invoking the REST endpoint `/users/self` in `GET`, the `X-Syncope-Entitlements` response header will list all the entitlements owned by the requesting user.

4.4.3. Bulk Operations

Some REST endpoints feature the *bulk mode*, e.g. the capability to perform a given operation onto several items at the same time.

The table below shows the bulk operations available.

Any Objects	• `DELETE` - remove several any objects at once
Groups	• `PROVISION` - provision all members of the given group onto all the associated external resources • `DEPROVISION` - deprovision all members of the given group from all the associated external resources • `DELETE` - remove several groups at once
Users	• `SUSPEND` - suspend several users at once • `REACTIVATE` - set several users at once back to the active state • `MUSTCHANGEPASSWORD` - force several users at once to change their passwords • `DELETE` - remove several users at once
Tasks	• `DRYRUN` - executes several tasks at once, with the DryRun option set • `EXECUTE` - executes several tasks at once • `DELETE` - remove several tasks at once
External Resources	• `DEPROVISION` - delete several users, groups or any objects at once from an external resource but keep in the internal storage and leave the external resource associated • `UNLINK` - remove the association between several users, groups or any objects at once and an external resource, without performing any deprovisioning operation • `UNASSIGN` - unlink and deprovision several users, groups or any objects at once from an external resource

4.4.4. Client Library

The Java client library simplifies the interaction with the Core by hiding the underlying HTTP communication details and providing native methods and payload objects.

The library is available as a Maven artifact:

```
<dependency>
  <groupId>org.apache.syncope.client</groupId>
  <artifactId>syncope-client-lib</artifactId>
  <version>2.0.4-SNAPSHOT</version>
</dependency>
```

> Do not forget to add the following repository to your `pom.xml`:
>
> ```
> <repository>
>   <id>ASF</id>
>
> <url>https://repository.apache.org/content/repositories/snapshots/</url
> >
>   <snapshots>
>     <enabled>true</enabled>
>   </snapshots>
> </repository>
> ```

Initialization

First you need to build an instance of `SyncopeClientFactoryBean` by providing the deployment base URL, as follows:

```
SyncopeClientFactoryBean clientFactory = new SyncopeClientFactoryBean().
              setAddress("http://localhost:9080/syncope/rest/");
```

You might also select a specific domain - other than `Master`, choose to exchange XML payloads - rather than JSON (default), or to select HTTP compression (more options in the Javadoc):

```
SyncopeClientFactoryBean clientFactory = new SyncopeClientFactoryBean().
              setAddress("http://localhost:9080/syncope/rest/").
              setDomain("Two").
              setContentType(SyncopeClientFactoryBean.ContentType.XML).
              setUseCompression(true);
```

At this point an instance of `SyncopeClient` can be obtained by passing the login credentials via:

```
SyncopeClient client = clientFactory.create("admin", "password");
```

Or you can combine into a single statement as:

```
SyncopeClient client = new SyncopeClientFactoryBean().
              setAddress("http://localhost:9080/syncope/rest/").
              create("admin", "password");
```

Usage

Select one of the RESTful services and invoke one of the available methods:

```
LoggerService loggerService = client.getService(LoggerService.class);

LoggerTO loggerTO = loggerService.read(LoggerType.LOG,
"org.apache.syncope.core.connid");
loggerTO.setLevel(LoggerLevel.DEBUG);

loggerService.update(LoggerType.LOG, loggerTO);
```

ℹ More RESTful services could be available besides the default set, as there might be extensions installed in the given deployment; the Apache Camel Provisioning Manager provides the CamelRouteService, for instance.

💡 Advanced REST features are also available from `SyncopeClient` instances: check the javadoc for more information.

Example 11. Search for users, groups or any objects

All search operations return paged result handlers which can be exploited both for getting the actual results and for extrapolating pagination coordinates.

```
UserService userService = client.getService(UserService.class);

int count = userService.search(new
AnyQuery.Builder().page(0).size(0).build()).getTotalCount(); ①

PagedResult<UserTO> matchingUsers = userService.search(
    new AnyQuery.Builder().realm(SyncopeConstants.ROOT_REALM).

fiql(SyncopeClient.getUserSearchConditionBuilder().is("username").equalTo("ros*ini
").query()).
    build()); ②

PagedResult<UserTO> matchingUsers = userService.search(
    new AnyQuery.Builder().realm(SyncopeConstants.ROOT_REALM).

fiql(SyncopeClient.getUserSearchConditionBuilder().isNull("loginDate").query()).
    build()); ③

PagedResult<UserTO> matchingUsers = userService.search(
    new AnyQuery.Builder().realm(SyncopeConstants.ROOT_REALM).
    fiql(SyncopeClient.getUserSearchConditionBuilder().inRoles("Other").query()).
    build()); ④

AnyObjectService anyObjectService = client.getService(AnyObjectService.class);

PagedResult<AnyObjectTO> matchingAnyObjects = anyObjectService.search(
    new AnyQuery.Builder().realm(SyncopeConstants.ROOT_REALM).
    fiql(SyncopeClient.getAnyObjectSearchConditionBuilder("PRINTER").query()).
    build()); ⑤

GroupService groupService = client.getService(GroupService.class);

PagedResult<GroupTO> matchingGroups = groupService.search(
    new AnyQuery.Builder().realm("/even/two").page(3).size(150).
    fiql(SyncopeClient.getGroupSearchConditionBuilder().isAssignable().
        and("name").equalTo("palo*").query()).
    build()); ⑥
```

① get the total number of users available in the given deployment (and domain)

② get users in the root realm with username matching the provided wildcard expression

③ get users in the root realm with no values for `loginDate`, i.e. that have never authenticated to the given deployment

④ get users in the root realm with role `Other` assigned

⑤ get all any objects in the root realm with type `PRINTER`

⑥ get all groups that can be assigned to users or any objects in the `/even/two` realm - third page of the result, where each page contains 150 items

Example 12. Delete several users at once

```
UserService userService = client.getService(UserService.class);

BulkAction bulkAction = new BulkAction();
bulkAction.setType(BulkAction.Type.DELETE);

final int pageSize = 100;
final int count = userService.search(
        new AnyQuery.Builder().page(0).size(0).build()).getTotalCount(); ①
for (int page = 1; page <= (count / pageSize) + 1; page++) {
    for (UserTO user : userService.search(
            new AnyQuery.Builder().page(page).size(pageSize).build()).getResult())
{  ②

        bulkAction.getTargets().add(user.getKey()); ③
    }
}

BulkActionResult bulkResult = userService.bulk(bulkAction).
        readEntity(BulkActionResult.class); ④
Map<String, BulkActionResult.Status> results = bulkResult.getResults(); ⑤
```

① get the total number of users available in the given deployment (and domain)

② loop throgh all users available, using paginated search

③ add each user to the bulk action

④ execute the `DELETE` bulk action

⑤ examine the bulk action results

Example 13. Self-read own profile information

```
Pair<Map<String, Set<String>>, UserTO> self = client.self();
UserTO userTO = self.getRight(); ①
Map<String, Set<String>> realm2entitlements = self.getLeft(); ②
```

① UserTO of the requesting user

② for each realm, the owned entitlements

Example 14. Change user status

```
String key = ...; ①
StatusPatch statusPatch = new StatusPatch();
statusPatch.setKey(key);
statusPatch.setType(StatusPatchType.SUSPEND); ②
UserTO userTO = userService.status(statusPatch).
  readEntity(new GenericType<ProvisioningResult<UserTO>>() {
  }).getEntity(); ③
```

① assume the key of the user to be suspended is known in advance

② `ACTIVATE`, `SUSPEND`, `REACTIVATE` values are accepted, and honored depending on the actual status of the user being updated

③ request for user update and read back the updated entity

4.5. Customization

Only Maven projects can be customized: if using Standalone, Debian packages or the GUI installer, none of the customizations discussed below can be applied.

Apache Syncope is designed to be as flexible as possible, to best suit the various environments in which it can be deployed. Besides other aspects, this means that every feature and component can be extended or replaced.

Once the project has been created from the provided Maven archetype, the generated source tree is available for either adding new features or replacing existing components.

Override behavior

As a rule of thumb, any file of the local project will take precedence over a file with the same name in the same directory of the standard Apache Syncope release.

For example, if you place

```
core/spring/src/main/java/org/apache/syncope/core/spring/security/SyncopeAuthenticationProvider.java
```

in the local project, this file will be picked up instead of SyncopeAuthenticationProvider.

The same happens with resources as images or HTML files; if you place

```
console/src/main/resources/org/apache/syncope/client/console/pages/BasePage.html
```

in the local project, this file will be picked up instead of BasePage.html.

This general behavior might have exceptions, as highlighted below.

In general, the Embedded Mode (see the Apache Syncope Getting Started Guide for details) allows the user to work comfortably from a single workstation, with no need of additional setup; it is effectively implemented as the `all` Maven profile, where the available optional components and extensions are enabled.
When deploying the generated WAR artifacts into an external JavaEE Container however, the required components and extensions need to be explicitly selected and enabled, as shown in the following text.

Deployment directories

Apache Syncope needs three base directories to be defined:

- bundles - where the connector bundles are stored;
- log - where all the system logs are written;
- conf (optional) - where configuration files are located, if overriding the default values is needed.

⚠ The `bundles` directory should only contain connector bundle JAR files.
The presence of any other file might cause the unavailability of any connector bundle in Apache Syncope.

For reference, the suggested directory layout can be created as follows:

```
$ mkdir /opt/syncope
$ mkdir /opt/syncope/bundles
$ mkdir /opt/syncope/log
$ mkdir /opt/syncope/conf
```

The WAR artifacts are generated by running the Maven command (with reference to the suggested directory layout):

```
$ mvn clean verify \
   -Dconf.directory=/opt/syncope/conf \
   -Dbundles.directory=/opt/syncope/bundles \
   -Dlog.directory=/opt/syncope/log
$ cp core/target/classes/*properties /opt/syncope/conf
$ cp console/target/classes/*properties /opt/syncope/conf
$ cp enduser/target/classes/*properties /opt/syncope/conf
$ cp enduser/target/classes/customForm.json /opt/syncope/conf
```

After downloading all of the dependencies that are needed, three WAR files will be produced:

1. `core/target/syncope.war`
2. `console/target/syncope-console.war`
3. `enduser/target/syncope-enduser.war`

If no failures are encountered, your basic Apache Syncope project is now ready to be deployed.

JPDA Debug in Embedded Mode

The Java™ Platform Debugger Architecture (JPDA) is a collection of APIs aimed to help with debugging Java code.

Enhancing the embedded profile of the enduser module to enable the JPDA socket is quite straightforward: just add the <profile> below to enduser/pom.xml:

```
<profile>
  <id>debug</id>

  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <inherited>true</inherited>
        <configuration>
          <configuration>
            <properties>
              <cargo.jvmargs>
              -Xdebug

-Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
              -noverify -XX:+CMSClassUnloadingEnabled
              -XX:+UseConcMarkSweepGC -Xmx1024m -Xms512m
              </cargo.jvmargs>
            </properties>
          </configuration>
        </configuration>
      </plugin>
    </plugins>
  </build>
</profile>
```

Now, from the enduser subdirectory, execute:

```
mvn -P embedded,debug
```

At this point your favorite IDE can be attached to the port 8000; please note that you might need to add -XX:MaxPermSize=512m to <cargo.jvmargs> in order to run with JDK 7.

4.5.1. Core

When providing custom Java classes implementing the defined interfaces or extending the existing implementations, their package **must** be rooted under org.apache.syncope.core, otherwise they will not be available at runtime.

Besides replacing existing classes as explained above, new implementations can be provided under `core/src/main/java` for the following components:

- propagation, push, pull and logic actions
- push / pull correlation rules
- reconciliation filter builders
- custom tasks
- reportlets
- account and password rules for policies
- plain schema validators
- mapping item transformers
- virtual attribute cache
- workflow adapters
- provisioning managers
- notification recipient providers

New REST endpoints

Adding a new REST endpoint involves several operations:

1. create - in an extension's `rest-api` module or under `common` otherwise - a Java interface with package `org.apache.syncope.common.rest.api.service` and proper JAX-RS annotations; check CamelRouteService for reference;
2. if needed, define supporting payload objects - in an extension's `common-lib` module or under `common` otherwise; check CamelRouteTO for reference;
3. implement - in an extension's `rest-cxf` module or under `core` otherwise - the interface defined above in a Java class with package `org.apache.syncope.core.rest.cxf.service`; check CamelRouteServiceImpl for reference.

By following such conventions, the new REST endpoint will be automatically picked up alongside the default services.

The override behavior might have exceptions; if you need to customize one of the Spring context definitions. For example, if you want to customize securityContext.xml , you will also need to replace the following text in `core/src/main/webapp/WEB-INF/web.xml`,

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

with

```
classpath*:/coreContext.xml
classpath:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

to be sure that `core/src/main/resources/securityContext.xml` is picked up.
Please also note that the actual list of Spring context files to include might depend on the configured extensions.

Select the Activiti User Workflow Adapter

Add the following dependency to `core/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.core</groupId>
  <artifactId>syncope-core-workflow-activiti</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Copy `core/src/main/resources/all/workflow.properties` to `core/src/main/resources/workflow.properties`.

Enable the Apache Camel Provisioning Manager

Add the following dependencies to `core/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.camel</groupId>
  <artifactId>syncope-ext-camel-rest-cxf</artifactId>
  <version>${syncope.version}</version>
</dependency>
<dependency>
  <groupId>org.apache.syncope.ext.camel</groupId>
  <artifactId>syncope-ext-camel-persistence-jpa</artifactId>
  <version>${syncope.version}</version>
</dependency>
<dependency>
  <groupId>org.apache.syncope.ext.camel</groupId>
  <artifactId>syncope-ext-camel-provisioning</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Copy `core/src/main/resources/all/provisioning.properties` to `core/src/main/resources/provisioning.properties`.

Enable the Swagger extension

Add the following dependency to `core/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext</groupId>
  <artifactId>syncope-ext-swagger-ui</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Enable the SAML 2.0 Service Provider extension

Add the following dependencies to `core/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.saml2sp</groupId>
  <artifactId>syncope-ext-saml2sp-rest-cxf</artifactId>
  <version>${syncope.version}</version>
</dependency>
<dependency>
  <groupId>org.apache.syncope.ext.saml2sp</groupId>
  <artifactId>syncope-ext-saml2sp-persistence-jpa</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Copy `core/src/main/resources/all/saml2sp-logic.properties` to `core/src/main/resources/saml2sp-logic.properties`.

Setup a keystore and place it under the configuration directory, then review the content of `core/src/main/resources/saml2sp-logic.properties` accordingly.

Enable the Elasticsearch extension

⚠ This extension requires the latest JDK 8 that is available.

Add the following dependencies to `core/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.elasticsearch</groupId>
  <artifactId>syncope-ext-elasticsearch-provisioning-java</artifactId>
  <version>${syncope.version}</version>
</dependency>
<dependency>
  <groupId>org.apache.syncope.ext.elasticsearch</groupId>
  <artifactId>syncope-ext-elasticsearch-persistence-jpa</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Download

persistence.properties

and

elasticsearchClientContext.xml

then save both under `core/src/main/resources`.

Now, adjust the parameters in `core/src/main/resources/elasticsearchClientContext.xml` to match your Elasticsearch deployment.

Finally, replace the following text in `core/src/main/webapp/WEB-INF/web.xml`:

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

with

```
        classpath*:/coreContext.xml
        classpath*:/elasticsearchClientContext.xml
        classpath*:/securityContext.xml
        classpath*:/logicContext.xml
        classpath*:/restCXFContext.xml
        classpath*:/persistenceContext.xml
        classpath*:/provisioning*Context.xml
        classpath*:/workflow*Context.xml
```

4.5.2. Console

When providing custom Java classes implementing the defined interfaces or extending the existing implementations, their package **must** be rooted under `org.apache.syncope.client.console`, otherwise they will not be available at runtime.

Enable the Apache Camel Provisioning Manager

Add the following dependency to `console/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.camel</groupId>
  <artifactId>syncope-ext-camel-client-console</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Enable the SAML 2.0 Service Provider extension

Add the following dependencies to `console/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.saml2sp</groupId>
  <artifactId>syncope-ext-saml2sp-client-console</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Copy `console/src/main/resources/all/saml2sp-agent.properties` to `console/src/main/resources/saml2sp-agent.properties`.

4.5.3. Enduser

Given the nature of the Enduser Application, all the files required by the AngularJS-based frontend to run are generated under the local project's `enduser/src/main/webapp/app/` directory and are available for full customization.

The files in use by the Apache Wicket-based backend are still subject to the general override behavior, instead.

Enable the SAML 2.0 Service Provider extension

Add the following dependencies to `enduser/pom.xml`:

```
<dependency>
  <groupId>org.apache.syncope.ext.saml2sp</groupId>
  <artifactId>syncope-ext-saml2sp-client-enduser</artifactId>
  <version>${syncope.version}</version>
</dependency>
```

Copy `enduser/src/main/resources/all/saml2sp-agent.properties` to `enduser/src/main/resources/saml2sp-agent.properties`.

i18n

The Enduser Application comes with a native internationalization mechanism.

Under the `enduser/src/main/webapp/app/languages/` directory, a sub-directory for each supported language is available; each language sub-directory contains two JSON files:

- `static.json` for application messages;
- `dynamic.json` for labels (including attributes).

Changing the content of these files will result in updating the Enduser messages accordingly.

In order to add support for a new language (taking French as reference):

- add the support for the new language by updating `index.html`:

```
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.it.js"></script>
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.en.js"></script>
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.de.js"></script>
```

in

```
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.it.js"></script>
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.en.js"></script>
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.de.js"></script>
  <script src="../webjars/kendo-ui-core/${kendo-ui-
core.version}/js/cultures/kendo.culture.fr.js"></script>
```

- add the new language entry in `js/app.js` under `availableLanguages`, by updating

```
    $rootScope.languages = {
      availableLanguages: [
        {id: '1', name: 'Italiano', code: 'it', format: 'dd/MM/yyyy
HH:mm'},
        {id: '2', name: 'English', code: 'en', format: 'MM/dd/yyyy
HH:mm'},
        {id: '3', name: 'Deutsch', code: 'de', format: 'dd/MM/yyyy
HH:mm'}
      ]
    };
```

as

```
    $rootScope.languages = {
      availableLanguages: [
        {id: '1', name: 'Italiano', code: 'it', format: 'dd/MM/yyyy
HH:mm'},
        {id: '2', name: 'English', code: 'en', format: 'MM/dd/yyyy
HH:mm'},
        {id: '3', name: 'Deutsch', code: 'de', format: 'dd/MM/yyyy
HH:mm'}
        {id: '4', name: 'Français', code: 'fr', format: 'dd/MM/yyyy
HH:mm'}
      ]
    };
```

- copy the `enduser/src/main/webapp/app/languages/en/` directory into `enduser/src/main/webapp/app/languages/fr/` and modify the JSON files under the new directory

Form customization

The Enduser Application allows to customize the form in order to:

- hide / show attributes

- set attributes read-only for users
- provide default value(s)

Under the `enduser/src/main/resources` directory, the `customForm.json` file is available, allowing to configure form customization.

Hot deploy

The `customForm.json` could be edited and reloaded without the need of re-starting the Java EE container.

The `customForm.json` default content is just an empty object `{}`: if such file is missing, empty or not valid, form customization will be simply disabled and all attributes will be shown.

Example 15. Sample form customization

```
{
  "PLAIN":
          {
            "show": true,
            "attributes": {
              "firstname": {
                "readonly": true,
                "defaultValues": ["defaultFirstname1", "defaultFirstname2"]
              },
              "surname": {
                "readonly": false,
                "defaultValues": []
              },
              "fullname": {
                "readonly": false
              },
              "email": {
                "readonly": false,
                "defaultValues": ["test@apache.org"]
              },
              "userId": {
                "readonly": false
              },
              "cool": {
                "readonly": true,
                "defaultValues": ["true"]
              },
              "additional#loginDate": {
                "readonly": false
              },
              "additional#cool": {
                "readonly": false,
                "defaultValues": ["true"]
              }
            }
          },
  "DERIVED":
          {
            "show": false
          },
  "VIRTUAL":
          {
            "show": true,
            "attributes": {
              "virtualdata": {
                "readonly": true,
                "defaultValues": ["defaultVirtualData"]
              }
            }
          }
}
```

The `customForm.json` file has two main levels:

1. Schema type, e.g. `PLAIN`, `DERIVED`, `VIRTUAL`;
2. Attributes: list of attributes (by schema type) to be shown on the form.

Schema type

The schema type level allows to define customization of the three sub-forms available in the Enduser Application's form.

Only one, two or all three sections can be specified, in order to customize only what is really needed.

Moreover, a global boolean field `show` is available, to indicate that the whole sub-form should be shown or hidden. When not specified, `show` is treated as `true`.

Attributes

The attributes level contains a map of attributes to show.

Each attribute has:

- a name, e.g. the name of the Schema from which the attribute is generated
- a body, that specifies if the attribute should be readonly, and possibly its default values

Example 16. Form attribute specification

```
            "firstname": {
              "readonly": true,
              "defaultValues": ["defaultFirstname1", "defaultFirstname2"]
            },
```

Here, `firstname` is readonly and has two default values `defaultFirstname1` and `defaultFirstname2`.

An empty `attributes` field translates to skip filtering and show all attributes; for example:

```
{
  "PLAIN":
          {
            "show": true,
            "attributes": {}
          }
}
```

shows all `PLAIN` attributes.

If all attributes are to be hidden, please set `"show": false`, instead.

The `readonly` field should not be confused with the read-only flag available for Plain and Virtual schema.
Within Enduser form customization, `readonly` prevenst the user’s browser to modify the value of a given attribute.

`defaultValues` is a string array: this means, in particular, that date values should be specified as strings (timestamps).
Moreover, `defaultValues` do not overwrite any existing value.

4.5.4. Extensions

Extensions can be part of a local project, to encapsulate special features which are specific to a given deployment.

For example, the CHOReVOLUTION IdM - based on Apache Syncope - provides an extension for managing via the Core and visualizing via the Admin UI the running choreography instances.

4.6. System Administration

Where are the configuration files?

Depending on which Apache Syncope distribution you are running, the configuration files mentioned in the following text might reside in different locations.

Standalone

Assuming that `$CATALINA_HOME` is the Apache Tomcat base directory created when the distribution archive was unzipped, the configuration files are located under

- `$CATALINA_HOME/webapps/syncope/WEB-INF/classes/`
- `$CATALINA_HOME/webapps/syncope-console/WEB-INF/classes/`
- `$CATALINA_HOME/webapps/syncope-enduser/WEB-INF/classes/`

Debian packages

The configuration files will be first searched in `/etc/apache-syncope`, then under

- `/usr/share/tomcat8/webapps/syncope/WEB-INF/classes/`
- `/usr/share/tomcat8/webapps/syncope-console/WEB-INF/classes/`
- `/usr/share/tomcat8/webapps/syncope-enduser/WEB-INF/classes/`

GUI installer

Assuming that `$CONF_DIRECTORY` is the directory selected from the installer, the configuration files will be first searched in `$CONF_DIRECTORY`, then under the selected Java EE container's application classpath.

Maven project

Assuming that `$CONF_DIRECTORY` is the directory passed among deployment directories at build time and that `$SOURCE` is the path where the Maven project was generated, the configuration files will be first searched in `$CONF_DIRECTORY`, then under the selected Java EE container's application classpath, according to the content of

- `$SOURCE/core/target/classes/`
- `$SOURCE/console/target/classes/`
- `$SOURCE/enduser/target/classes/`

4.6.1. DBMS

The changes reported below to support different DBMSes are not complete files, but only show the lines that need to be updated.

PostgreSQL

In `provisioning.properties`:

```
quartz.jobstore=org.quartz.impl.jdbcjobstore.PostgreSQLDelegate
quartz.sql=tables_postgres.sql
```

In `domains/Master.properties` (for the `Master` domain):

```
Master.driverClassName=org.postgresql.Driver
Master.url=jdbc:postgresql://localhost:5432/syncope
Master.schema=
Master.username=syncope
Master.password=syncope
Master.databasePlatform=org.apache.openjpa.jdbc.sql.PostgresDictionary
Master.orm=META-INF/spring-orm.xml
```

⚠ This assumes that you have a PostgreSQL instance running on localhost, listening on its default port 5432 with a database `syncope` fully accessible by user `syncope` with password `syncope`.

MySQL

In `provisioning.properties`:

```
quartz.jobstore=org.quartz.impl.jdbcjobstore.StdJDBCDelegate
quartz.sql=tables_mysql_innodb.sql
```

⚠ This assumes that the InnoDB engine is enabled in your MySQL instance - if this is not the case, then change the value for `quartz.sql` to `tables_mysql.sql`.

In `domains/Master.properties` (for the `Master` domain):

```
Master.driverClassName=com.mysql.jdbc.Driver
Master.url=jdbc:mysql://localhost:3306/syncope?characterEncoding=UTF-8
Master.schema=
Master.username=syncope
Master.password=syncope
Master.databasePlatform=org.apache.openjpa.jdbc.sql.MySQLDictionary(blobTypeName=LONGB
LOB)
Master.orm=META-INF/spring-orm.xml
Master.audit.sql=audit_mysql_innodb.sql
```

⚠ This assumes that the InnoDB engine is enabled in your MariaDB instance - if this is not the case, then change the value for `Master.audit` to `audit.sql`.

⚠ This assumes that you have a MySQL instance running on localhost, listening on its default port 3306 with a database `syncope` fully accessible by user `syncope` with password `syncope`.

MariaDB

In provisioning.properties:

```
quartz.jobstore=org.quartz.impl.jdbcjobstore.StdJDBCDelegate
quartz.sql=tables_mariadb.sql
```

> ⚠ This assumes that the InnoDB engine is enabled in your MariaDB instance - if this is not the case, then change the value for quartz.sql to tables_mariadb.sql.

In domains/Master.properties (for the Master domain):

```
Master.driverClassName=org.mariadb.jdbc.Driver
Master.url=jdbc:mariadb://localhost:3306/syncope?characterEncoding=UTF-8
Master.schema=
Master.username=syncope
Master.password=syncope
Master.databasePlatform=org.apache.openjpa.jdbc.sql.MariaDBDictionary(blobTypeName=LONGBLOB)
Master.orm=META-INF/spring-orm.xml
```

> ⚠ This assumes that you have a MySQL instance running on localhost, listening on its default port 3306 with a database syncope fully accessible by user syncope with password syncope.

Oracle Database

In provisioning.properties:

```
quartz.jobstore=org.quartz.impl.jdbcjobstore.oracle.OracleDelegate
quartz.sql=tables_oracle.sql
```

In domains/Master.properties (for the Master domain):

```
Master.driverClassName=oracle.jdbc.OracleDriver
Master.url=jdbc:oracle:thin:@localhost:1521:orcl
Master.schema=SYNCOPE
Master.username=syncope
Master.password=syncope
Master.databasePlatform=org.apache.openjpa.jdbc.sql.OracleDictionary
Master.orm=META-INF/spring-orm-oracle.xml
Master.audit.sql=audit_oracle.sql
```

> ⚠ This assumes that you have an Oracle instance running on localhost, listening on its default port 1521 with a database syncope under tablespace SYNCOPE, fully accessible by user syncope with password syncope.

MS SQL Server

In provisioning.properties:

```
quartz.jobstore=org.quartz.impl.jdbcjobstore.MSSQLDelegate
quartz.sql=tables_sqlServer.sql
```

In domains/Master.properties (for the Master domain):

```
Master.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
Master.url=jdbc:sqlserver://localhost:1344;database=syncope;selectMethod=cursor;sendSt
ringParametersAsUnicode=false
Master.schema=dbo
Master.username=syncope
Master.password=syncope
Master.databasePlatform=org.apache.openjpa.jdbc.sql.SQLServerDictionary
Master.orm=META-INF/spring-orm-sqlserver.xml
Master.audit.sql=audit_sqlserver.sql
```

> ⚠ This assumes that you have a MS SQL Server instance running on localhost, listening on its default port 1344 with a database syncope fully accessible by user syncope with password syncope.

4.6.2. Database Connection Pool

The internal storage is the central place where all data of a given Core deployment are located.

After choosing the appropriate DBMS, it is of fundamental importance to provide an adequate configuration for the related database connection pool.

The database connection pool can be:

1. Application-managed (default); based on HikariCP, the related parameters can be tuned in the related domain configuration file, e.g. domains/Master.properties, for the Master domain.
2. JavaEE Container-managed, via the JNDI resource matching the name specified for a given domain, e.g. java:comp/env/jdbc/syncopeMasterDataSource for the Master domain.
   Each JavaEE Container provides its own way to accomplish this task:
   - Apache Tomcat 8
   - Apache Tomcat 8.5
   - Glassfish 4.1
   - Payara

  - Wildfly 9
  - Wildfly 10

4.6.3. JavaEE Container

Apache Tomcat 8 and 8.5

On GNU / Linux - Mac OS X, create `$CATALINA_HOME/bin/setenv.sh` with similar content (keep everything on a single line):

```
JAVA_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server \
-Xms1536m -Xmx1536m -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m \
-XX:MaxPermSize=256m -XX:+DisableExplicitGC"
```

On MS Windows, create `%CATALINA_HOME%\bin\setenv.bat` with similar content (keep everything on a single line):

```
set JAVA_OPTS=-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server
-Xms1536m -Xmx1536m -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m
-XX:MaxPermSize=256m -XX:+DisableExplicitGC
```

It is recommended to define a separate datasource for each domain (the following example is for the `Master` domain and MySQL): please also check that the connection parameters are the same as configured for DBMS:

```
<Resource name="jdbc/MasterDataSource" auth="Container" type="javax.sql.DataSource"
          factory="org.apache.tomcat.jdbc.pool.DataSourceFactory" testWhileIdle="true"
          testOnBorrow="true" testOnReturn="true" validationQuery="SELECT 1"
validationInterval="30000"
          maxActive="100" minIdle="2" maxWait="10000" initialSize="2"
removeAbandonedTimeout="20000"
          removeAbandoned="true" logAbandoned="true" suspectTimeout="20000"
          timeBetweenEvictionRunsMillis="5000" minEvictableIdleTimeMillis="5000"
          jdbcInterceptors="org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;
            org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer"
          username="syncope" password="syncope"
driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/syncope?characterEncoding=UTF-8"/>
```

Be sure to put the corresponding JDBC driver JAR file under `$CATALINA_HOME/lib` for each datasource defined.

Glassfish 4.1 and Payara Server

When using a datasource for internal storage, be sure to add

```
<resource-ref>
  <res-ref-name>jdbc/MasterDataSource</res-ref-name>
  <jndi-name>jdbc/MasterDataSource</jndi-name>
</resource-ref>
```

right after `</context-root>` in `core/src/main/webapp/WEB-INF/glassfish-web.xml`, assuming that your Glassfish instance provides a datasource named `jdbc/MasterDataSource`.

Wildfly 9 and 10

Replace

```
    <dependency>
      <groupId>org.apache.syncope.core</groupId>
      <artifactId>syncope-core-persistence-jpa</artifactId>
    </dependency>
```

with

```
    <dependency>
      <groupId>org.apache.syncope.core</groupId>
      <artifactId>syncope-core-persistence-jpa</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.apache.bval</groupId>
          <artifactId>bval-jsr</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
```

in `core/pom.xml`.

Add

```
    <dependency>
      <groupId>javax.xml.ws</groupId>
      <artifactId>jaxws-api</artifactId>
      <version>2.2.11</version>
    </dependency>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-core</artifactId>
      <version>${cxf.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-transports-http</artifactId>
      <version>${cxf.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-ws-policy</artifactId>
      <version>${cxf.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-wsdl</artifactId>
      <version>${cxf.version}</version>
    </dependency>
```

as additional dependencies in `core/pom.xml`, `console/pom.xml` and `enduser/pom.xml`.

Replace

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

with

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

in core/src/main/webapp/WEB-INF/web.xml.

Download restCXFContext.xml and save it under core/src/main/resources/.

Finally, add

```
<property name="jpaPropertyMap">
  <map>
    <entry key="openjpa.MetaDataFactory"
           value="jpa(URLs=vfs:/content/${project.build.finalName}.war/WEB-
INF/classes, Resources=${Master.orm})"/>
  </map>
</property>
```

in core/src/main/resources/domains/MasterDomain.xml for the MasterEntityManagerFactory bean.

4.6.4. High-Availability

When deploying multiple Syncope Core instances with a single database or database cluster, it is of fundamental importance that the contained OpenJPA instances are correctly configured for remote event notification.
Such configuration, in fact, allows the OpenJPA data cache to remain synchronized when deployed in multiple JVMs, thus enforcing data consistency across all Syncope Core instances.

Replace

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath*:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

with

```
classpath*:/coreContext.xml
classpath*:/securityContext.xml
classpath*:/logicContext.xml
classpath*:/restCXFContext.xml
classpath:/persistenceContext.xml
classpath*:/provisioning*Context.xml
classpath*:/workflow*Context.xml
```

in core/src/main/webapp/WEB-INF/web.xml.

Download persistenceContext.xml and domains.xml then save both under core/src/main/resources/.

The default configuration in domains.xml is

```
<entry key="openjpa.RemoteCommitProvider" value="sjvm"/>
```

which is suited for single JVM installations; with multiple instances, more options like as TCP or JMS are available; see the OpenJPA documentation for reference.

> ⚠ The OpenJPA documentation's XML snippets refer to a different configuration style; for example, when used in domains.xml, this:
>
> ```
> <property name="openjpa.RemoteCommitProvider"
> value="tcp(Addresses=10.0.1.10;10.0.1.11)"/>
> ```
>
> becomes:
>
> ```
> <entry key="openjpa.RemoteCommitProvider"
> value="tcp(Addresses=10.0.1.10;10.0.1.11)"/>
> ```

4.6.5. Domains Management

Domains are defined by three files in the configuration directory; assuming that the domain name is Two, such files are:

- domains/TwoDomain.xml - general configuration;
- domains/Two.properties - for DBMS parameters;
- domains/TwoContent.xml - for content initialization.

When adding a new domain to an existing deployment it is possible to copy, rename and edit the files available for the Master domain, which is always present.

> 🔥 Adding a new domain requires re-deploying the Core application and restarting the Java EE container.

Once a new domain is added, the admin credentials for such domain can be set via the admin console, CLI or barely invoking the REST layer through curl.

Example 17. Create new domain and set admin credentials

The JSON payload below, when sent via `POST` to the `/domains` endpoint, will create a new `newDomain` domain, and set the admin password to `newPassword`, to be stored via `SHA` cipher.

```
{
  "key": "newDomain",
  "adminPwd": "newPassword",
  "adminCipherAlgorithm": "SHA"
}
```

4.6.6. Set admin credentials

⚠ The procedure below affects only the `Master` domain; for other domains check above.

The default password for the `admin` user is `password`.

The credentials are defined in the `security.properties` file; text encoding must be set to UTF-8:

- `adminUser` - administrator username
- `adminPassword` - SHA1 hash evaluation of cleartext password (represented as a sequence of 40 hexadecimal digits)
- `adminPasswordAlgorithm` - algorithm to be used for hash evaluation (default value: SHA1)

For GNU / Linux and Mac OS X, the SHA1 password can be obtained via the `sha1sum` command-line tool of GNU Core Utilities:

```
echo -n "new_password" | sha1sum
```

For MS Windows, some options are available:

- MS File Checksum Integrity Verifier
  install, save your password to a file (e.g. `password.txt` without EOL) and issue at command line:

```
fciv.exe -sha1 password.txt
```

- GnuWin32 port of GNU utilities for MS Windows
- Cygwin Unix-like environment and command-line interface for Microsoft Windows (featuring GNU Core Utilities)

4.6.7. Configure ConnId locations

Apache Syncope can be configured to use either local or remote connector bundles:

- **local** connector bundles are located somewhere in the same filesystem where the Java EE container running Apache Syncope is deployed;
- **remote** connector bundles are provided via Java or .NET connector server.

While local connector bundles feature an easy setup, remote connector bundles allow enhanced deployment scenarios and are particularly useful when it is needed to deal with architectural security constraints or when a connector bundle requires to run on a specific platform OS (say MS Windows) while Apache Syncope is deployed on another platform OS (say GNU/Linux).

The `connid.properties` file holds the configuration for defining which ConnId locations (either local or remote) will be considered.

The format is quite straightforward:

```
connid.locations=location1,\
location2,\
...
locationN
```

where each location is the string representation of an URI of the form `file:/path/to/directory/` for local locations, `connid://key@host:port` for remote non-SSL connector servers or finally `connids://key@host:port[?trustAllcerts=true]` for remote SSL connector servers, with optional flag to disable certificate check.

Example 18. Single local location

```
connid.locations=file:/opt/syncope/bundles/
```

Example 19. Single remote location

```
connid.locations=connid://sampleKey@windows2008:4554
```

Example 20. Multiple locations

```
connid.locations=file:/opt/syncope/bundles/,\
file:/var/tmp/bundles/,\
connid://sampleKey@windows2008:4554,\
connids://anotherKey@windows2008:4559,\
connids://aThirdKey@linuxbox:9001?trustAllCerts=true
```

4.6.8. Deal with internal storage export - import

Almost every configurable aspect of a given deployment is contained in the internal storage: schemas, connectors, resources, mapping, roles, groups, tasks and other parameters.

During the implementation phase of an Apache Syncope-based project, it might be useful to move such configuration back and forth from one Apache Syncope instance to another (say developer's laptop and production server).
One option is clearly to act at a low level by empowering DBMS' dump & restore capabilities, but what if the developer is running MySQL (or even in-memory H2) while the sysadmin features Oracle?

Wipe existing content

When not running in-memory H2, the internal storage's content must be wiped before starting Apache Syncope, otherwise the provided content will be just ignored.

Check `core-persistence.log` for message

```
Empty database found, loading default content
```

If the internal storage is not empty, instead, you will get

```
Data found in the database, leaving untouched
```

All references in the following are set to `MasterContent.xml`; when other domains are defined, the content file is renamed accordingly. For example, `TwoContent.xml` if domain name is `Two`.

MySQL and lower case table names

On some platforms (namely, Mac OS X) MySQL is configured by default to be case insensitive: in such cases, you might want to edit the `/etc/my.cnf` file and add the following line in the `[mysqld]` section:

```
lower_case_table_names=1
```

Export

This task can be accomplished either via the admin console, CLI or by barely invoking the REST layer through curl, for example:

```
curl -X GET -u admin:password -o MasterContent.xml \
  http://localhost:9080/syncope/rest/configurations/stream
```

Import

Basically, all you need to do is to replace the local `MasterContent.xml` with the one exported as explained above; this file is located at:

- `$TOMCAT_HOME/webapps/syncope/WEB-INF/classes/domains/MasterContent.xml` for Standalone
- `/usr/share/tomcat8/webapps/syncope/WEB-INF/classes/domains/MasterContent.xml` for Debian packages
- `core/src/test/resources/domains/MasterContent.xml` for Maven projects in embedded mode
- `core/src/main/resources/domains/MasterContent.xml` for Maven projects

4.6.9. Enable the Activiti Modeler

Due to licensing issues - see this comment for more information - it is not possible to embed the Activiti Modeler, which provides a powerful graphical web editing interface for Activiti BPM, into any Apache Syncope artifact; thus, some manual steps are required to enable it on a working Apache Syncope deployment, for use with the admin console.

> This procedure requires Apache Maven since it is using a fake project to perform all of the required setup tasks.

First of all, generate a new Maven project as described in the Apache Syncope Getting Started Guide, then build via

```
mvn -P all clean install
```

At this point, copy the `console/target/activiti-modeler/` directory in the desired location of the host where the admin console is deployed, then set the value of `activitiModelerDirectory` with this path in the `console.properties` file.

4.6.10. Install connector bundles

Connector bundles are made available as JAR files and can be configured, for a given deployment:

- for Maven project, in local sources;
- for all distributions, at run-time.

Local sources

Different version of predefined connector bundle

First of all, verify which connector bundles are predefined in your project by looking at your project’s parent POM.

As you can see, there are several Maven properties on the form `connid.*.version`, controlling the related connector bundle’s version.

If you want your own project to use a different version of a given connector bundle, all you need to do is to override the related property in your own project's root pom.xml.

Hence, supposing that you would like to use `net.tirasa.connid.bundles.db.table` version `2.2.5-SNAPSHOT` rather than `2.2.4` shipped with Apache Syncope, add the following property to your own project's root `pom.xml`:

```
<properties>
   ...
   <connid.database.version>2.2.5-SNAPSHOT</connid.database.version>
</properties>
```

Non-predefined connector bundle

If the needed connector bundle is not in the predefined set as shown above, you will need to add a new property into your own project's root `pom.xml`:

```
<properties>
   ...
   <my.new.connector.version>1.0.0</my.new.connector.version>
</properties>
```

then change the `maven-dependency-plugin` configuration both in `core/pom.xml` and `console/pom.xml` from

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <inherited>true</inherited>
  <executions>
    <execution>
      <id>set-bundles</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>copy</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

to

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <inherited>true</inherited>
  <configuration>
    <artifactItems>
      <artifactItem>
        <groupId>my.new.connector.groupId</groupId>
        <artifactId>my.new.connector.artifactId</artifactId>
        <version>${my.new.connector.version}</version>
      </artifactItem>
    </artifactItems>
  </configuration>
  <executions>
    <execution>
      <id>set-bundles</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>copy</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Run-time

Connector bundles can be added or replaced at run-time by performing the following steps:

1. Download the required connector bundle JAR file;
2. Copy the downloaded JAR file into one of configured ConnId locations, typically the `bundles` directory where the other connector bundles are already available.

4.6.11. E-mail Configuration

The `mail.properties` holds the configuration options to enable the effective delivery of notification e-mails:

- `smtpHost` - the mail server host, typically an SMTP host;
- `smtpPort` - the mail server port;
- `smtpUser` - (optional) the username for the account at the mail host;
- `smtpPassword` - (optional) the password for the account at the mail host;
- `smtpProtocol` - the mail protocol;
- `smtpEncoding` - the default encoding to use for MIME messages;
- `smtpConnectionTimeout` - the connection timeout value in milliseconds, to the mail host;
- `mailDebug` - when `true`, enable the debugging of email, including the handshake, authentication, delivery and disconnection.

In order to make the changes to `mail.properties` effective, the Java EE container needs to be restarted.

Be sure to provide a sensible value for the `notificationjob.cronExpression` configuration parameter, otherwise the notification tasks will not be triggered; see below for details.

4.6.12. Keystore

A Java Keystore is a container for authorization certificates or public key certificates, and is often used by Java-based applications for encryption, authentication, and serving over HTTPS. Its entries are protected by a keystore password. A keystore entry is identified by an alias, and it consists of keys and certificates that form a trust chain.

A keystore is currently required by the SAML 2.0 Service Provider extension in order to sign and / or encrypt the generated SAML 2.0 requests.

While a sample keystore is provided, it is **strongly** recommended to setup a production keystore; in the following, a reference procedure for this is reported.

The procedure below is not meant to cover all possible options and scenarios for generating a keystore, nor to provide complete coverage of the `keytool` command.

Create new keystore

```
keytool -genkey \
  -keyalg RSA \
  -keysize 2048 \
  -alias saml2sp \
  -dname "CN=SAML2SP,OU=Apache Syncope, O=The ASF, L=Wilmington, ST=Delaware, C=US" \
  -keypass akyepass \
  -storepass astorepass \
  -storetype JKS \
  -keystore saml2sp.jks
```

This command will create a keystore file with name `saml2sp.jks` in the execution directory, containing a new 2048-bit RSA key pair, under the specified alias (`saml2sp`); password values for `keypass` and `storepass` are also set.

Create new CSR

```
keytool -certreq \
  -alias saml2sp \
  -keyalg RSA \
  -file certreq.pem \
  -keypass akyepass \
  -storepass astorepass \
  -storetype JKS \
  -keystore saml2sp.jks
```

This command will create a CSR file with name `certreq.pem` in the execution directory, within the keystore generated above.
The generated CSR file can be sent to a Certificate Authority (CA) to request the issuance of a CA-signed certificate.

Have the CSR signed by a Certificate Authority (CA)

This step cannot be automated, and is definitely out of the scope of the this document.

Before proceeding, it is fundamental to have ready the root / intermediate CA certificate(s) and the signed certificate.

Import the certificates into the keystore

```
keytool -import \
  -alias root \
  -file cacert.pem \
  -keypass akyepass \
  -storepass astorepass \
  -storetype JKS \
  -keystore saml2sp.jks
```

This command will import the root / intermediate CA certificate(s) from the `cacert.pem` file into the keystore generated above.

```
keytool -import \
  -alias saml2sp \
  -file cert.pem \
  -keypass akyepass \
  -storepass astorepass \
  -storetype JKS \
  -keystore saml2sp.jks
```

This command will import the signed certificate from the `cert.pem` file into the keystore generated above.

Finalize

The keystore file saml2sp.jks must be now placed in the configuration directory; the relevant part of the saml2sp-agent.properties file should be:

```
keystore.name=saml2sp.jks
keystore.type=jks
keystore.storepass=astorepass
keystore.keypass=akyepass
sp.cert.alias=saml2sp
```

4.6.13. Configuration Parameters

Most run-time configuration options are available as parameters and can be tuned either via the admin console, CLI or barely invoking the REST layer through curl:

- password.cipher.algorithm - which cipher algorithm shall be used for encrypting password values; supported algorithms include SHA-1, SHA-256, SHA-512, AES, S-MD5, S-SHA-1, S-SHA-256, S-SHA-512 and BCRYPT; salting options are available in the security.properties file;
- jwt.lifetime.minutes - validity of JSON Web Token values used for authentication (in minutes);
- notificationjob.cronExpression - cron expression describing how frequently the pending notification tasks are processed: empty means disabled;

  Restarting the Java EE container is required when changing value for this parameter.

- notification.maxRetries - how many times the delivery of a given notification should be attempted before giving up;

  Restarting the Java EE container is required when changing value for this parameter.

- token.length - the length of the random tokens that can be generated as part of various workflow processes, including password reset;
- token.expireTime - the time after which the generated random tokens expire;
- selfRegistration.allowed - whether self-registration (typically via the enduser application) is allowed;
- passwordReset.allowed - whether the password reset feature (typically via the enduser application) is allowed;
- passwordReset.securityQuestion - whether the password reset feature involves security questions;
- authentication.attributes - the list of attributes whose values can be passed as login name for authentication, defaults to username; please note that the related plain schemas must impose the unique constraint, for this mechanism to work properly;
- authentication.statuses - the list of workflow statuses for which users are allowed to authenticate;

> ⚠ Suspended Users are anyway not allowed to authenticate.

- `log.lastlogindate` - whether the system updates the `lastLoginDate` field of users upon authentication;
- `tasks.interruptMaxRetries` - how many attempts shall be made when interrupting a running task;
- `return.password.value` - whether the hashed password value shall be returned when reading users;
- `identity.recertification.day.interval` - number of days between identity recertifications.

Besides this default set, new configuration parameters can be defined to support custom code.

4.7. Migration from Apache Syncope 1.2

Apache Syncope 2.0 brings several enhancements and new features, compared to the 1.2 release.

For this reason, it is not possible to *update* an existing 1.2 deployment, rather it is necessary to *migrate* the whole configuration, users and roles to a brand new 2.0 deployment.

4.7.1. Preparation

With reference to the Apache Syncope Getting Started Guide, perform the following steps:

1. Install the CLI application
2. Create a new Maven project for Apache Syncope 2.0 and then add the following dependency to `core/pom.xml`

```
<dependency>
  <groupId>org.apache.syncope.core</groupId>
  <artifactId>syncope-core-migration</artifactId>
  <version>2.0.4-SNAPSHOT</version>
</dependency>
```

4.7.2. Migrate configuration

Export configuration from 1.2

First, export the configuration from the 1.2 deployment via

```
curl -X GET -u username:password -o content.xml
protocol://host:port/syncope/rest/configurations/stream
```

where `username`, `password`, `protocol`, `host` and `port` reflect the Java EE container installation for the 1.2 deployment.
The configuration of the 1.2 deployment is now in the `content.xml` file.

Obtain configuration file for 2.0

Now process the exported configuration of the 1.2 deployment to obtain a basic 2.0 configuration, by invoking the CLI as follows:

On GNU / Linux - Mac OS X

```
./syncopeadm.sh migrate --conf /path/to/content.xml /dest/path/to/MasterContent.xml
```

On Windows

```
syncopeadm.bat migrate --conf \path\to\content.xml \dest\path\to\MasterContent.xml
```

The result of this invocation is the generated `MasterContent.xml` file and possibly an output message such as the following:

```
You are running: migrate --conf /path/to/content.xml /dest/path/to/MasterContent.xml

Virtual items, require manual intervention:
<?xml version='1.0' encoding='UTF-8'?><dataset>
  <VirSchema key="virtualdata"/>
  <VirSchema key="virtualPropagation"/>
  <VirSchema key="rvirtualdata"/>
  <VirSchema key="mvirtualdata"/>
  <VirSchema READONLY="1" key="virtualReadOnly"/>
  <MappingItem extAttrName="name" mapping_id="1" intAttrName="virtualdata"
               mandatoryCondition="type == 'F'" password="0" purpose="BOTH"/>
  <MappingItem password="0" mapping_id="11" extAttrName="givenname"
               intAttrName="virtualReadOnly" mandatoryCondition="false"
purpose="BOTH"/>
  <MappingItem password="0" mapping_id="11" extAttrName="givenname"
               intAttrName="virtualPropagation" mandatoryCondition="false"
purpose="BOTH"/>
  <MappingItem extAttrName="businessCategory" mapping_id="1"
               intAttrName="rvirtualdata" mandatoryCondition="false" password="0"
purpose="BOTH"/>
  <MappingItem mapping_id="17" password="0" extAttrName="USERNAME"
               intAttrName="virtualdata" mandatoryCondition="false" purpose="BOTH"/>
  <MappingItem mapping_id="17" password="0" extAttrName="SURNAME"
               intAttrName="virtualPropagation" mandatoryCondition="false"
purpose="BOTH"/>
</dataset>



 - Migration completed; file successfully created under
/dest/path/to/MasterContent.xml
```

Virtual schemas and associated mapping cannot be automatically migrated: take note of the message above for further operations.

Finalize configuration for 2.0

After putting the generated `MasterContent.xml` file under the `core/src/test/resources/domains` folder in the new 2.0 Maven project, build and start in embedded mode, while always watching the log files under:

- `core/target/log/`
- `console/target/log/`
- `enduser/target/log/`

If errors are found, make appropriate corrections into `core/src/test/resources/domains/MasterContent.xml` - this might involve migrating custom classes originally developed for 1.2 into their respective 2.0 counterparts.

When no exceptions are reported in the logs, log into the admin console and check if all configuration items (schema definitions, external resources, notifications, ...) were correctly migrated. If anything is missing, take care to re-add them manually.

If using delegated administration under 1.2, reconstruct roles and entitlements under the new security model.

Now, define the virtual schema and mapping items according to the output message obtained above when invoking the CLI.

> ⚠ If making modifications via the admin console, do not forget to export the updated configuration before shutting down the embedded mode, then use the downloaded file to update `core/src/test/resources/domains/MasterContent.xml`.

Finally, verify that all operations (create, update, delete, propagate, sync / push) about users and roles used in the 1.2 deployment are working fine (create, update, delete, propagate, pull / push) with users and groups in the 2.0 Maven project.

When everything works as expected, export the updated configuration before shutting down the embedded mode and use the downloaded file to update both `core/src/main/resources/domains/MasterContent.xml` and `core/src/test/resources/domains/MasterContent.xml`.

4.7.3. Migrate users and roles

After deploying the 2.0 Maven project into one of supported Java EE containers, with internal storage set to one of supported DBMSes, ensure that the 1.2 deployment's internal storage DBMS is reachable.

The steps below are to be performed on the 2.0 deployment.

Define migration AnyTypeClass

Create the following plain schemas:

1. `migrationCipherAlgorithm` - string, read-only

2. `migrationResources` - string, multi-value, read-only
3. `migrationMemberships` - string, multi-value, read-only

Then, define the `migration` AnyTypeClass and assign the three plain schemas above.

Create migration Connector

Create an instance of the Scripted SQL bundle:

1. set connection details to the 1.2 deployment's internal storage DBMS
2. download the Groovy scripts and configure accordingly
3. assign the `SEARCH` and `SYNC` capabilities

Create migration External Resource and Mapping

Create an External Resource for the Connector created above, and set the provisioning rules for:

- `USER` as `__ACCOUNT__`, with at least the following mapping:

Internal Attribute	External Attribute	Other
`username`	`username`	flagged as remote key, mandatory, purpose: `PULL`
`password`		flagged as password, mandatory, purpose: `PULL`
`migrationCipherAlgorithm`	`cipherAlgorithm`	mandatory, purpose: `PULL`
`migrationResources`	`__RESOURCES__`	purpose: `PULL`

- `GROUP` as `__GROUP__`, with at least the following mapping:

Internal Attribute	External Attribute	Other
`name`	`name`	flagged as remote key, mandatory, purpose: `PULL`
`migrationResources`	`__RESOURCES__`	purpose: `PULL`
`migrationMemberships`	`__MEMBERSHIPS__`	mandatory, purpose: `PULL`

More attributes should be added to the mapping information in order to pull values from the 1.2 deployments.

Setup migration Pull Task

Setup a pull task for the External Resource created above, set it for `FULL_RECONCILIATION` mode and configure the MigrationPullActions class among PullActions.

Migrate

Before actual migration, use the admin console features to explore the External Resource and check that all expected information is reported.

Another check to perform is to run the pull task set up above with the DryRun option and watch the execution results.

Finally, execute the pull task and check the execution results.

> 💡 If the number of users and roles to import from the 1.2 deployment is high, it is suggested to change the pull mode to `FILTERED_RECONCILIATION` for a relevant subset of entities to migrate, check the results and eventually switch back to `FULL_RECONCILIATION`.