Home

Agate Documentation

Agate is a web application that offers users related services to the OBiBa software stack (see also General Presentation section): user authentication, user profile management, user notifications. The following guide intend to provide the information and instructions necessary to install, customize, operate and use Agate.

Agate Server Documentation

The Agate Server Administrator Guide consists of two different manuals intended for system administrators:

- Agate Server Installation Guide which provides detailed instructions on how to install Agate server,
- Agate Server Configuration Guide which provides post-install instructions.

Agate Web Application User Guide

The Agate Web Application User Guide details how to enter users, groups and applications into Agate; it is intended for users of Agate (with various permission levels).

Agate Python Client Documentation

The Agate Python Client User Guide also consists of two manuals, mainly intended for system administrators willing to automate tasks:

- Agate Python Client Installation Guide which provides the instruction to install this client,
- Agate Python Commands which provides details on the command line tools.

General Presentation

Targeted at individual studies and study consortia, OBiBa software stack (Opal, Mica2 etc.) provides a software solution for epidemiological data management, analysis and publication. While Opal--the core data warehouse application--provides all the necessary tools to import, transform and describe data, Mica provides everything needed to build personalized web data portals and publish content of research activities of both studies and consortia. Based on the content defined in Mica, Drupal is the prefered platform to build your personalized web portal.

Agate is the OBiBa's central authentication server which intends to be easy to install and to use. Agate centralizes also some user related services such as profile management, and a notification system using emails.

Users, Groups and Applications

Overview

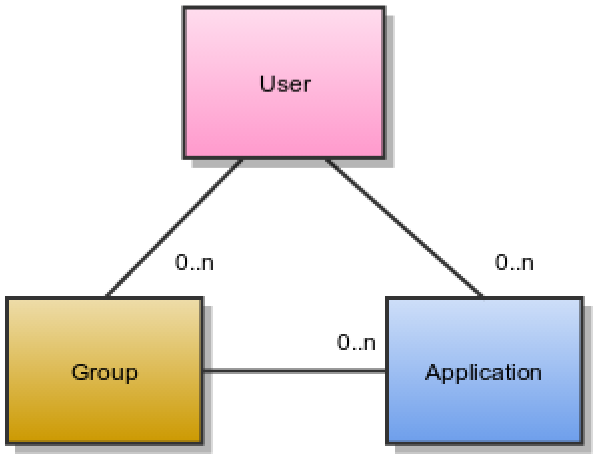
Summary

Agate is a central user directory that can be used by external applications to:

- authenticate users,
- send notification emails.

Domain

The following diagram describes the domain handled by Agate. Each entity of this domain can be edited individually in the Agate Web Application administration interface.



User

A user is described by some properties. Among these properties, the user name and email must be unique in the system: when signing in, a user can provide its name or email. The authentication is done by providing a password, which is stored in the Agate database in a digested form.

A user can belong to some groups.

A user can have access to some applications. If no application is provided, the user can only access to Agate. Otherwise, listed applications will have the user authenticated by Agate.

Group

A group is uniquely identified by its name. A group can be associated to one or applications.

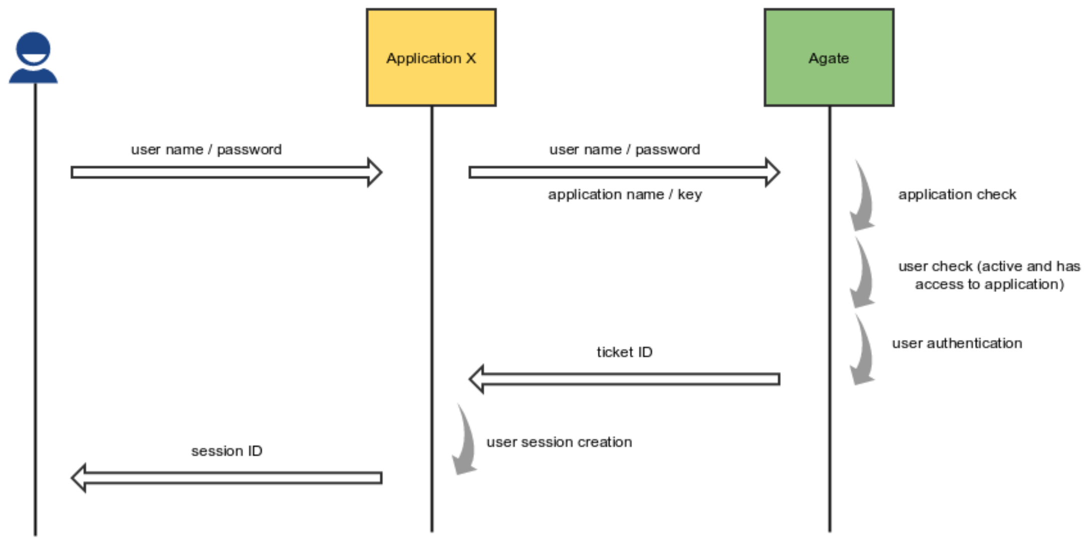
Members of a group can have access to the applications associated to it.

Application

An application has a name and a key. Each time an external application wants to use the services of Agate, it must provide in the request its name and key. This allows Agate to check the validity of the actions to be performed and the information to be returned.

Authentication Flow

When a user tries to sign-in an application X, this application delegates the user authentication to Agate. If successful, a ticket is created in Agate (to track user activity) and user session local to the application X is created. This local user session allows the application to not query Agate each time user authorization check is requested.



Architecture, Servers and Clients

Overview

- Summary
- Agate Server
- Mica Server
- Opal Server
- Drupal Server

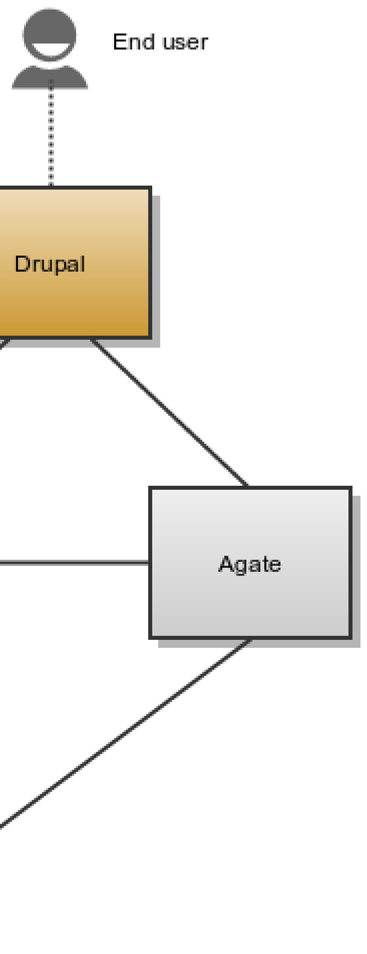
Summary

The architecture of Mica is split in several servers:

- Mica server: holds the domain and controls what is to be published,
- Opal server: holds the data with their dictionary and provide statistics services,
- Agate server: user directory for data access requests management.
- Drupal server: the web portal front-end using Mica server as its source of published documents and Agate server as its user directory.

Mica, Opal and Agate are applications developed by OBiba. OBiBa also provides extensions for the Drupal application. Each of these OBiBa servers expose web services to allow easy interconnection. The *Mica web portal* is the final application which leverages each server specific domain and functionalities in one.

The following diagram shows how these servers are linked together:



Agate Server

Agate application is used for:

- having a user directory shared between OBiBa's applications,
- having centralized services such as profile management and email notifications.

Mica Server

Mica application is used for:

- defining and publishing network, study and dataset catalogues,
- search for variables.

Installation and configuration guides can be found in the section Mica Server Administrator Guide.

Editors and reviewers of the Mica web portal content can access to the web interface of this server as described in the Mica Web Application User Guide.

Mica server is a client of Opal and Agate servers.

Opal Server

Opal application is used for:

- defining data dictionaries (variables),
- storing data,
- providing data summary statistics.

Opal offers well established security controls, allowing to NOT expose individual-level data. Note also that the Opal server is only accessed by the

Mica server, reducing the risk of data compromisation from a malicious end user.

Installation and configuration guides can be found in the Opal Server Administrator Guide.

Mica expects at least one Opal server when some datasets are defined. Additional Opal servers can also be identified to access to distributed datasets.

Drupal Server

Drupal is a content management system, i.e. an application allowing to build fully customizable web portals. Drupal can be extended by modules and themes: Mica and Agate modules have been developed to access to the services of these servers. Drupal server is therefore a client of Mica and Agate servers.

Installation and configuration guides about Drupal as a Mica client can be found in the Mica Drupal Client User Guide documentation.

Agate Server Administrator Guide

Contents of this Guide

Introduction

This guide is for whoever will set up Agate--typically, a system administrator.

The guide covers hardware and software requirements and includes procedures for deploying Agate on a server.

When requirements are met, administrators can follow:

1. the Agate Server Installation Guide,
2. and the Agate Server Configuration Guide.

Agate server package is available in different format:

Type	Package Repository
Debian	Debian Repository
RPM	RPM Repository
Zip	Zip Repository
Docker	Docker Repository

Requirements

Server Hardware Requirements

Component	Requirement
CPU	Recent server-grade or high-end consumer-grade processor
Disk space	8 Gb or more.
Memory (RAM)	Minimum: 4 GB Recommended: >4 GB

Server Software Requirements

Software	Suggested version	Download Link	Usage
Java	>= 1.8.x	Java Oracle Download	Java runtime environment
MongoDB	>= 2.4.x	MongoDB downloads	Database engine

While Java is required by Agate server application, MongoDB can be installed on another server.

Agate Server Installation Guide

Contents of this Guide

Introduction

The aim of this guide is to explain in details how to install Agate server application.

Installing Agate

Agate is distributed as a Debian package and as a zip file. Installing Debian package is recommended on Debian-like Linux systems.

The resulting installation has default configuration that makes Agate ready to be used (as soon as a MongoDB server is available). Once installation is done, see Agate Server Configuration Guide.

Installation of Agate Debian package (recommended)

Agate is available as a Debian package from OBiBa Debian repository.

Download Agate Debian package

To proceed installation, do as follows:

1. **Install Debian package**. Follow the instructions in the repository main page for installing Agate.

2. **Manage Agate Service**: after package installation, Agate server is running: see how to manage the Service.

Installation of Agate RPM package (recommended on RPM-based Linux distributions)

Agate is available as a RPM package from OBiBa RPM repository.

Download Agate RPM package

To proceed installation, do as follows:

1. **Install RPM package**. Follow the instructions in the RPM repository main page for installing Agate.

2. **Manage Agate Service**: after package installation, Agate is running: see how to manage the Service.

Installation of Agate Zip distribution

Agate is also available as a Zip file.

Download Agate Zip package

To install Agate zip distribution, proceed as follows:

1. **Download Agate distribution**
2. **Unzip the Agate distribution**. Note that the zip file contains a root directory named `agate-x.y.z-dist` (where *x*, *y* and *z* are the major, minor and micro releases, respectively). You can copy it wherever you want. You can also rename it.

3. **Create an AGATE_HOME environment variable**
4. **Separate Agate home from Agate distribution directories (recommended)**. This will facilitate subsequent upgrades.

   Set-up example for Linux

   ```
   mkdir agate-home
   cp -r agate-x-dist/conf agate-home
   export AGATE_HOME=`pwd`/agate-home
   ./agate-x-dist/bin/agate
   ```

5. **Launch Agate**. This step will create/update the database schema for Agate and will start Agate: see Regular Command.

For the administrator accounts, the credentials are "administrator" as username and "password" as password. See User Directories Configuration to change it.

Upgrading Agate

There are two way of upgrading Agate, the one you use depends on the way you installed Agate in the first place.

Upgrading Debian Package

If you installed Agate via the Debian package, you may update it using the command:

```
apt-get install agate
```

Manually upgrade from Agate Zip distribution

Follow the Installation of Agate Zip distribution above but make sure you don't overwrite your a*gate-home* directory.

Configuring Agate

See Agate Server Configuration Guide.

Executing Agate

Launching Agate Server

Service

When Agate is installed through the Debian package, Agate server can be managed as a service.

Service Environment

Options for the Java Virtual Machine can be modified if Agate service needs more memory. To do this, modify the value of the environment variable `JAVA_ARGS` in the file `/etc/default/agate`.

Service Script

Main actions on Agate service are: `start`, `stop`, `status`, `restart`. For more information about available actions on Agate service, type:

```
service agate help
```

Service Logs

The Agate service log files are located in `/var/log/agate` directory.

Manually

The Agate server can be launched from the command line. The environment variable `AGATE_HOME` needs to be setup before launching Agate manually.

Command Environment

Configure the following environment variables:

Environment variable	Required	Description
`AGATE_HOME`	yes	Path to the Agate "home" (configuration and file system) directory. Note: `AGATE_HOME` should point to the location (the parent) of the `conf` directory.

`JAVA_OPTS`	no	Options for the Java Virtual Machine.
		
		For example:
		`-Xmx4096m -XX:MaxPermSize=256m`
		
		To change the defaults update:
		`bin/agate` or `bin/agate.bat`

Regular Command

Make sure Command Environment is setup and execute the command line (`bin` directory is in your execution `PATH`)):

```
agate
```

Executing this command upgrades the Agate server and then launches it.

Command Logs

The Agate server log files are located in `AGATE_HOME/logs` directory. If the `logs` directory does not exist, it will be created by Agate.

Using Agate

Accessing Agate from the Web

To access Agate with a web browser the following urls may be used (port numbers may be different depending on HTTP Server Configuration):

- `http://<servername>:8081` will provide a connection without encryption,
- `https://<servername>:8444` will provide a connection secured with ssl.

See Agate Web Application User Guide for more details.

Installation Troubleshooting

If you encounter an issue during the installation and you can't resolve it, please report it in our Agate Issue Tracker.

Reporting errors

Agate logs can be found in `/var/log/agate`. If the installation fails, always refer to this log when reporting an error.

Agate Server Configuration Guide

Contents of this Guide

Prerequisites

Agate server is installed, following the instructions described in Agate Server Installation Guide.

Agate Server Configuration Files

Agate has some configuration files that allows fine tuning of your Agate server. See Agate Configuration Files documentation.

Reverse Proxy Configuration

Agate server can be accessed through a reverse proxy server.

Apache

Example of Apache directives that:

- redirects HTTP connection on port 80 to HTTPS connection on port 443
- refines organization's specific certificate and private key.

```
<VirtualHost *:80>
    ServerName agate.your-organization.org
    ProxyRequests Off
    ProxyPreserveHost On
    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
    RewriteEngine on
    ReWriteCond %{SERVER_PORT} !^443$
    RewriteRule ^/(.*) https://agate.your-organization.org:443/$1 [NC,R,L]
</VirtualHost>
<VirtualHost *:443>
    ServerName agate.your-organization.org
    SSLProxyEngine on
    SSLEngine on
    SSLProtocol All -SSLv2 -SSLv3
    SSLHonorCipherOrder on
    # Prefer PFS, allow TLS, avoid SSL, for IE8 on XP still allow 3DES
    SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384
EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+AESG CM EECDH
EDH+AESGCM EDH+aRSA HIGH !MEDIUM !LOW !aNULL !eNULL !LOW !RC4 !MD5 !EXP
!PSK !SRP !DSS"
    # Prevent CRIME/BREACH compression attacks
    SSLCompression Off
    SSLCertificateFile /etc/apache2/ssl/cert/your-organization.org.crt
    SSLCertificateKeyFile
/etc/apache2/ssl/private/your-organization.org.key
    ProxyRequests Off
    ProxyPreserveHost On
    ProxyPass / https://localhost:8444/
    ProxyPassReverse / https://localhost:8444/
</VirtualHost>
```

Agate Configuration Files

Overview

Summary

The following configuration files are available in the **`AGATE_HOME/conf`** directory.

File	Description
shiro.ini	The user authentication file.
application.yml	The main configuration file to be edited before starting Agate server.
logback.xml	The logs configuration file (for advanced settings).

Agate Server Main Configuration File

The file **AGATE_HOME/conf/application.yml** is to be edited to match your server needs. This file is written in YAML format allowing to specify a hierarchy within the configuration keys. The YAML format uses indentations to express the different levels of this hierarchy. The file is already pre-filled with default values (to be modified to match your configuration), just be aware that you should not modify the indentations. In the following documentation, the configuration keys will be presented using the *dot-notation* (levels are separated by dots) for readability.

HTTP Server Configuration

Agate server is a web application and as such, you need to specify on which ports the web server should listen to incoming requests.

Property	Description
`server.port`	HTTP port number. Generally speaking this port should not be exposed to the web. Use the https port instead.
`server.host`	Web server host name.
`https.port`	HTTPS port number.

MongoDB Server Configuration

Agate server will store its data (system configuration, networks, studies, datasets, etc.) in a MongoDB database. You must specify how to connect to this database.

Property	Description
`mongodb.url`	MongoDB host and port names using the format: `host:port`
`mongodb.databaseName`	Name of the Agate server database in MongoDB. If it does not exist it will be automatically created.
`mongodb.username`	User name for connection to MongoDB database.
`mongodb.password`	User password for connection to MongoDB database.
mongodb.authSource	The name of the authentication database.
mongodb.options	Read Connection Options to learn more. Do not include the ***uri.*** in the URL.

By default MongoDB does not require any user name, it is highly recommended to configure the database with a user. This can be done by enabling the Client Access Control procedure.

Follow these steps to enable the Client Access Control on your server:

- create a user with the proper roles on the target databases
- restart the MongoDB service with Client Access Control enabled

> Once the MongoDB service runs with Client Access Control enabled, all database connections require authentication.

MongoDB 3.x Settings

The default authentication mechanism has changed since MongoDB 3.0 and the driver used by Agate does not support (yet) this new mechanism. According to the documentation about the authentication mechanisms:

> *MongoDB uses the* SCRAM-SHA-1 *as the default challenge and response authentication mechanism. Previous versions used* M ONGODB-CR *as the default.*

Then before creating the user, restore MONGODB-CR as the default mechanism with the following script:

```
use admin
var schema = db.system.version.findOne({"_id" : "authSchema"})
schema.currentVersion = 3
db.system.version.save(schema)
```

MongoDB User Creation Example

The example below creates the *agateadmin* user for *agate* database:

MongoDB Console

```
use admin
db.createUser(
  {
    user: "agateadmin",
    pwd: "agateadmin",
    roles: [
      {
        "role" : "readWrite",
        "db" : "agate"
      },
      {
        "role" : "dbAdmin",
        "db" : "agate"
      },
      {
          "role": "clusterMonitor",
          "db": "admin"
      },
      {
          "role": "readAnyDatabase",
          "db": "admin"
      }
    ]
  }
)
```

Here is the required configuration snippet in */etc/agate/application.yml* for the above user:

```
spring:
  data:
    mongodb:
      uri:
mongodb://agateadmin:agateadmin@localhost:27017/agate?authSource=admin
```

Agate requires either *clusterMonitor* or *readAnyDatabase* role on the *admin* database for validation operations. The first role is useful for a cluster setup and the latter if your MongoDB is on a single server.

Mail Server Configuration

Agate offers notification services to its registered applications. For that purpose a mail server must be configured.

Property	Description
`spring.mail.host`	Mail server host name or IP.
`spring.mail.port`	Mail server port.
`spring.mail.user`	Mail server user name.
`spring.mail.password`	Mail server user password.
`spring.mail.protocol`	Communication protocol.
`spring.mail.tls`	Enable communication encryption with the SMTP mail server.
`spring.mail.auth`	Enable SMTP authentication.
`spring.mail.from`	From email address.

Shiro Configuration

Shiro is the authentication and authorization framework used by Agate. There is a minimum advanced configuration that can be applied to specify how Shiro will hash the password. In practice this only applies to the users defined in the *shiro.ini* file. Default configuration is usually enough.

Property	Description
`shiro.password.nbHashIterations`	Number of re-hash operations.
`shiro.password.salt`	Salt to be applied to the hash.

reCaptcha configuration

Agate uses the reCaptcha service in the sign up page in order to avoid spam and abuse. reCaptcha is a free web service but you need to register in order to get the necessary credentials to use it.

Property	Description
recaptcha.verifyUrl	URL of the reCaptcha API endpoint used to verify the captcha challenge.
recaptcha.secret	Secret used by agate to communicate with reCaptcha.
client.reCaptchaKey	Public reCaptcha key used by the client, i.e. web browser, displaying the captcha challenge.

Other Configurations

Configurations not listed above are available in the main configuration. They are for internal use only.

Agate Server Logs Configuration File

The file is **AGATE_HOME/conf/logback.xml** allows to specify how Agate server should log it's internal activity. Remember that outputing too much logs could affect the performance of your server. Default log file is **AGATE_HOME/log/agate.log**. Agate server needs to be restarted for the changes to be effective. Default configuration is usually enough.

Upgrading to Agate 1.3.x

Upgrade will affect the configuration files and database access.

As a general rule, always backup the configuration files and the databases before upgrading. Make sure also you can restore them!

Agate configuration file

Some configurations changed in file */etc/agate/application.yml*

Configurations with prefix "mongodb." are useless, you can delete them. Now, we need the property "spring.data.mongodb.uri". Exemple for the above user :

```
spring:
  data:
    mongodb:
      uri:
mongodb://agateadmin:agateadmin@localhost:27017/agate?authSource=admin
```

User Directories Configuration

Overview

- Summary
- User Directories
    - File Based User Directory

Summary

The security framework that is used by Agate for authentication, authorization etc. is Shiro. Configuring Shiro for Agate is done via the file **`AGATE_HOME/conf/shiro.ini`**.

More information about how to configure Shiro using the `ini` file can be found here.

Default configuration is a static user `'administrator'` with password `'password'` (or the one provided while installing Agate Debian package).

Remember to change default administrator password if you did not installed Agate with the Debian package!

User Directories

By default Agate server has several built-in user directories:

- a file-based user directory (`shiro.ini` file),
- the internal user directory persisted in the MongoDB database.

Although it is possible to register some additional user directories, this practice is currently not recommended.

In the world of Shiro, a user directory is called a *realm*.

File Based User Directory

The file-based user directory configuration file **`AGATE_HOME/conf/shiro.ini`**.

It is not recommended to use this file-based user directory. It is mainly dedicated to define a default system super-user and a password for the anonymous user.

For a better security, user passwords are encrypted with a one way hash such as sha256.

The example `shiro.ini` file below demonstrates how encryption is configured.

conf/shiro.ini

```
# =======================
# Shiro INI configuration
# =======================

[main]
# Objects and their properties are defined here,
# Such as the securityManager, Realms and anything else needed to build the
SecurityManager


[users]
# The 'users' section is for simple deployments
# when you only need a small number of statically-defined set of User
accounts.
#
# Password here must be encrypted!
# Use shiro-hasher tools to encrypt your passwords:
#   DEBIAN:
#     cd /usr/share/agate/tools && ./shiro-hasher -p
#   UNIX:
#     cd <AGATE_DIST_HOME>/tools && ./shiro-hasher -p
#   WINDOWS:
#     cd <AGATE_DIST_HOME>/tools && shiro-hasher.bat -p
#
# Format is:
# username=password[,role]*
administrator =
$shiro1$SHA-256$500000$dxucP0IgyO99rdL0Ltj1Qg==$qssS60kTC7TqE61/JFrX/OEk0j
sZbYXjiGhR7/t+XNY=,agate-administrator

[roles]
# The 'roles' section is for simple deployments
# when you only need a small number of statically-defined roles.
# Format is:
# role=permission[,permission]*
agate-administrator = *
```

Passwords must be encrypted using shiro-hasher tools (included in Agate tools directory):

Password encryption for Linux

```
cd /usr/share/agate/tools
./shiro-hasher -p
```

Notification Emails Configuration

Overview

Summary

Agate offers a notification emails service to the registered applications. Based on email templates, an application can request Agate to send emails to one or more of its users. These templates are defined in the **`AGATE_HOME/conf/templates`** directory. Agate is using email templates for sending its notifications (email confirmation, reset password etc.).

The email templates specific to an application are located in the directory **AGATE_HOME/conf/templates/<application name>**.

The template engine used for building the email messages is thymeleaf.

Agate Web Application User Guide

Contents of this Guide

Introduction

The *Agate Web Application* is the administration web interface of the Agate server. It is NOT the end-user web portal and therefore firewall policies can (or should) be applied to restrict access to administrators or content editors.

See the Agate Domain presentation page for a detailed description of the type of *documents* that can be edited through this web interface.

The following manuals are available:

- Users Management: add, edit users
- Groups Management: add, edit groups
- Applications Management: add, edit applications
- Tickets Management: track user sessions
- Administering Agate Server: configure server settings

Requirements

This web interface is a javascript application requiring a modern web browser. There is no requirement regarding the operating system.

Users Management

Overview

Summary

The user pages are: the list of users page, the list of users requesting to join page, and user view and edit pages. See also User section in domain documentation.

Permissions

Users with the `agate-administrator` role have access to these pages.

Operations

Add a user

Agate administrators can create users. The "General information" section contains system defined properties as well as configured attributes defined by the administrator. The "Access" section contains information related to the Role in agate, Groups and Applications for the user. Some user specific attributes can be defined too.

Edit a user

All the information for a user but his user name can be edited.

Delete a user

A user can be deleted.

Reset a user's password

Click the reset password button to send the user, an email with details on how to reset his password.

Approve/Reject a user request

User requests can be approved or rejected. When a user sends a request, it is created with a status *pending*. If the request is rejected the user is removed, otherwise his status becomes *approved*.

Groups Management

Overview

Summary

Users can grouped in groups associated with a list of applications. Members of a group get access to the applications associated with it. See also Group section in domain documentation.

The group pages are: the list of groups page and group view and edit pages.

Permissions

Users with `agate-administrator` role can access these pages.

Operations

Add group

Creates a group defined by a unique name.

Edit group

The description and the list of associated applications can be edited.

Delete group

A group can be deleted if there are no users associated with it.

Applications Management

Overview

Summary

An application is an external system that can use agate as a central authentication system. Once an application is registered in agate, it can use its credentials (name and key) to connect with agate. See also Application section in domain documentation.

The application pages are: the list of applications page and application view and edit pages.

Permissions

Users with `agate-administrator` role can access these pages.

Operations

Add an application

Creates a new application that can access agate with the defined name and key. The application name has to be unique in agate.

Edit an application

Edits an application's properties. The name can not be changed.

Delete an application

An application can be deleted only if there are no groups or users associated with it.

Tickets Management

Overview

Summary

Tickets are used to track the requests done on a specific user by the applications. A ticket is identified by a *token* which is an obscure identifier used by the applications internally.

Permissions

Users with `agate-administrator` role have access to this page.

Operations

Delete ticket

A ticket can be deleted to clear the history of requests done on a specific user by the applications.

Administering Agate Server

Overview

Summary

The *Administration* menu is available to users with the role `agate-administrator`. This menu gives access to server configuration and status.

Properties

Here, you can define the following general configuration variables of Agate:

Name	Description
Name	The name of the organization using this instance of Agate server. It will be used when sending notification emails.
Public URL	Public base URL of the server. It will be used when sending notification emails.
Short term timeout	Ticket expiration timeout in hours.
Long term timeout	Ticket expiration timeout in hours when "remember me" option is selected.
Inactive timeout	User account expiration timeout in days.
Sign up form offers to choose the username	User name will be extracted from user email.

Encryption Keys

This section presents the tool related to the encryption–through HTTPS–of transactions between Agate and its clients by means of a trusted or a self-signed certificate.

In the instruction below, when you are told to cut and paste the content of the certificate, private key or of an `*.pem` file, make sure that you copy **all** content, that is **including** the lines containing "`-----BEGIN XXXXXXXX-----`" and "`-----END XXXXXXXX-----`".

Create a (self-signed) certificate

Go to **Administration > Encryption Keys**, click on the *Add Keys* drop-down under the subsection title Encryption Keys and select *Create*.

1. Click on the *Add Keys* drop-down under the subsection title Encryption Keys.
2. Select *Create.*
3. Fill in the form and click on *Save*.
4. Click on the Download Certificate button under the section title Encryption Keys.

Your certificate (`*.pem` file) should automatically be downloaded on your computer.

Import a certificate

Go to **Administration > Encryption Keys** , click on the *Add Keys* drop-down under the subsection title "Encryption Keys" and select *Import* . Here you may use (1) certificate and (2) private key that you created using third party software *e.g.*, OpenSSL. Note that:

1. Both the certificate and the private key *must* in PEM format *i.e.*, you can read them and the file starts with a `----- BEGIN` [...].

2. You must copy the certificate (or the content of the `*.crt` file) in the public key box and the private key (or the content of the `*.key` file) in the private key box.

In either case, you finish by clicking on *Save*. Finally, in order for the changes to be taken in account you need to restart Agate server.

User Attributes

Additional user attributes can be declared. They will appear in the user form (including sign-up).

Agate Python Client User Guide

Contents of this Guide

Summary

Agate Python client, a command line scripting tool written in Python, enables automation of tasks in a Agate server.

Installation

You can install Agate Python Client via the following two methods:

- use a Python package
- use the Debian package manager

Please read Agate Python Client Installation Guide for more details.

Usage

To get the options of the command line:

```
agate --help
```

This command will display which sub-commands are available. Further, given a subcommand obtained from command above, its help message can be displayed via:

```
agate <subcommand> --help
```

This command will display available subcommands.

Commands

The following subcommands are available:

Command	Description
add-user	Add a new user.
delete-user	Delete a user.
add-group	Add a new group.
delete-group	Delete a group.
add-application	Add a new application.
delete-application	Delete an application.
rest	Request directly the Agate REST API, for advanced users

Requirements

Python 2.x must be installed on the system. See more about Python.

Agate Python Client Installation Guide

Contents of this Guide

Introduction

There are two ways to install the Agate Python Client package on your computer:

- either from the Debian package (Debian-base Linux only),
- or from the Python package (Linux or Windows).

Installing Agate Python Client

Installation of the Debian package (recommended)

Agate Python client is available as a Debian package from OBiBa Debian repository.

Download Agate Python Client Debian package

Then Install the package:

```
sudo apt-get install agate-python-client
```

Some Debian-based systems (Debian Wheezy or Ubuntu Precise releases) only provide an older version of *python-protobuf* package dependency. When executing the agate python client the following error is reported:

```
ImportError: cannot import name enum_type_wrapper
```

This can be fixed manually using the commands:

```
apt-get purge python-protobuf

apt-get install python-pip

pip install protobuf>=2.5.0

apt-get install agate-python-client
```

Installation of the RPM package (recommended for Fedora-based systems)

Agate Python client is available as a RPM package from OBiBa RPM repository.

Download Agate Python Client RPM package

Then Install the package:

```
sudo yum install agate-python-client
```

Installation of the Python package

This type of package is cross-platform (Linux, Windows, Mac).

Install on Linux or Mac

1. Download the most recent version from:

   Download Agate Python Client package
2. Decompress the file and enter the installation folder:

   ```
   tar xvzf agate-python-client-X.XX.tar.gz
   cd agate-python-client-X.XX
   ```

3. Install the package:

   ```
   sudo python setup.py install --record installed_files.lst
   ```

   The '–record' will generate a list of installed files on your system. Since there is no uninstaller, you can use this file to remove the Agate Python Client package. You can do this by executing the following command:

   ```
   $ sudo cat installed_files.lst | xargs rm -rf
   ```

Install on Windows

Using Cygwin

You can install Cygwin, making sure that CURL, Python, gcc are included and follow these steps inside a Cygwin BASH window:

```
cd /usr/lib
cp libcurl.dll.a libcurl.a
cd <your-desired-dir>
curl -C - -O
http://download.obiba.org/agate/stable/agate-python-client-X.XX.tar.gz
tar xzvf agate-python-client-X.XX.tar.gz
cd agate-python-client-X.XX
python setup.py install --record installed_files.lst
```

Using plain Windows tools

This Windows installation is the most complicated one but does not required any third party tools. You are required to do a few manual installations before the package is fully usable. The following steps were tested on a Windows 7.

1. You must have Python installed on your Windows system. Run this installer in case you don't have one.
2. Download the Google protobuf binary and make sure that its containing folder is in your path.
3. Download the protobuf source package containing the setup.py file and follow these steps:

```
unzip protobuf-2.5.0.zip
cd protobuf-2.5.0/python
python setup.py install
```

4. Go to the Python Libs site and download the file ***pycurl-7.19.0.win-amd64-py2.7.exe***
5. Run the installer and follow the instructions until the package is installed
6. Download the http://download.obiba.org/agate/stable/agate-python-client-X.XX.zip and follow these steps:

```
unzip
http://download.obiba.org/agate/stable/agate-python-client-X.XX.zip
cd agate-python-client-X.XX
python setup.py bdist_wininst
cd dist
```

7. Execute the generated installer and follow the instructions (agate-python-client-X.XX.win-amd64.exe)

Testing the Installation

Test the installation by performing a REST call to your Agate server:

```
agate rest /applications --agate <YOUR-AGATE-SERVER-URL>  --user
<YOUR-USER> --password <YOUR-PASSWORD> --json
```

Agate Python Commands

- Add User Command
- Delete User Command
- Add Group Command
- Delete Group Command
- Add Application Command
- Delete Application Command
- REST API Command

Add User Command

Overview

- Synopsis
    - Credentials
    - Options
    - Extras
- Example

Synopsis

Add a new user.

```
agate add-user <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The user name, required and unique.
`--email EMAIL`	The user email, required and unique.
`--upassword UPASSWORD`	The user password, required.
`--first-name FIRST_NAME`	The user first name.
`--last-name LAST_NAME`	The user last name.
`--applications [APPLICATIONS [APPLICATIONS ...]]`	The applications in which the user can sign-in, space separated.
`--groups [GROUPS [GROUPS ...]]`	The groups to which the user belongs, space separated.
`--role ROLE`	The role of the user. Default is "agate-user", which gives only the right to user to access to its own profile. Other possible value is "agate-administrator".
`--status STATUS`	Only active users can sign-in. Default value is "ACTIVE". Other possible values are: "PENDING", "APPROVED" or "INACTIVE".

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Add a new user.

```
agate add-user -ag http://localhost:8081 -u administrator -p password
--name user1 --email user1@example.org --applications mica drupal
```

Delete User Command

Overview

Synopsis

Delete a user.

```
agate delete-user <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The user name, mutually exclusive with email.
`--email EMAIL`	The user email, mutually exclusive with name.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Delete a user.

```
agate delete-user -ag http://localhost:8081 -u administrator -p password
--name user1
```

Add Group Command

Overview

Synopsis

Add a new group.

```
agate add-group <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The group name, required and unique.
`--description DESCRIPTION`	The description of the group, optional.
`--applications [APPLICATIONS [APPLICATIONS ...]]`	The applications in which the users members of the group can sign-in, space separated.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Add a new group.

```
agate add-group -ag http://localhost:8081 -u administrator -p password
--name researchers --applications mica drupal
```

Delete Group Command

Overview

Synopsis

Delete a group (does not delete the users that are members of the group).

```
agate delete-group <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The group name, required.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Delete a group.

```
agate delete-group -ag http://localhost:8081 -u administrator -p password
--name researchers
```

Add Application Command

Overview

Synopsis

Add a new group.

```
agate add-application <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The application name, required and unique.
`--description DESCRIPTION`	The description of the application, optional.
`--key KEY`	The application key, required.
`--redirect REDIRECT`	Callback URL to the application's server, required in the OAuth context.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Add a new application.

```
agate add-application -ag http://localhost:8081 -u administrator -p
password --name someapp --key ABCDEFGH1234
```

Delete Application Command

Overview

Synopsis

Delete a group (does not delete the users that are members of the group).

```
agate delete-application <CREDENTIALS> [OPTIONS] [EXTRA]
```

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--name NAME`	The application name, required.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

Example

Delete a application.

```
agate delete-application -ag http://localhost:8081 -u administrator -p
password --name someapp
```

REST API Command

Overview

Synopsis

This command is for advanced users wanting to directly access to the REST API of Agate server.

```
agate rest ws <CREDENTIALS> [OPTIONS] [EXTRA]
```

Web Service

Options	Descriptions
`ws`	Web service path, for instance: /user/xxx

Credentials

Authentication is done either by username/password credentials.

Options	Description
`--agate AGATE, -ag AGATE`	Agate server base url.
`--user USER, -u USER`	User name. User with appropriate permissions is expected depending of the REST resource requested.
`--password PASSWORD, -p PASSWORD`	User password.

Options

Options	Descriptions
`--method METHOD, -m METHOD`	HTTP method: GET (default), POST, PUT, DELETE, OPTIONS.
`--accept ACCEPT, -a ACCEPT`	Accept header (default is application/json).
`--content-type CONTENT_TYPE, -ct CONTENT_TYPE`	Content-Type header (default is application/json).
`--json, -j`	Pretty JSON formatting of the response.

Extras

Options	Descriptions
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose outut

Example

Get all users.

```
agate rest -ag http://localhost:8081 -u administrator -p password -j /users
```

Agate OAuth2 API

Overview

- Summary

Summary

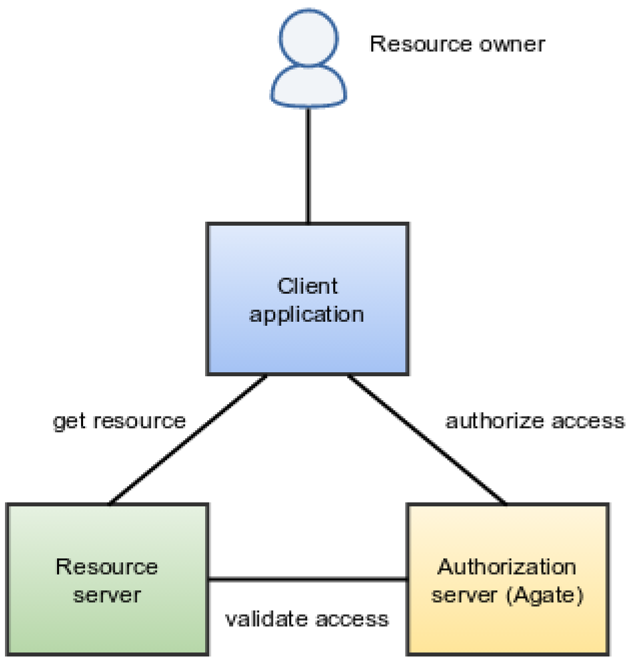
Agate exposes web services that implements the OAuth2 protocol. OAuth2 is an open authorization protocol which enables applications to access each others data. The authorization refers to the fact that these data are accessed on behalf of a resource owner.

For more details, the OAuth2 specifications are available at RFC6749. See also the OpenID Connect specifications built on top of OAuth2.

Roles

The OAuth2 protocol defines several roles:

- the **resource owner** is the person or application that owns the data that is to be shared. In our case a user on Agate could be a resource owner. The resource they own is their data. The resource owner is depicted in the diagram as a person, which is probably the most common situation. The resource owner could also be an application.
- the **resource server** is the server hosting the resources. For instance, Opal or Mica are resource servers.
- the **client application** is the application requesting access to the resources stored on the resource server. These resources are owned by the resource owner. A client application could be an X-ray images analyser that extracts the image data from Opal.
- the **authorization server** is the server authorizing the client application to access the resources of the resource owner: this is the role of Agate. The authorization server and the resource server can be the same server, but it doesn't have to. When Agate is also the resource server, the resource that is accessed is the user profile.



Client ID, Client Secret and Redirect URI

Before a client application can request access to resources on a resource server, the client application must first register with the authorization server associated with the resource server. In Agate this is done by adding the client as a new Application.

The registration is typically a one-time task. Once registered, the registration remains valid, unless the client application registration is revoked by the Agate's administrator.

At registration the client application is assigned a client ID and a client secret (password) by the authorization server. The client ID and secret is unique to the client application on that authorization server. In terms of Agate's domain, the client ID is the application's name and the client secret is the application's key.

During the registration the client needs to provide a redirect URI. This redirect URI is used when a resource owner grants authorization to the client application. When a resource owner has successfully authorized the client application via the authorization server, the resource owner is redirected back to the client application, to the redirect URI.

Scopes

The scopes are space-separated the application IDs, optionally qualified by a permission. As an example, if an application registered in Agate with ID **`foo`** declares the **`read`** permission (= the permission to access to the resource granted by `foo` is read-only), then the authorization scope will be **`foo:read`**. If no action is specified, Agate will assume that `foo` grants full access to the resource. The permissions are specific to the application and it is the responsibility of the resource server to handle them as announced.

Flows

- Authorization Code Grant Flow: when a client application wants access to the resources of a resource owner, hosted on a resource server, the client application must first obtain an authorization code grant from the authorization server (Agate).
- Resource Owner Password Credentials Grant Flow: suitable for clients capable of obtaining the resource owner's credentials (username and password),
- OpenID Connect Flow: when client wants to get the user information from Agate (authorization and resource are the same).

Authorization Code Grant Flow

Overview

Summary

When a client application wants access to the resources of a resource owner, hosted on a resource server, the client application must first obtain an authorization code grant from the authorization server (Agate). The following explains how such a grant is obtained.

Step 1. Authorization

Request

The client application must redirect the user to the Agate authorization page which is:

```
GET https://agate.example.org/ws/oauth2/authorize?<PARAMETERS>
```

The following values should/could be passed as parameters:

Parameter	Description
`client_id`	Client application that will be granted the authorization (required).
`response_type`	The expected value is: `code` (required).
`scope`	Space separated application names (required).
`redirect_uri`	URL to redirect back to (optional, if not specified default client application redirect URI will be used).
`state`	Unique string to be passed back upon completion (optional, recommended).

Agate will redirect the "user-agent" (usually a web browser) to a web page where the resource owner can grant or deny the requested authorizations.

Response

If the resource owner accepts to grant thre requested authorizations to the client application, then the response will consist of a redirect to the provided `redirect_uri` with the following request parameters:

Parameter	Description
`code`	The authorization code.
`state`	The state parameter value that was provided in the request (if any).
`expires_in`	Information about the expiration time (in seconds) before the authorization expires.

The redirect request will then look like:

```
GET
https://client.example.org/redirect?code=AUTHORIZATION_CODE&state=STATE&ex
pires_in=7775999
```

From then, it is the responsibility of the client application to response to this request with a redirect to the relevant client application page.

Errors

The following response errors can be encountered during this step.

```
GET
https://client.example.org/redirect?error=ERROR_CODE&error_description=ERR
OR_MESSAGE
```

Again it is the responsibility of the client application to handle this response.

Error Code	Description
access_denied	When the user refuses to grant the requested authorization.
invalid_scope	The requested scope is not one of the declared resource application scopes.
missing_application_redirect_uri	The client application does not have a default redirect URI. This is a client application definition issue.
invalid_redirect_uri	The provided redirect URI does not starts with the client application's default redirect URI.
server_error	Other errors.

Step 2. Access Token Issuing

Request

The REST endpoint to be used is:

```
POST https://agate.example.org/ws/oauth2/token
```

The form parameters to be sent within the body of the request are:

Parameter	Description
`client_id`	Client application name (required).

`client_secret`	Client application secret key (required).
`grant_type`	The expected value is: `authorization_code` (required).
`code`	The authorization code from the Step 1 (required).
`redirect_uri`	Must match the originally submitted URI (if one was sent).

Response

The response is a JSON object with the following properties:

Property	Description
`access_token`	The access token. Agate provides signed tokens that implement the JSON Web Token specification.
`token_type`	What you can do with this token; in the case of Agate the value for this property is `bearer`.
`expires_in`	Information about the expiration time (in seconds) before the token expires.

An example of response would be:

```
{
    "access_token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlZGl0b3IiLCJpc3MiOiJhZ2F0ZTo1NmZjMzg0MmNj
ZjJjMWM3ZWM1YzVkMTQiLCJpYXQiOjE0NTk0NTg0NTgsImV4cCI6MTQ1OTQ4NzI1OCwianRpIj
oiNTZmZDkxOWFjY2YyYzFjN2VjNWM1ZDE2IiwiYXVkIjpbIm1pY2EiLCJ0b3RvIl0sImNvbnRl
eHQiOnsic2NvcGVzIjpbIm1pY2EiXSwidXNlciI6eyJuYW1lIjoiSnVsaWUiLCJncm91cHMiOl
sibWljYS1lZGl0b3IiXSwiZmlyc3RfbmFtZSI6Ikp1bGllIn19fQ.PqlLSZegdPLM2byp0jsgW
V-XM3Xed8DP4I03kbUUEeo",
    "token_type": "bearer",
    "expires_in": 28799
}
```

Being a JSON Web Token (JWT), the access token can be decoded. There are three parts in a JWT: the header, the payload and the signature. This could give for example:

```
{
    "alg": "HS256"
}
.
{
    "sub": "editor",
    "iss": "agate:56fc3842ccf2c1c7ec5c5d14",
    "iat": 1459458458,
    "exp": 1459487258,
    "jti": "56fd919accf2c1c7ec5c5d16",
    "aud": [
        "mica",
        "client_app"
    ],
    "context": {
        "scopes": [
            "mica"
        ],
        "user": {
            "name": "Julie LaTendresse",
            "groups": [
                "mica-editor"
            ],
            "first_name": "Julie",
            "last_name": "Latendresse"
        }
    }
}
.
[signature]
```

The JWT payload contains some basic details on the subject (in addition to the standard claims). These are available in the `context` object (which is a claim specific to Agate). The properties of the context are:

Property	Description
`user.name`	The user full name for display.
`user.first_name`	The user first name (if any).
`user.last_name`	The user last name (if any).
`user.groups`	The user groups.
`scopes`	Reminder of the scopes associated to the authorization code grant.

Note that this step can be repeated as many times as necessary, using the same authorization code that was granted at step 1.

Errors

When an error is encountered during this step, the JSON object returned contains the description of the error, for example:

```
{
    "error_description":"Authorization with code
'3b1d664fb09407972d4c212081789c6f' does not exist",
    "error":"NoSuchAuthorizationException"
}
```

Step 3. Resource Access

The client application will use the access token as a bearer of resource owner identity to get the resource from the resource server. How the access token should be passed to the resource application is out of the concern of Agate.

Most common practice (this is the case for Opal and Mica) is that the access token is placed in the headers of the HTTP request issued by the client application on the resource server. This can be expressed as a curl command:

```
curl -X GET --header "Authorization: Bearer ACCESS_TOKEN"
http://resource.example.org/some/path
```

Step 4. Access Token Validation

The resource server has received an access token from a client application. Although the access token delivered by Agate is a JWT that contains in its payload all the basic information (subject identification, authorized scopes), it is the responsibility of the resource application to validate this token.

This can be achieved by requesting the REST end point:

```
GET https://agate.example.org/ws/ticket/ACCESS_TOKEN/_validate
```

Note that the resource application must identifies itself in this request. This can be expressed as a curl command:

```
curl -X GET --header "X-App-Auth: Basic `echo -n
"APPLICATION_NAME:APPLICATION_KEY" | base64`"
https://agate.example.org/ws/ticket/ACCESS_TOKEN/_validate
```

The expected response code is 200 (OK), without a response body.

Possible validation errors are:

- application could not be identified,
- access token signature verification has failed,
- access token issuer is not the current Agate instance,
- application is not part of the audience of the access token,
- access token has expired,
- user is not active any more.

Resource Owner Password Credentials Grant Flow

Overview

Summary

The resource owner password credentials grant is suitable for clients capable of obtaining the resource owner's credentials (username and password, typically using an interactive form). This implies that the resource owner has a trust relationship with the client application, such as the device operating system or a highly privileged application.

> Agate's implementation of this flow is very limited. The access token obtained with this flow does not provide authorization to access the resource applications. This flow's main use case is to authenticate the resource owner.

Access Token Issuing

Request

The REST end point to be used is:

```
POST https://agate.example.org/ws/oauth2/token
```

The form parameters to be sent within the body of the request are:

Parameter	Description
`client_id`	Client application name (required).
`client_secret`	Client application secret key (required).
`grant_type`	The expected value is: `password` (required).
`username`	The user name.
`password`	The user password.

Response

The response is a JSON object with the following properties:

Property	Description
`access_token`	The access token. Agate provides signed tokens that implement the JSON Web Token specification.
`token_type`	What you can do with this token; in the case of Agate the value for this property is `bearer`.
`expires_in`	Information about the expiration time (in seconds) before the token expires.

An example of response would be:

```
{
    "access_token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlZGl0b3IiLCJpc3MiOiJhZ2F0ZTo1NmZjMzg0MmNj
ZjJjMWM3ZWM1YzVkMTQiLCJpYXQiOjE0NTk0NTg0NTgsImV4cCI6MTQ1OTQ4NzI1OCwianRpIj
oiNTZmZDkxOWFjY2YyYzFjN2VjNWM1ZDE2IiwiYXVkIjpbIm1pY2EiLCJ0b3RvIl0sImNvbnRl
eHQiOnsic2NvcGVzIjpbIm1pY2EiXSwidXNlciI6eyJuYW1lIjoiSnVsaWUiLCJncm91cHMiOl
sibWljYS1lZGl0b3IiXSwiZmlyc3RfbmFtZSI6Ikp1bGllIn19fQ.PqlLSZegdPLM2byp0jsgW
V-XM3Xed8DP4I03kbUUEeo",
    "token_type": "bearer",
    "expires_in": 28799
}
```

Being a JSON Web Token (JWT), the access token can be decoded. There are three parts in a JWT: the header, the payload and the signature. This could give for example:

```
{
    "alg": "HS256"
}
.
{
    "sub": "editor",
    "iss": "agate:56fc3842ccf2c1c7ec5c5d14",
    "iat": 1459458458,
    "exp": 1459487258,
    "jti": "56fd919accf2c1c7ec5c5d16",
    "aud": [
        "client_app"
    ],
    "context": {
        "user": {
            "name": "Julie LaTendresse",
            "groups": [
                "mica-editor"
            ],
            "first_name": "Julie",
            "last_name": "Latendresse"
        }
    }
}
.
[signature]
```

The JWT payload contains some basic details on the subject (in addition to the standard claims). These are available in the `context` object (which is a claim specific to Agate). The properties of the context are:

Property	Description
`user.name`	The user full name for display.
`user.first_name`	The user first name (if any).
`user.last_name`	The user last name (if any).

`user.groups`	The user groups.

Note that this step can be repeated as many times as necessary, using the same authorization code that was granted at step 1.

Errors

When an error is encountered during this step, the JSON object returned contains the description of the error, for example:

```
{
    "error_description":"Authorization with code
'3b1d664fb09407972d4c212081789c6f' does not exist",
    "error":"NoSuchAuthorizationException"
}
```

OpenID Connect Flow

Overview

Summary

OpenID connect is an extension on top of OAuth2, so the authorization and token enpoints are the same as described in Agate OAuth2 API. Curr ently the OpenID Connect implementation in Agate only supports the authorization code flow.

Step 1. Authorization

This first step is the same as in the one in the Authorization Code Grant Flow: see authorization request and response. The scope to be requested must contain at least `openid` in addition to more specific scopes. Currently the only supported scopes are: `email` and `profile`.

Scope	Description
`openid`	User name (required).
`email`	User email address and whether this email was verified (optional).
`profile`	User first and last names (optional).
`phone`	User phone (not supported).
`address`	User address (not supported).

An example of an OpenID connect authorization request will then look like:

```
GET
https://agate.example.org/ws/oauth2/authorize?client_id=xxx&response_type=
code&scope=openid+email+profile
```

Step 2. ID Token Issuing

This second step is similar to the access token issuing. When the authorization includes the `openid` scope, the response will contain an additional `id_token` in JWT format. An example of the reponse is:

```
{
  "access_token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsImlzcyI6ImFnYXRlOjU3MDUyMjA2ZTRi
MGRlNDNlYzE5NzM2YiIsImlhdCI6MTQ2MDA0MTU4NCwiZXhwIjoxNDYwMDcwMzg0LCJqdGkiOi
I1NzA2Nzc3MGU0YjBmZjM3ODJkYmQ2MjIiLCJhdWQiOlsiZHJ1cGFsIl0sImNvbnRleHQiOnsi
c2NvcGVzIjpbIm9wZW5pZCJdLCJ1c2VyIjp7Im5hbWUiOiJKb2hubnkgQi4gR29vZCIsImxhc3
RfbmFtZSI6Ikdvb2QiLCJncm91cHMiOlsibWljYS11c2VyIl0sImZpcnN0X25hbWUiOiJKb2hu
bnkgQi4ifX19.7SblBktnvXaoBFL61Rx_jb6PXXYPr4TFMlyi4ZYP5xE",
  "id_token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsImlzcyI6ImFnYXRlOjU3MDUyMjA2ZTRi
MGRlNDNlYzE5NzM2YiIsImdpdmVuX25hbWUiOiJKb2hubnkgQi4iLCJmYW1pbHlfbmFtZSI6Ik
dvb2QiLCJuYW1lIjoiSm9obm55IEIuIEdvb2QiLCJlbWFpbCI6ImpvaG55Lmdvb2RAZXhhbXBs
ZS5jb20iLCJlbWFpbF92ZXJpZmllZCI6ZmFsc2UsImlhdCI6MTQ1OTk3Mzc1OCwiZXhwIjoxND
Y3NzQ5NzU4LCJhdWQiOiJkcnVwYWwifQ.1IqjodUNGZ8pKnlxmjzR0XcDgs8Hnl-ufeFsSNH3q
aA",
  "expires_in": 28799,
  "token_type": "bearer"
}
```

The `id_token` represents the following structure (when using `scope=openid email profile`):

```
{
    "alg": "HS256"
}
.
{
    "sub": "user1",
    "iss": "agate:57052206e4b0de43ec19736b",
    "given_name": "Johnny B.",
    "family_name": "Good",
    "name": "Johnny B. Good",
    "email": "johny.good@example.com",
    "email_verified": false,
    "iat": 1459973758,
    "exp": 1467749758,
    "aud": "drupal"
}
.
[signature]
```

Step 3. ID Resource Access

In addition to the `id_token` included in the access token response, the user information can be retrieved from the UserInfo end point. This step is similar to the resource access one (the resource is then the user information).

An example of ID resource request is:

```
curl -X GET --header "Authorization: Bearer ACCESS_TOKEN"
https://agate.example.org/ws/oauth2/userinfo
```

The response is in JSON format and contains the user profile claims. An example of a response is:

```
{
  "family_name": "Good",
  "sub": "user1",
  "iss": "agate:57052206e4b0de43ec19736b",
  "email_verified": false,
  "given_name": "Johnny B.",
  "email": "johny.good@example.com",
  "name": "Johnny B. Good"
}
```