Agate Documentation

OBiBa

Sep 25, 2018

Contents

Targeted at individual studies and study consortia, OBiBa software stack (Opal, Mica 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 preferred 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.

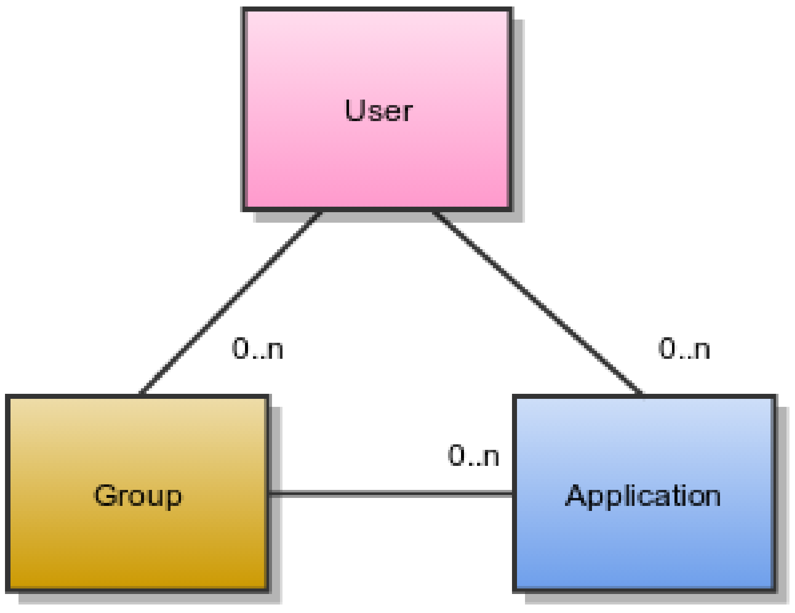
Warning: Agate documentation is in the process of being rewritten. See also the `Agate Documentation Archive`

CHAPTER 1

Introduction

1.1 Users, Groups and Applications

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.



1.1.1 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.

1.1.2 Group

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

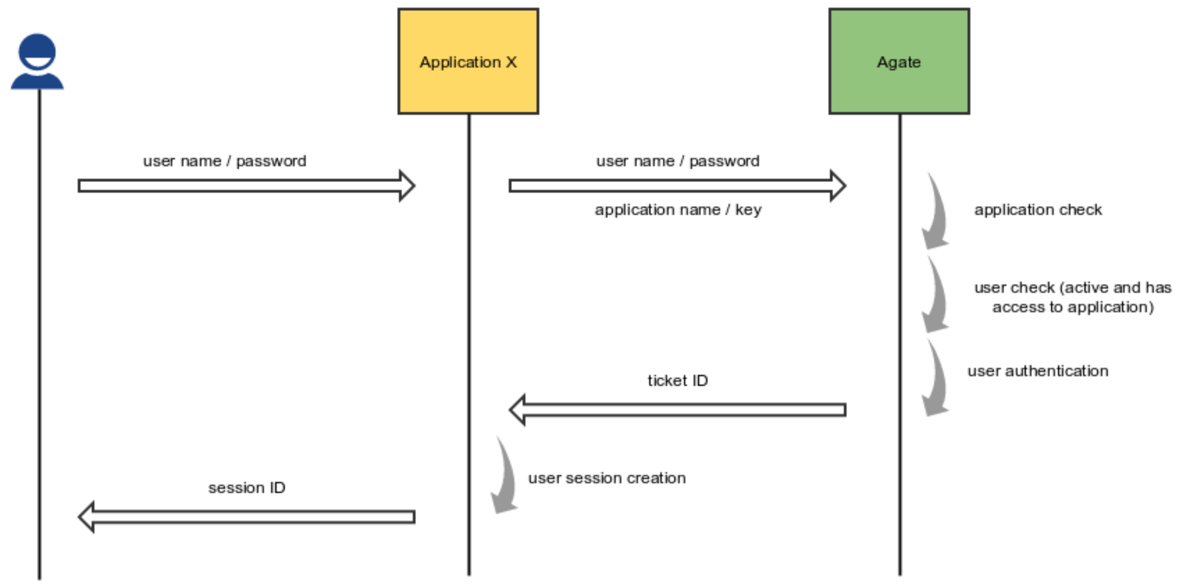
Members of a group can have access to the applications associated to it.

1.1.3 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.

1.2 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.



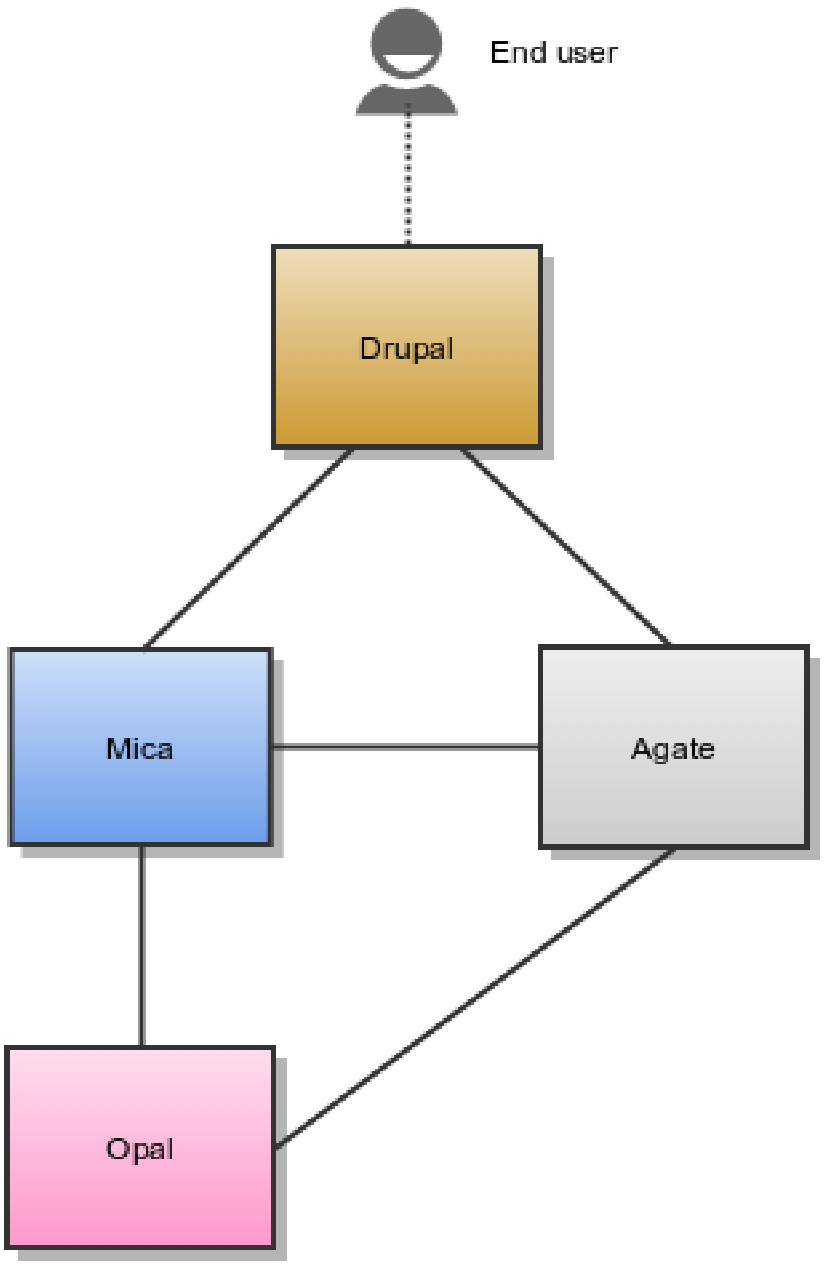
1.3 Architecture, Servers and Clients

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:



1.3.1 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.

1.3.2 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.

1.3.3 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.

1.3.4 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.

CHAPTER 2

Installation

Agate is a stand-alone Java server application that requires MongoDB as database engine.

2.1 Requirements

2.1.1 Server Hardware Requirements

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

2.1.2 Server Software Requirements

Software	Suggested version	Download link	Usage
Java	>= 1.8.x	Java Oracle downloads	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.

2.2 Install

Agate is distributed as a Debian/RPM package and as a zip file. 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 *Configuration* instructions.

2.2.1 Debian Package Installation

Agate is available as a Debian package from OBiBa Debian repository. To proceed installation, do as follows:

- Install Debian package. Follow the instructions in the repository main page for installing Agate.
- Manage Agate Service: after package installation, Agate server is running: see how to manage the Service.

2.2.2 RPM Package Installation

Agate is available as a RPM package from OBiBa RPM repository. To proceed installation, do as follows:

- Install RPM package. Follow the instructions in the RPM repository main page for installing Agate.
- Manage Agate Service: after package installation, Agate is running: see how to manage the Service.

2.2.3 Zip Distribution Installation

Agate is also available as a Zip file. To install Agate zip distribution, proceed as follows:

- Download Agate distribution
- 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.
- Create an `AGATE_HOME` environment variable
- 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
```

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.

2.3 Upgrade

The upgrade procedures are handled by the application itself.

2.3.1 Debian Package Upgrade

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

```
apt-get install agate
```

2.3.2 RPM Package Upgrade

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

```
yum install agate
```

2.3.3 Zip Distribution Upgrade

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

2.4 Execution

2.4.1 Server launch

Service

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

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**.

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

```
service agate help
```

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.

Environment variable	Required	Description
`AGATE_HOME`	yes	Path to the Agate "home" 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`

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.

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.

2.4.2 Usage

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

- http://localhost:8081 will provide a connection without encryption,
- https://localhost:8444 will provide a connection secured with ssl.

2.4.3 Troubleshooting

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

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

CHAPTER 3

Configuration

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.

3.1 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.

3.2 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
`spring.data.mongodb.uri`	MongoDB URI. Read Standard Connection String Format to learn more.

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

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

MongoDB User Creation Example

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

```
use admin

db.createUser( {
  user: "agateadmin", pwd: "agateadmin",
  roles: [
       { "role" : "readWrite", "db" : "agate" },
       { "role" : "dbAdmin", "db" : "agate" },
       { "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
```

Note: 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.

3.3 User Directories

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**. See also Shiro ini file documentation.

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

By default Agate server has several built-in user directories (in the world of Shiro, a user directory is called a realm):

- 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. It is also not recommended to use this file-based user directory for adding users. It is mainly dedicated to define a default system super-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.

```
# =======================
# 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/
↪OEk0jsZbYXjiGhR7/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):

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

3.4 Notification Emails

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.

3.5 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,
- specifies acceptable protocols and cipher suites,
- 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>
```

For performance, you can also activate Apache's compression module (mod_deflate) with the following settings (note the json content type setting) in file */etc/apache2/mods-available/deflate.conf*:

```
<IfModule mod_deflate.c>
  <IfModule mod_filter.c>
      # these are known to be safe with MSIE 6
      AddOutputFilterByType DEFLATE text/html text/plain text/xml
      # everything else may cause problems with MSIE 6
      AddOutputFilterByType DEFLATE text/css
      AddOutputFilterByType DEFLATE application/x-javascript application/javascript␣
↪application/ecmascript
      AddOutputFilterByType DEFLATE application/rss+xml
```

(continues on next page)

(continued from previous page)

```
      AddOutputFilterByType DEFLATE application/xml
      AddOutputFilterByType DEFLATE application/json
  </IfModule>
</IfModule>
```

CHAPTER 4

Web 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 *Users, Groups and Applications* 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
- *Administration*: configure server settings

4.1 Requirements

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

CHAPTER 5

Users Management

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* domain documentation.

5.1 Permissions

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

5.2 Operations

5.2.1 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.

5.2.2 Edit a user

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

5.2.3 Delete a user

A user can be deleted.

5.2.4 Reset a user's password

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

5.2.5 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.

CHAPTER 6

Groups Management

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* domain documentation.

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

6.1 Permissions

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

6.2 Operations

6.2.1 Add group

Creates a group defined by a unique name.

6.2.2 Edit group

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

6.2.3 Delete group

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

CHAPTER 7

Applications Management

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* domain documentation.

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

7.1 Permissions

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

7.2 Operations

7.2.1 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.

7.2.2 Edit an application

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

7.2.3 Delete an application

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

CHAPTER 8

Tickets Management

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.

8.1 Permissions

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

8.2 Operations

8.2.1 Delete ticket

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

CHAPTER 9

Administration

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

9.1 Properties

The following general configuration properties can be modified:

Property	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.

9.2 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.

Note: 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-----`.

9.2.1 Create a (self-signed) certificate

Useful when in testing phase, not recommended in production.

1. Click on the *Add Keys* drop-down.
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.

9.2.2 Import a certificate

It is recommended to use a valid key pair in production.

1. Click on the *Add Keys* drop-down
2. Select *Import*. Here you may use (1) certificate and (2) private key that you created using third party software e.g., OpenSSL. Note that both the certificate and the private key must be in PEM format.
3. Save.
4. Finally, in order for the changes to be taken in account you need to restart Agate server.

9.3 User Attributes

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

CHAPTER 10

Python Introduction

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

10.1 Requirements

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

10.2 Installation

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

- use the Debian/RPM package manager
- use a Python package

10.2.1 Debian Package Installation

Follow the OBiBa Debian Repository instructions and run:

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

10.2.2 RPM Package Installation

Follow the OBiBa RPM Repository instructions and run:

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

10.2.3 Python Package Installation

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

Install on Linux or Mac

1. Download the most recent version
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
```

Note: 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 Google 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 most recent version 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)

10.3 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.

CHAPTER 11

User Commands

User management commands.

11.1 Add User

Add a new user.

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

11.1.1 Credentials

Authentication is done by username/password credentials.

Option	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.

11.1.2 Options

Option	Description
`--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".

11.1.3 Extras

Option	Description
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

11.1.4 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
```

11.2 Delete User

Delete a user.

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

11.2.1 Credentials

Authentication is done by username/password credentials.

Option	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.

11.2.2 Options

Option	Description
`--name NAME`	The user name, mutually exclusive with email.
`--email EMAIL`	The user email, mutually exclusive with name.

11.2.3 Extras

Option	Description
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

11.2.4 Example

Delete a user by its name.

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

CHAPTER 12

Group Commands

Group management commands.

12.1 Add Group

Add a new group.

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

12.1.1 Credentials

Authentication is done by username/password credentials.

Option	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.

12.1.2 Options

Option	Description
`--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.

12.1.3 Extras

Option	Description
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

12.1.4 Example

Add a new group.

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

12.2 Delete Group

Delete a group.

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

12.2.1 Credentials

Authentication is done by username/password credentials.

Option	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.

12.2.2 Options

Option	Description
`--name NAME`	The group name, required.

12.2.3 Extras

Option	Description
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

12.2.4 Example

Delete a group.

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

CHAPTER 13

Application Commands

Application management commands.

13.1 Add Application

Add a new application.

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

13.1.1 Credentials

Authentication is done by username/password credentials.

Option	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.

13.1.2 Options

Option	Description
`--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.

13.1.3 Extras

Option	Description
-h, --help	Show the command help's message
--verbose, -v	Verbose output

13.1.4 Example

Add a new application.

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

13.2 Delete Application

Delete an application.

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

13.2.1 Credentials

Authentication is done by username/password credentials.

Option	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.

13.2.2 Options

Option	Description
--name NAME	The application name, required.

13.2.3 Extras

Option	Description
-h, --help	Show the command help's message
--verbose, -v	Verbose output

13.2.4 Example

Delete an application.

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

CHAPTER 14

Other Commands

Other commands for advanced users.

14.1 Web Services

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

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

14.1.1 Arguments

Argument	Description
`ws`	Web service path, for instance: /user/xxx

14.1.2 Credentials

Authentication is done by username/password credentials.

Option	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.

14.1.3 Options

Option	Description
`--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.

14.1.4 Extras

Option	Description
`-h, --help`	Show the command help's message
`--verbose, -v`	Verbose output

14.1.5 Example

Get all users.

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

CHAPTER 15

OAuth2 Introduction

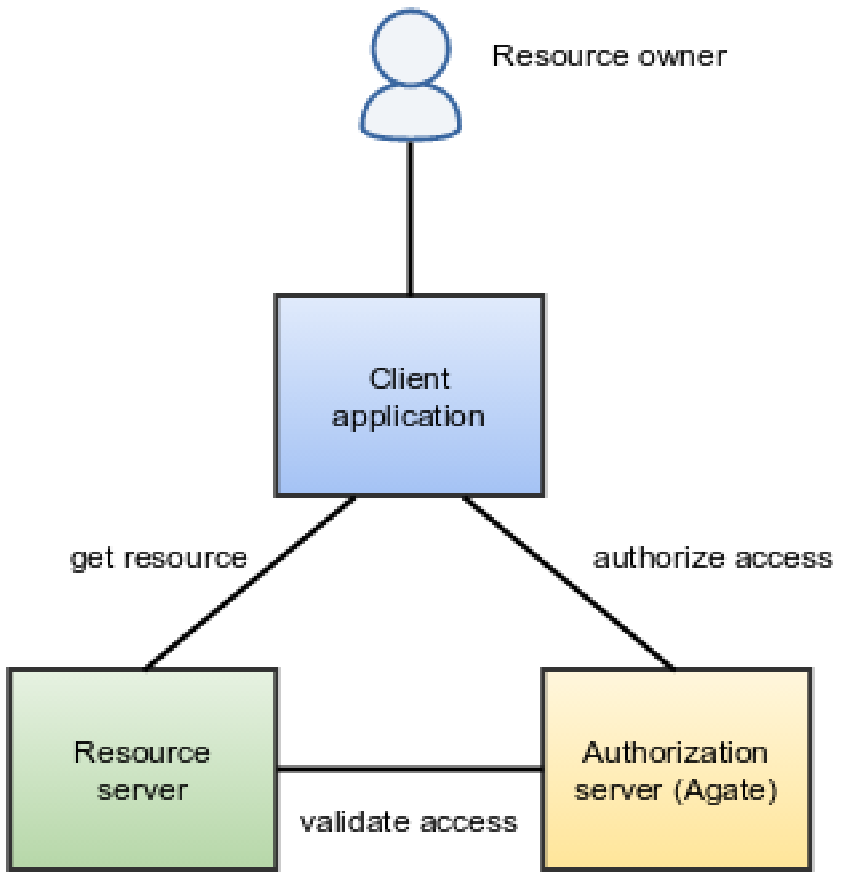
Agate exposes web services that implements the OAuth2 protocol. OAuth2 is an open authorization protocol which enables applications to access each others data. The authorization refers to the fact that these data are accessed on behalf of a resource owner.

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

15.1 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.



15.2 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.

15.3 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.

15.4 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).

CHAPTER 16

Authorization Code Grant Flow

16.1 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.

16.2 Step 1. Authorization

16.2.1 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.

16.2.2 Response

If the resource owner accepts to grant the 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&expires_
↪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.

16.2.3 Errors

The following response errors can be encountered during this step.

```
GET https://client.example.org/redirect?error=ERROR_CODE&error_description=ERROR_
↪MESSAGE
```

Parameter	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.

16.3 Step 2. Access Token Issuing

16.3.1 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:

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).

16.3.2 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.
↪eyJzdWIiOiJlZGl0b3IiLCJpc3MiOiJhZ2F0ZTo1NmZjMzg0MmNjZjJjMWM3ZWM1YzVkMTQiLCJpYXQiOjE0NTk0NTg0NTgsImV
↪PqlLSZegdPLM2byp0jsgWV-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.

16.3.3 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"
}
```

16.4 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
```

16.5 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.

CHAPTER 17

Resource Owner Password Credentials Grant Flow

17.1 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.

17.2 Access Token Issuing

17.2.1 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.

17.2.2 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.
↪eyJzdWIiOiJlZGl0b3IiLCJpc3MiOiJhZ2F0ZTo1NmZjMzg0MmNjZjJjMWM3ZWM1YzVkMTQiLCJpYXQiOjE0NTk0NTg0NTgsIm
↪PqlLSZegdPLM2byp0jsgWV-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.

17.2.3 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"
}
```

CHAPTER 18

OpenID Connect Flow

18.1 Summary

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

18.2 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
```

18.3 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.
↪eyJzdWIiOiJ1c2VyMSIsImlzcyI6ImFnYXRlOjU3MDUyMjA2ZTRiMGRlNDNlYzE5NzM2YiIsImlhdCI6MTQ2MDA0MTU4NCwiZXh
↪7SblBktnvXaoBFL61Rx_jb6PXXYPr4TFMlyi4ZYP5xE",
  "id_token": "eyJhbGciOiJIUzI1NiJ9.
↪eyJzdWIiOiJ1c2VyMSIsImlzcyI6ImFnYXRlOjU3MDUyMjA2ZTRiMGRlNDNlYzE5NzM2YiIsImdpdmVuX25hbWUiOiJKb2hubn
↪1IqjodUNGZ8pKnlxmjzR0XcDgs8Hnl-ufeFsSNH3qaA",
  "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]
```

18.4 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.",
```

(continues on next page)

(continued from previous page)

```
  "email": "johny.good@example.com",
  "name": "Johnny B. Good"
}
```