OpenAGE Viewer 0.10.0 Manual

OpenAGE Viewer 0.10.0 Manual

Manual for version 0.10.0.SNAPSHOT (453bf801351f03c24997a967e4aea63257971a50)

Table of Contents

Part I. Getting started

Chapter 1. Understanding the Viewer application

Goals of the application

Create front-office applications for OpenAGE

OpenAGE enables an agile approach for building business applications. This approach implies a "back-office" approach with standardized navigation and a standardized look-and-feel built using a standard set of user interface widgets arranged around a master-detail style user interface.

Despite being standardized, forms and lists in the OpenAGE back-office are often flexible enough for many applications, and are augmented by extension points for implementing business rules and workflow, as well as using built-in security profiles for controlling access and visibility of information, and flexible searching tools (free-text search and structured advanced search).

OpenAGE Viewer is primarily designed to add flexibility to the way in which information is made available, modified, and to the way in which users can navigate, search, and display information. Whereas the back-office is designed for productivity in terms of structuring data, the front-office is designed for productivity in customizing presentation and navigation (whilst leveraging all the advantages of security profiles and business rules handled by the back-office).

The front-office delegates data access, security profiles, and business rules to the back-office. It provides several levels of customization:

- CSS stylesheet modifications for basic visual theming.
- HTML template modifications for customizing the way in which information is presented and the way in which the user can navigate.
- OSGi[1] plug-ins: requiring some Java development skills, these plug-ins can be added, modified, and removed without stopping the OpenAGE Viewer application, enabling more advanced customization (enhancing business rules, modifying page processing, integrating other components of your information system, for example).

Create applications with the tools you prefer

OpenAGE Viewer is aimed at front-end developers and graphic designers. As such, it is designed to make it easy to get instant feedback on design changes, as well as to deploy them.

Shared media files (such as pictures, video, and audio) can be dropped in and made instantly available. Visual themes can be easily organized into packages of templates, each with their own collection of packages of stylesheets. These templates and stylesheets are easy to edit directly with existing visual design tools and text editors, whatever you prefer. When you are satisfied with the result, it is easy to package up and deploy these design assets on a production server.

When you need to go even further and add customized behavior (security profiles, business logic, or navigation), this can be done using robust industry-standard OSGi Java-based plug-in technology. Java code using our documented API can be compiled and deployed without having to restart the OpenAGE Viewer server, and can even be modified and replaced as many times as you need, without needing to restart the server.

[1] http://www.osgi.org/

Chapter 2. Installing the Viewer application

Requirements

OpenAGE Viewer runs on modern 64-bit operating systems with a compatible Java® SE 8 runtime environment. Supported server operating systems for production use include Microsoft® Windows® Server 2008 and various Linux distributions, such as Ubuntu LTS releases.

OpenAGE Viewer *is not* a Java® EE web application. For maximum agility and productivity, it embeds its own Java-based HTTP application server, which can be deployed (if required) in production in tandem with a reverse proxy such as Apache HTTPD or NGINX. This design decision makes it possible to offer a fully-dynamic front-office application environment; such productivity is difficult to achieve using the classic deployment approach using enterprise application servers (the OpenAGE back-office is designed for such servers).

Directory layout

Application directory

OpenAGE Viewer is normally supplied as a compressed ZIP archive, "`ViewShell-0.10.0.SNAPSHOT.zip`". When decompressed, this creates a directory called "`viewshell`", containing:

1. "`viewshell.jar`", the executable Java archive containing the application server.
2. "`lib`", containing the third-party libraries required to start the application server.

Please note that unlike OSGi bundles, these "`.jar`" files *cannot* be replaced without stopping the Viewer application.

When you start OpenAGE Viewer, it needs to be able to create several files in a specific place (all within the same directory, referred to from here on using Unix-style environment variable syntax as `$VSH_HOME`). Refer to the section called “Specifying VSH_HOME” for details

Viewer data directory

Specifying VSH_HOME

OpenAGE Viewer uses a specific directory to store application data, including stylesheets, templates, runnable Java code, log files, and configuration. The directory used is referred to as `$VSH_HOME`; however the path of this directory is not fixed, and is selected according to the following rules:

1. By using the value of the Java system property `VSH_HOME`, if defined, or
2. By using the value of an environment variable `VSH_HOME`, if defined, or
3. By creating a "`VSH_HOME`" subdirectory in the working directory used to start "`viewshell.jar`", if the user starting the application has sufficient permissions, or
4. By creating a "`VSH_HOME`" subdirectory in the home directory of the user starting the application, if the user starting the application has sufficient permissions.

If none of these values can be used, the application will quit with an operating system error return code.

If the application starts successfully, normally several directories, files (including log files and default configuration) will be created as necessary in the specified location. Care should be taken (depending upon the option selected) that the appropriate user is used, and that environment variables are defined as expected.

The "assets" directory

The `assets` directory contains two subdirectories: "`media`" and "`themes`".

The `media` directory contains files that are served from the application server using the "media" path (by default, the server runs on "`http://hostname:8086/front/`", so this would map to "`http://hostname:8086/front/media/*`"). Files in the `media` directory are therefore always accessible (unless specific security filters are installed by a plug-in component) for all visual themes.

The `themes` directory contains *theme packs*, which can either be *imported* "`.zip`" theme packages, or *expanded* directories. The structure of a theme pack depends on the type of theme pack; by default, an implementation based on FreeMarker[1] is included, but other technologies can be used.

Caution

When exporting a theme pack from a development environment to another environment (such as a production server), you should use the export facilities provided by the OpenAGE Viewer application API; this preserves the unique identifiers associated with files, used when associating data and navigation with visual themes.

The "system" directory

The `system` directory contains the OSGi bundles required by the OpenAGE Viewer application server and the third-party libraries that these components require. These bundles are automatically deployed by `viewshell.jar` when it starts.

The system bundles are versioned, and can be deleted and replaced dynamically at runtime, for example to perform live upgrades or apply hotfixes without stopping the server.

Caution

By default, when a system bundle is removed, the application server automatically uninstalls the associated code objects *and* stops all other components (including system bundles and plugin bundles) that have declared dependencies. For example, removing the HTTP system bundle (or any of its dependencies) makes the front-office unavailable until it is reactivated (this requires that all of its dependencies are satisfied).

The "plugins" directory

The `plugins` directory can contain extra OSGi bundles, if required by your application. Deploying a new bundle is as simple as copying the bundle "`.jar`" file into this directory.

If your bundle needs to perform specific actions when it is started or stopped, you should add a class implementing the OSGi `BundleActivator` interface (and indicate this in the bundle's internal meta-data "`MANIFEST.MF`" file).

The "conf" directory

The `conf` directory contains configuration files. The main configuration files are:

[1] http://freemarker.apache.org/

- `config-system.xml`: configures the HTTP server and other main attributes. Changes to this file are immediately applied, including server path, hostname, and port.
- `config-logging.xml`: configures logging. Most changes in this file are immediately applied (however, a server restart may be required in some cases).
- `config-mimetypes.xml`: determines how filename extensions are associated with IETF MIME types (HTTP content types).

Additional configuration files will be created in this directory. These configuration files generally reflect configuration settings applied via the application API or created on demand by plug-in components.

The "cache" directory

The `cache` directory contains temporary files as required by the OSGi Framework used by the OpenAGE Viewer application server. These files are not intended for external tampering; all normal operations can be performed using the API and the other directories within `$VSH_HOME`.

The "logs" directory

The `logs` directory will by default contain 5 log files:

- `system.log`: contains information from the OpenAGE Viewer application shell and system components.
- `http.log`: contains information from the HTTP component to enable tracing of HTTP request and response processing. The mapped diagnostic context (a unique request identifier that tracks all log entries related to a unique request) is applied to every HTTP call to facilitate tracking of all user requests on the server side.
- `user.log`: exists to contain log entries from plug-in components, should this be necessary, to easily distinguish such tracing information from other log entries.
- `osgi.log`: contains log entries emitted using the OSGi `LogService`.
- `library.log`: contains log entries emitted by third-party libraries using system bundles or plug-in bundles. When additional third-party libraries are added (for plug-in components), output from these libraries can be added using the `LIBRARY_NAMESPACES` property in the "`config-logging.xml`" configuration file.

Other options exist for managing log files, such as the quantity (level) of information (warning, information, debugging) in log files, and managing size, rollover, and backups.

When running as a Windows® service, additional log files will be generated. Refer to "the section called “Using Microsoft® Windows® systems”".

Running Viewer

Starting from the command line

Assuming that you have installed and correctly configured a Java SE 8 environment, from within the directory where `viewshell.jar` is installed (along with its associated `lib` subdirectory), you can start OpenAGE Viewer with the following command:

```
java -jar viewshell.jar
```

In the event of receiving `UnsupportedClassVersionError`, check your configuration: this normally indicates that the **java** command was mapped to an earlier version of Java SE.

When this command is successfully issued, the command line is blocked until the application is stopped. The application can be halted if necessary using **CTRL+C** in the command line, or by using the terminate command.

Stopping the server using a browser

The terminate command can be issued without authentication over HTTP after changing the default settings (refer to Options for `server.signal.uri-enabled`). Depending upon the server endpoint configuration (here we assume that the endpoint is "`http://localhost:8086/front/`"), it could be issued as follows:

```
http://localhost:8086/front/admin/signal/TERM
```

When the command is successful, the server responds in plain text with `TERM` and then shuts down.

Caution

For production servers, we recommend disabling this feature, as it exposes the application server to *denial of service* attacks. This feature is not enabled by default.

Stopping the server using a JMX console

OpenAGE Viewer exposes a management interface using JMX by default, and as such can be managed by tools such as `jconsole`. Using your JMX client, connect to the `viewshell.jar` process, then go to *MBeans* view, and navigate to `fr.reflexe.viewshell`. The `HostApplication` MBean provides a `terminate()` operation, which will shutdown OpenAGE Viewer and disconnect the JMX client.

Installing OpenAGE Viewer as a service

Using Microsoft® Windows® systems

On a Microsoft® Windows® system system, the application should be installed in the `%ProgramFiles%` directory, and `VSH_HOME` will be a subdirectory of the `%ProgramData%` directory, where the name of the subdirectory matches the name of the service, which by default is "`ViewShell`".

Therefore, in a default installation, the following paths will be used:

- `C:\Program Files\viewshell` as the program installation directory.
- `C:\ProgramData\ViewShell` as the `VSH_HOME` application data directory, including log files, configuration information, and runtime data.

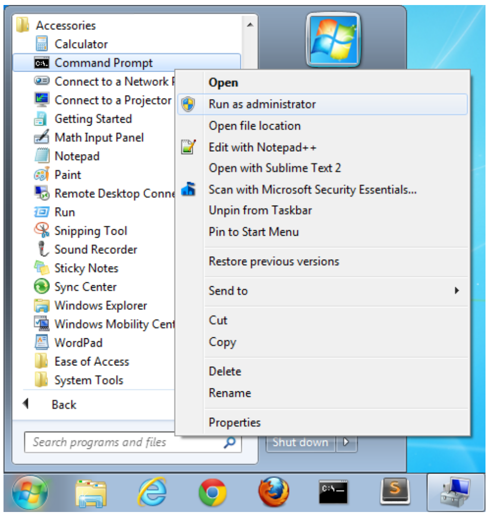
On typical Microsoft® Windows® installations, the environment variable `%ProgramData%` is defined and accessible to all users, including normal user accounts and the "`Local System`" account used to run services. If the environment variable is for some reason undefined, then `VSH_HOME` will default to creating and using a "`home`" directory within the installation directory. You can also override the default service installation script to specify an alternative path (and modify other parameters, if necessary).

Before proceeding to install the service, you must define the `%JAVA_HOME%` environment variable, so that it refers to the installation directory of a 64-bit JDK (this directory will contain the "`bin`", "`lib`", and "`jre`" directories, along with some other files and directories).

To install the service, you must open a command prompt as an administrator, and navigate to the "`viewshell`" installation directory, then to the "`service`" subdirectory. You can then run the

`InstallService.bat` script to install the service (the script will also attempt to start the service immediately).

Figure 2.1. Opening a command prompt as an administrator



For example, you might enter the following commands:

```
cd %ProgramFiles%\viewshell\service
InstallService.bat
```

Running the `InstallService.bat` script will normally produce the following output:

```
Installing Windows Service (for 64-bit Intel/AMD architecture).
Running script with administrative permissions.

Using 64-bit JVM: C:\java\jdk8
Using VSH_HOME:   C:\ProgramData\ViewShell
Using LOGPATH:    C:\ProgramData\ViewShell\logs
Using CLASSPATH:  C:\Program Files\viewshell\viewshell.jar

Installing "ViewShell" service using ViewShell.exe
Starting "ViewShell" service...

Started service: check log files to confirm startup (see %LOGPATH%, above)
```

If the script fails for any reason (for example, insufficient permissions or incorrect paths), then an ERROR message will be displayed in the command window. Note that you should check the logfiles, as suggested, as in some cases the service can install correctly but the application may fail to start correctly (for example, insufficient permissions or a conflict with another application running as a server).

To uninstall the service, you must again open the command prompt as an administrator, and you might enter the following commands:

```
cd %ProgramFiles%\viewshell\service
UninstallService.bat
```

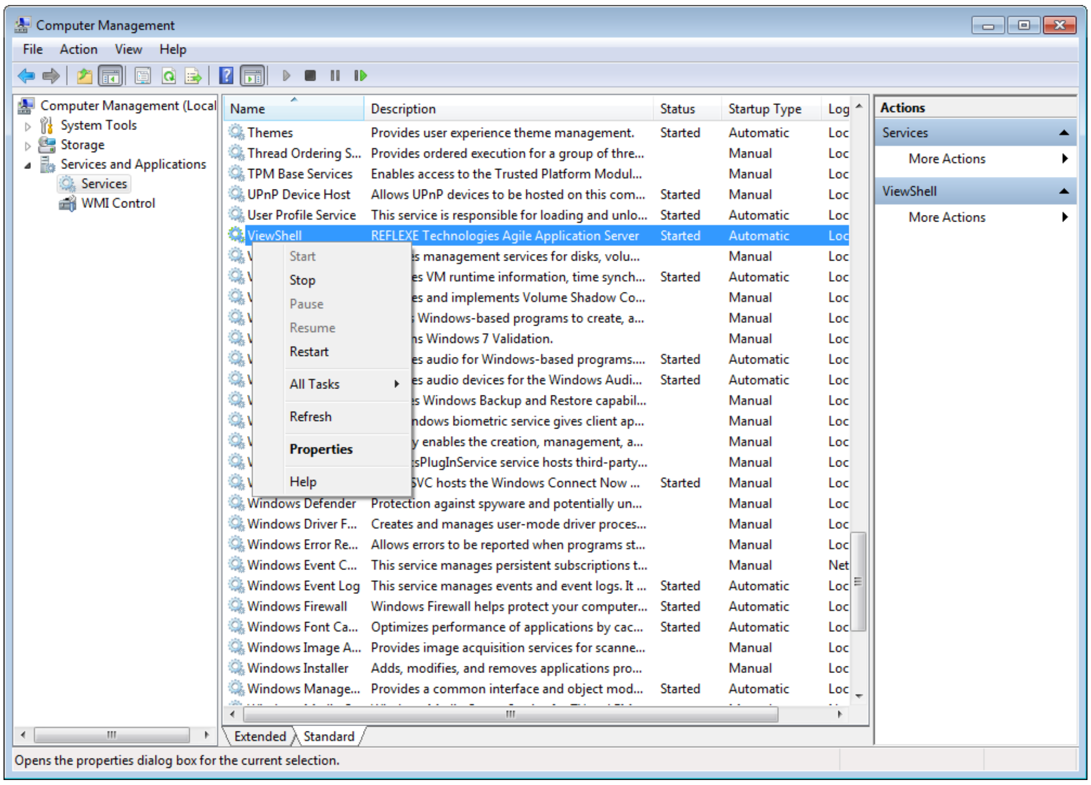
Running the `UninstallService.bat` script will normally produce the following output:

```
Uninstalling Windows Service (for 64-bit Intel/AMD architecture).
Running script with administrative permissions.

Uninstalled "ViewShell" service.
```

When the application runs as a service, you can start it, stop it, and restart it using the "Computer Management" window (right click on the "Computer" icon on the Microsoft® Windows® desktop, and select the "Manage" option).

Figure 2.2. Computer Management window showing the running service

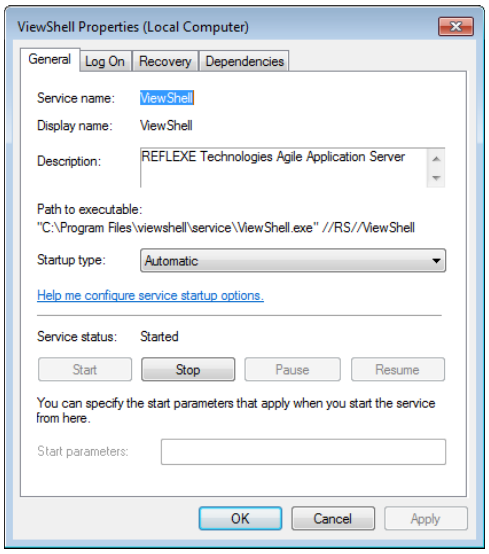


Name	Description	Status	Startup Type	Log
Themes	Provides user experience theme management.	Started	Automatic	Loc
Thread Ordering S...	Provides ordered execution for a group of thre...		Manual	Loc
TPM Base Services	Enables access to the Trusted Platform Modul...		Manual	Loc
UPnP Device Host	Allows UPnP devices to be hosted on this com...	Started	Manual	Loc
User Profile Service	This service is responsible for loading and unlo...	Started	Automatic	Loc
ViewShell	REFLEXE Technologies Agile Application Server	Started	Automatic	Loc
\	s management services for disks, volu...		Manual	Loc
\	es VM runtime information, time synch...	Started	Automatic	Loc
\	es and implements Volume Shadow Co...		Manual	Loc
\	Windows-based programs to create, a...		Manual	Loc
\	ns Windows 7 Validation.		Manual	Loc
\	es audio for Windows-based programs....	Started	Automatic	Loc
\	es audio devices for the Windows Audi...	Started	Automatic	Loc
\	s Windows Backup and Restore capabil...		Manual	Loc
\	ndows biometric service gives client ap...		Manual	Loc
\	y enables the creation, management, a...		Manual	Loc
\	sPlugInService service hosts third-party...		Manual	Loc
\	SVC hosts the Windows Connect Now ...	Started	Manual	Loc
Windows Defender	Protection against spyware and potentially un...		Manual	Loc
Windows Driver F...	Creates and manages user-mode driver proces...		Manual	Loc
Windows Error Re...	Allows errors to be reported when programs st...		Manual	Loc
Windows Event C...	This service manages persistent subscriptions t...		Manual	Net
Windows Event Log	This service manages events and event logs. It ...	Started	Automatic	Loc
Windows Firewall	Windows Firewall helps protect your computer...	Started	Automatic	Loc
Windows Font Ca...	Optimizes performance of applications by cac...	Started	Automatic	Loc
Windows Image A...	Provides image acquisition services for scanne...		Manual	Loc
Windows Installer	Adds, modifies, and removes applications pro...		Manual	Loc
Windows Manage...	Provides a common interface and object mod...	Started	Automatic	Loc

Opens the properties dialog box for the current selection.

The start, stop, and restart actions are available from the above window's toolbar. You can also access a more detailed dialog box by right-clicking the service and selecting the "Properties" option. The following dialog window is displayed:

Figure 2.3. General settings for the Microsoft® Windows® service


ViewShell Properties (Local Computer)
General
Log On
Recovery
Dependencies
Help me configure service startup options.
Start
Stop
Pause
Resume
OK
Cancel
Apply

Service name: ViewShell

Display name: ViewShell

Description: REFLEXE Technologies Agile Application Server

Path to executable:
"C:\Program Files\viewshell\service\ViewShell.exe" //RS//ViewShell

Startup type: Automatic

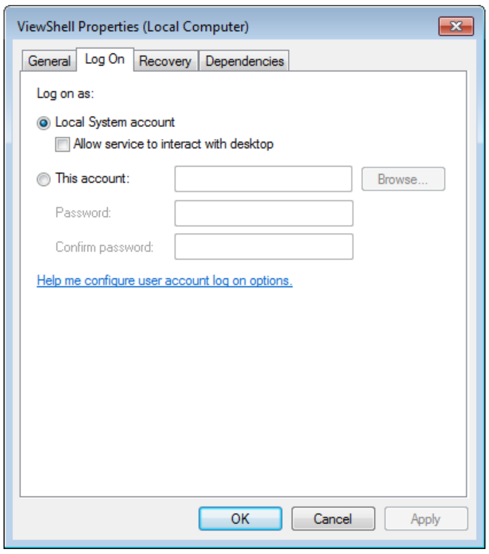
Service status: Started

You can specify the start parameters that apply when you start the service from here.

Start parameters:

The second tab in the window lets you specify an alternative user account to use for running the service, if you have specific security requirements.

Figure 2.4. User account settings for the Microsoft® Windows® service


ViewShell Properties (Local Computer)
General
Log On
Recovery
Dependencies
Browse...
Help me configure user account log on options.
OK
Cancel
Apply

Log on as:
Local System account [✓]
Allow service to interact with desktop [ ]
This account:
Password:
Confirm password:

Additional log files will be generated when running as a service.

- `service.YYYY-MM-DD.log`: contains debugging information generated when starting and stopping the service.
- `stdout.log`: contains any output to `STDOUT`.
- `stderr.log`: contains any output to `STDERR`.

Using Linux systems and JSVC

Running the application as the `root` user on Unix-type operating systems is not supported. The server administrator will however require `root` privileges to be able to install and control the server application process.

To install OpenAGE Viewer on a Linux server, starting by extracting `ViewShell-0.10.0.SNAPSHOT.zip` into an appropriate directory (such as `/var/opt`). After that, you need to install the Apache `jsvc` to start, stop and monitor the Viewer server process. You can install it using standard server tools such as `apt-get` (on Debian variants) or `yum` (on RedHat variants), or if necessary by downloading a source distribution from the Apache Commons[2] website, and compiling it. The minimum supported version of `jsvc` is 1.0.15.

Once OpenAGE Viewer and `jsvc` are successfully installed on your machine, you can create new Viewer-based server applications by renaming `viewshell/service/linux/vshservice` as the name of your new service, and then modify it according to the instructions contained in this file. Each independent Viewer-based server application will require a dedicated configuration file. Once you have deployed this new configuration file in `/etc/init.d`, you can start, stop and restart your OpenAGE Viewer service like many other services by using:

```
sudo /etc/init.d/vshservice start|stop|restart
```

Remember to configure your service for automatic startup. This can be done with the **update-rc.d** command, as follows:

```
sudo update-rc.d vshservice defaults
```

You can then select your service from the list of available services.

[2] http://commons.apache.org/proper/commons-daemon/jsvc.html

Installation using a reverse proxy

TODO: describe quick howto for Apache HTTPD.

TODO: necessary if using Linux/Unix and requiring port such as 80.

Chapter 3. Configuration

Configuring logging

OpenAGE Viewer writes information to log files in `$VSH_HOME/logs` as a way of monitoring correct operation and diagnosing any problems that may occur. The amount of information written to these files depends upon configuration settings which are read when the application starts.

These configuration settings are defined in the file `$VSH_HOME/conf/config-logging.xml`, which will by default look similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>ViewShell logging configuration</comment>
<entry key="LOG_ROLL_TYPE">DATE</entry>
<entry key="LOG_LEVEL">INFO</entry>
<entry key="LIBRARY_NAMESPACES">com.teamdev,com.jniwrapper,freemarker</entry>
<entry key="LOG_APPEND">true</entry>
<entry key="LOG_ROLL_COUNT">7</entry>
<entry key="LOG_ROLL_SIZE_MB">1</entry>
</properties>
```

The default settings therefore indicate that each of the standard logfiles (see the section called “The "logs" directory”) will be archived once per day, with up to 7 backup copies (the size of log files is ignored when the "date" rollover policy is used). If the application is restarted, then new log entries are appended to any existing log entries. The level of information is "informative" (therefore excluding debugging information for plug-in developers). Finally, logging output from the specified third-party libraries is configured here to be recorded in the `library.log` file.

Options for `LOG_ROLL_TYPE`

`NONE`	Indicates that log files continue to grow indefinitely.
`DATE`	Indicates that a new log file will be started once per day. Old log files are backed up within the limit of `LOG_ROLL_COUNT`, are renamed to include the date, and are compressed in GZIP format.
`SIZE`	Indicates that a new log file will be started when the size limit (in megabytes, as a positive integer) defined by `LOG_ROLL_SIZE_MB` is reached. Old log files are backed up within the limit of `LOG_ROLL_COUNT`, are renamed to include the date, and are compressed in GZIP format.

Options for `LOG_LEVEL`

`ERROR`	Only *errors* (failure to perform normal operations) are recorded in log files with this setting.
`WARN`	Both *warnings* and *errors* are recorded in log files with this setting. A warning indicates that an abnormal condition was detected (such as an incorrect configuration setting), or that some other unexpected situation arose during processing. Unlike an error, processing is generally able to continue in such cases (although perhaps not as expected, possibly indicating that a more serious error condition will occur later).
`INFO`	This is the default setting, and also includes *error* and *warning* messages. Informational messages will for example echo effective configurations settings (such as version information, lifecycle status of various components, detection of normal changes such as the addition of new content), and are therefore useful to verify that the application is behaving as expected.

`DEBUG`	This level should not be used in production unless a specific issue is encountered that requires additional problem analysis, as it generates much more additional information and therefore requires more disk space (the exact amount will depend upon your configuration and your usage scenarios). It includes messages from all of the above levels, and adds many more additional messages such as echoing the values of default settings and recording information exchanges between internal components. However, it *can* be very useful when creating applications using OpenAGE Viewer, especially when manipulating theme packs, style packs, and OSGi plug-in components.
`TRACE`	This level generates the most information of all, tracing much more detail about how internal components are behaving.

The other settings can be configured as follows:

- `LIBRARY_NAMESPACES`: this is a comma-separated list (containing by default the package namespaces of the default set of embedded libraries) of third-party libraries that can add log messages to the `library.log` file. If you add plug-ins that have dependencies on the SLF4J[1] or Commons Logging[2] APIs, you may add the associated logging namespaces here. This may in turn increase the size of log files, and it should be noted that third-party libraries do not necessarily choose logging levels for messages using the above criteria. Furthermore, note that there is no requirement to add additional logging configuration files traditionally associated with these API.
- `LIBRARY_INFO`: this, if present, is a comma-separated list of third-party libraries that are restricted to logging at `INFO` level (or more severe levels, such as `WARN` or `ERROR`). Some third-party libraries generate a large volume of debugging messages in `DEBUG` mode, which can be counter-productive; this option provides a solution to suppress such behavior.
- `LOG_APPEND`: when `true` (the default), new log messages are always appended to the current log file (if it exists). When `false`, any current log file will be overwritten when the application is restarted (this can be helpful when testing an application in a development environment).
- `LOG_ROLL_COUNT`: this determines how many log file backups should be maintained when `LOG_ROLL_TYPE` is set to either `DATE` or `SIZE`. The value can be either `0` (zero, no backups), a positive integer, or "`UNLIMITED`". With any value other than "UNLIMITED", the application will automatically delete old log files when the number of archived copies for a log file exceeds the limit defined here.
- `LOG_ROLL_SIZE`: this is only used when `LOG_ROLL_TYPE` is set to `SIZE`, and determines the size threshold (as a positive integer, in megabytes) that triggers archiving of each current log file and the creation of new log files.

Configuring the HTTP server

OpenAGE Viewer starts the embedded HTTP server using the settings defined in `$VSH_HOME/conf/config-system.xml` and will attempt to update these settings dynamically if a change to this file is detected while the application is running.

By default, this file will look similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="server.protocol">http</entry>
<entry key="server.hostname">localhost</entry>
<entry key="server.port">8086</entry>
<entry key="server.path">/front</entry>
```

[1] http://www.slf4j.org/
[2] http://commons.apache.org/logging/

```
<entry key="server.session-timeout">30</entry>
<entry key="server.using-session-cookies">false</entry>
<entry key="server.signal.uri-enabled">false</entry>
</properties>
```

The default settings generally start the server using the HTTP protocol using all available local host names, on TCP port 8086, with a base URI of "/front". Additionally, by default, the server does *not* enable HTTP session cookies (for compliance with the *European "e-Privacy" Directive 2009/136/EC*): you can either enable them by replacing the default value, or use URL-encoded session identifiers in web content served by OpenAGE Viewer. For applications that require sessions, it is recommended (where possible) to enable session cookies to improve protection against *session hijacking* and *session fixation* attacks.

Options for `server.protocol`

`http` In version 0.10.0, this is the only valid setting, and indicates that the server will use unencrypted HTTP/1.1 as the transport format. You may use a reverse-proxy configuration if you require HTTPS (SSL/TLS) support.

Options for `server.hostname`

`hostName` In version 0.10.0, although only one specific hostname binding may be specified for starting the server, by default an attempt is made to bind to all available hostnames on all available network interfaces provided by the computer's network configuration, using the address "`0.0.0.0`". This will generally be acceptable for testing on a local network but may need to be changed to enable only appropriate access to the application server from the Internet. If a reverse-proxy is used, it is usually desirable to restrict access to use only the loopback network interface.

At startup, the server will emit `DEBUG` messages to echo information relating to the local host and all discoverable network interfaces.

Note that if your server uses an IDN (International Domain Name), you can directly enter the Unicode form here (as you would enter it in a browser address bar). It will be converted as appropriate, and this can be verified in `http.log`.

Options for `server.port`

`portNumber` The port number must be a valid TCP port number in the range 0-65535 as defined by the IANA authority. A HTTP web server will typically use port 80 (443 for SSL/TLS), although on Unix systems such as Linux, this requires root permissions (therefore, use of a reverse-proxy is recommended in such cases). The default port of 8086 is chosen to avoid conflicts with well-known HTTP services but may already be used on some servers and workstations and should generally be changed to meet your requirements.

Options for `server.path`

`/uri` Specifies the base path to the content provided by OpenAGE Viewer. This can be "/" if the root path (no subdirectory) is preferred, or can indicate the base path, such as "/front" (the default setting). This must never be undefined, and must always start with "/" (it may not contain any additional "/" characters thereafter).

Options for `server.session-timeout`

`timeout` Specifies the number of *seconds* of inactivity before automatic termination of a browser session. Negative values are interpreted as "no timeout". By default, session timeout occurs after 30 minutes.

Options for `server.using-session-cookies`

`true`
: This setting enables session cookies wherever browsers support cookies. With most browsers, when session cookies are enabled, it is not possible to have more than one active session at a time (even when using multiple windows or browser tabs).

`false`
: This is the default setting, selected for compliance with the *European "e-Privacy" Directive 2009/136/EC*. When this setting is active, you must URL-encode *all* links within content served by the OpenAGE Viewer application server. The standard template engine makes this relatively easy, as a "`sid()`" (session ID) template function is provided to simplify such URL-encoding (refer to the section called “sid”). This mode also makes it possible for a user to open multiple simultaneous sessions in the same browser (with browser tabs, for example) each with a different user account.

  Note that this setting only affects *session cookies*. It does not in any way prevent working with other cookies.

Options for `server.signal.uri-enabled`

`true`
: This setting enables a basic shutdown interface using HTTP. When enabled, it can be used as described in the section called “Stopping the server using a browser”.

`false`
: This is the default setting, and recommended for production, to prevent *denial of service* attacks using the `TERM` application signal (note that this is not related to the Unix operating system signal with the same name). Other (more secure) shutdown procedures should be used instead with this setting.

The effective settings selected by the application are written to `http.log` at `INFO` level upon startup, and also whenever these settings are modified when the application is running. If the server fails to start (for example, incorrect hostname or a port already in use), the failure reason is normally recorded in this log file.

Configuring the default connector

OpenAGE Viewer by default does not directly provide any data with which to populate the user interface. Therefore, when installing the application, a key step involves setting up a connector to a data source. The most common data source is an OpenAGE back-office server, and therefore you will need to indicate the *endpoint URL*: this is generally the URL of the server and the OpenAGE context path.

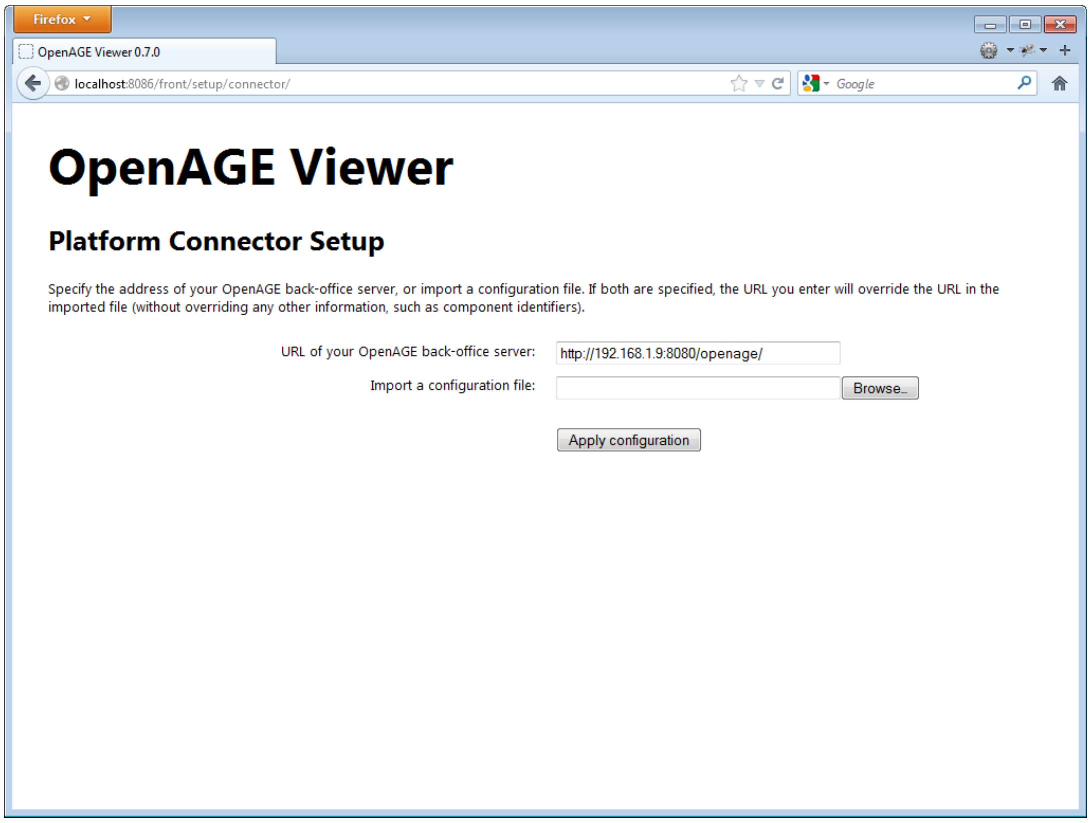
To configure the connector, the most straightforward approach is to use a web browser and navigate to the default address of your OpenAGE Viewer installation:

```
http://localhost:8086/front/
```

At startup, the default address will be written to `http.log` (see the section called “The "logs" directory”). If you have entered an appropriate address, you will be redirected to a setup user interface. The setup user interface is only available when no connector is provided by either configuration or by any installed plugin; it is automatically uninstalled after a connector is installed.

The first step is to either upload a previously-exported connector definition file, or to type in the endpoint URL. If you are installing a production server to run an application prepared on another server, you should import the connector definition, so that you also import the appropriate identifiers that were used to associated the user interface with the data sources. If you enter the endpoint URL without importing a definition file, then a new unique identifier will be generated. You can override the endpoint URL defined in an imported file by specifying it at the same time as you import the file.

Figure 3.1. Specifying or importing the back-office server



OpenAGE Viewer

Platform Connector Setup

Specify the address of your OpenAGE back-office server, or import a configuration file. If both are specified, the URL you enter will override the URL in the imported file (without overriding any other information, such as component identifiers).

URL of your OpenAGE back-office server: http://192.168.1.9:8080/openage/

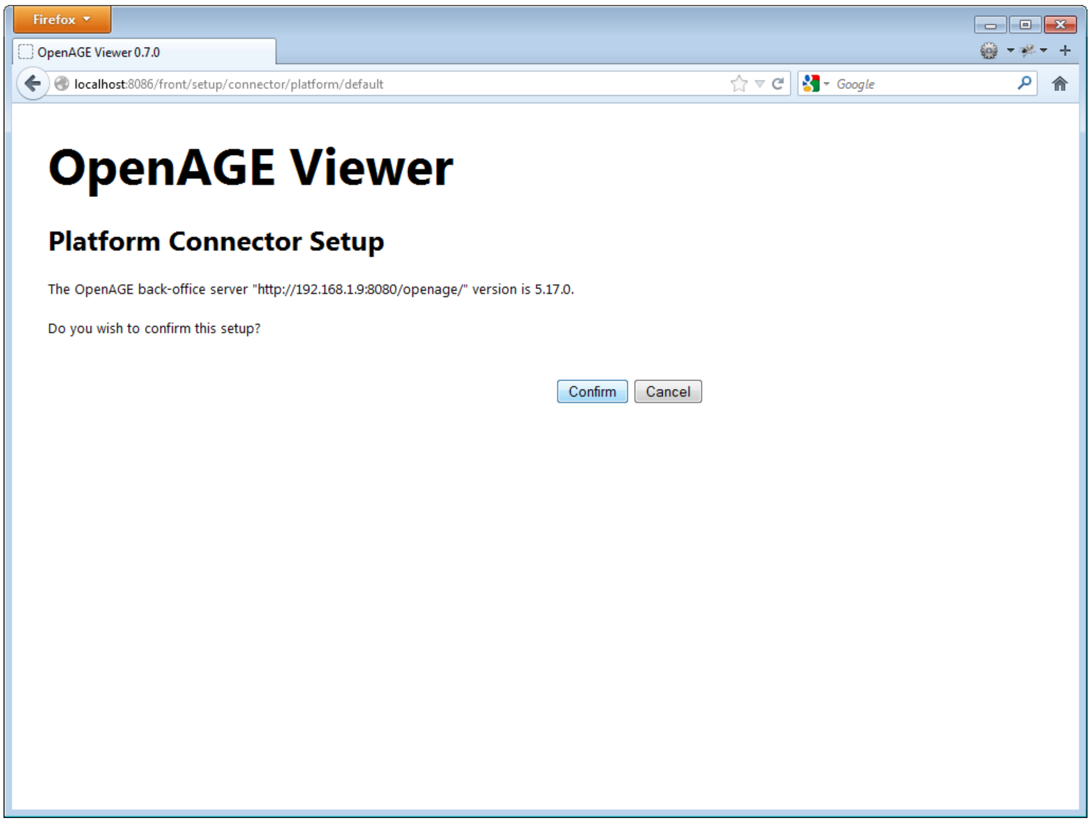
Import a configuration file: Browse...

Apply configuration

Click "Apply configuration" to proceed to the next step.

Before validating the connector definition, OpenAGE Viewer will attempt to contact the server at the specified endpoint URL. If the definition is valid, this is normally as simple as displaying the detected version and asking you to confirm the setup.

Figure 3.2. Confirming and applying the choice of the back-office server



OpenAGE Viewer

Platform Connector Setup

The OpenAGE back-office server "http://192.168.1.9:8080/openage/" version is 5.17.0.

Do you wish to confirm this setup?

However, sometimes this step can lead to additional information and warnings being displayed. In the event of a warning, you can still confirm your setup (if for example you are configuring OpenAGE Viewer prior to configuring the back-office server). Possible situations include:

- Version incompatibility: OpenAGE Viewer knows what the minimum compatible version of the back-office server is, but can only guess what the upper limit is.
- Server unavailability: the back-office server may be currently unavailable. In such cases, check your network configuration (host names, port numbers, firewalls, protocols), and ensure that you have specified the correct OpenAGE context path as part of the URL.
- Server certificate validation failures: if you have installed an SSL/TLS certificate from some certificate authorities, or if you are using a self-signed certificate for testing HTTPS, you may need to install the appropriate root certificate in the `cacerts` file of the Java environment used to run OpenAGE Viewer.

Click "Confirm" to apply the configuration, or "Cancel" to return to the previous step. Once the setup is confirmed, this interface uninstalls itself.

If this interface is not required, it can be deactivated via configuration (see Options for `setup.interface.enable`). Connectors can also be directly defined using configuration files or using a JMX console.

Caution

If OpenAGE Viewer is not installed on the same server as OpenAGE, ensure that both servers have synchronized clocks. The connector will automatically detect clock synchronization problems which may adversely affect the consistency and reliability of re-source processing, and beyond a configurable limit will deactivate various optimizations that depend upon clock synchronization. In short, if clocks are not synchronized, then application performance will be degraded.

Part II. Customizing OpenAGE Viewer

Chapter 4. Customizing the default templates

Template systems and packages

OpenAGE Viewer includes a theming service to make it easier to create user interfaces. The theming service supports multiple template systems, with a default FreeMarker[1]-based template system. All template systems have a well-defined package structure, and are designed to support rapid development and straightforward deployment.

The FreeMarker-based template system is oriented towards creating HTML-based user interfaces. Other template systems can be implemented using other template engines (for example, a template system could be implemented using XML and XLST, or could generate binary files such as spreadsheet documents or PDF documents). Despite this diversity, all template systems structure their packages in the same way. Using OSGi plug-ins, you can provide your own template systems, or even provide additional user interfaces that can be accessed over HTTP and directly access data connectors.

Template packages are structured as follows:

- The template system has access to files in the `assets/themes` directory (see the section called “The "assets" directory” for details). This directory can contain template packages, either as expanded directories (during development) or as imported "`.zip`" files (recommended for production). The theming service attaches template packages to template systems as appropriate.
- Each template package can contain one or more template files, associated media assets (such as images), and one or more style packages. In the case of the default FreeMarker-based template system, the template files are stored in the top-level directory of the template package as "`.ftl`" files, media files are stored in the `media` directory, and style packages are stored in the `style` directory. The `style` directory contains one subdirectory per style package, but does *not* directly store "`.css`" files themselves.
- Each style package can contain one or more stylesheets and associated media assets. In the case of the default FreeMarker-based template system, the "`.css`" files are stored in the top-level directory, and media assets are stored in a `media` subdirectory.

Template packages therefore define how to structure the contents of different pages in the user interface. Style packages make it possible to apply different visual themes and layouts to common templates (for example, varying how content is displayed by audience or device). Simple template packages are not required to contain style packages, and can directly store stylesheets as media assets. Style packages must be defined within the template package to which they apply.

HTTP access to template resources

TODO: describe paths, and IThemeService.THEME_MEDIA_URI.

Creating and modifying style packages

The default collection of templates and styles are not yet fully implemented in version 0.10.0 of OpenAGE Viewer. This feature will be documented later.

TODO

[1] http://freemarker.org/

Chapter 5. Creating custom templates

Default data provided for templates

When using the default page rendering service in OpenAGE Viewer, all templates have a default data added into the template for rendering, including the data for the page and the navigation context. All default data is defined using either standard Java objects, or objects defined in the `fr.reflexe.viewshell` API.

`page`	Provides access to properties of the page design (template, stylesheets, and so on). Implements a subinterface of `IThemedPage`.
`user`	Provides access to properties of the authenticated user (except on pages, such as the sign-on screen, where there is no user). Implements `IUser`.
`asset`	Provides access to a sequence of page assets, such as shared JavaScript libraries or web fonts. The items in this sequence depend upon the page design. Each item implements `IPageAsset`.
`styleSheet`	Provides access to a sequence of stylesheets (that should be compatible with the page template). The items in this sequence depend upon the page design. Each item implements `IStyleSheet`.
`dictionary`	Provides access to a hash of text localization dictionaries, for internationalization support. Hash keys are the dictionary names (which must be unique within the scope of the page). The items in this hash depend upon the page design; additional dictionaries can be accessed using template functions (see the section called “udict”). Each item implements `ITemplateDictionary`.
`contextUri`	A `String` containing the *context URI*, that is the path from the server host and port to the base path of the application. Always starts and ends with "/".
`mediaUri`	A `String` containing the *media URI*, that is the path from the server host and port to the path containing shared media files. Always starts and ends with "/".
`themeMediaUri`	A `String` containing the *theme media URI*, that is the path from the server host and port to the path containing media files that belong to the same template package as the page template. Always starts and ends with "/".

Template syntax with FreeMarker

Default settings for templates

OpenAGE Viewer applies the following default settings when working with FreeMarker templates:

- The character encoding for input and output files is UTF-8.
- The Java locale (regional settings) is `Locale.UK`, used when formatting dates and numbers. This can be overridden using template settings.
- Dates and times are formatted in the UTC time zone.
  - The default date format is `yyyy-MM-dd`, which will output (for example) `2008-10-04`.
  - The default time format is `HH:mm:ssz`, which will output (for example) `18:15:00UTC`.

- Numbers are formatted (unless overridden) using "," as the thousands separator (the digital grouping delimiter), and "." as the decimal mark (to separate the integer part of a whole number from the fractional part). This is a display-oriented default behavior of FreeMarker, so take care to use the "`?c`" built-in when including numbers that requiring further processing (such as including identity numbers in hyperlinks), to prevent unexpected behavior with numbers greater than 999.
- There is no default boolean formatting. See the section called “Boolean variables”.

Template file structure

Let's start by looking at a trivial example of a FreeMarker file that could be used to generate a simple HTML document. In this example, the template expects two template variables, "`listOfItems`" (a sequence) and "`title`" (a string):

```
[#ftl]❶
[#setting locale="fr_FR"]❷
<!DOCTYPE html>
<html>
<head>
<title>${title?html}❸</title>
</head>
<body>
<h1>${title?html}</h1>

[#if listOfItems?has_content]❹
    <table>
    [#list listOfItems as item]❺
        <tr><td>${item?html}</td></tr>
    [/#list]
    </table>
[#else]
    <p>Liste vide.</p>
[/#if]

</body>
</html>
```

❶ FreeMarker files using *square bracket syntax* always start with the `[#ftl]` directive. Without this directive, the other FreeMarker tags in the template would be expected using "<" and ">" symbols, which tends to be unreadable when mixed with HTML tags.

❷ An example of a `[#setting]` directive, here explicitly setting the template locale to French (France), which would affect date formatting and number formatting for any such values pushed into the template.

As with the preceeding `[#ftl]` directive, the line containing this directive (and no other non-whitespace text) will *not* be included in the template output; the first line will contain the HTML DOCTYPE declaration.

❸ An example of the use of a template variable, *and* the use of a FreeMarker built-in. The template variable is an expression (in programming terms), starting with "`${`" and ending with "`}`". Between the curly brackets, the name of the variable appears (`title`), followed by the `?html` built-in (which ensures that any special HTML characters are rendered without corrupting the page and without compromising security).

Note that the same variable is referred to again, further down the page.

❹ The `[#if *expression*]` directive is used to mark a section of the template such that it is only used to generate output if the *`expression`* evaluates to `true`. Expressions can for example compare numbers (equality, greater than, less than, and so on), a string of text (equality, starts with, ends with, and so on), or as in this example, check that the list "has content" (meaning "is not `null` and has at least one item) by using the `?has_content` built-in. The `[#if]` directive requires a matching `[/#if]` directive to mark the end of the conditional section, and different cases can be handled between the two marker using `[#elseif *expression*]` and `[#else]` (used here to display a message in French to indicate that the list is empty or unavailable).

❺ The `[#list]` marks a section (ending with `[/#list]`) that will be repeated for every `item` in `listOfItems`. The name of the `item` variable is arbitrary, chosen here by the template author as part of the `[#list]` directive.

The above example is text-based. You can use any visual HTML editor or text editor to create your templates. Some editors may be aware of FreeMarker syntax, others will just treat the directives (in square brackets) and template variables as text in the document. If you prefer using a visual HTML editor, you should be able to avoid working with the text during almost all of the design process.

Template variables

Template variables have unique case-sensitive names, and contain typed data (such as text, numbers, dates, boolean true/false values, and arbitrary objects with properties). Some variables, such as objects and lists, can contain other variables. In the case of FreeMarker, the template engine "wraps" variables into a specific set of types for more uniform processing; for example, FreeMarker will convert various different Java data types into either numbers, dates, or collections which can be used in a consistent manner in the template.

Other template systems may process data differently, but internally, OpenAGE Viewer always provides the same data to all template systems (this remark is important to remember if you implement an OSGi plug-in using templates: you should not specifically preformat your data for FreeMarker, as this is done for you automatically in almost all cases).

Using scalar variables in templates

FreeMarker supports a specific set of scalar (single-value) variable types in templates, specifically: text (strings), numbers, dates, and booleans. Everything else (objects and collections) is at some point (explicitly or automatically) converted into one of the scalar variable types. For example, the template will generally render items in a collection individually, and properties of an object individually, instead of trying to convert an entire collection or object graph into a single standard string format.

String variables

With FreeMarker templates (which ultimately generate text output), string variables are generally the most common. As such, there are many useful built-ins for converting case, formatting, padding, searching, and splitting strings (see the section called “Built-ins for text”). All other scalar variable types have a `?string` built-in that can convert to this type.

Number variables

FreeMarker has only one numeric variable type, covering both integers and decimal numbers with a fractional part. You can define default formatting using settings directives, as follows (the quoted values are replaceable examples):

```
[#setting number_format="#,##0.00"]
[#setting locale="fr_FR"]
```

The symbols used to define number formats are specified by the `DecimalFormat`[1] class. In particular, a comma (",") is *always* the grouping separator, and a full-stop (".") is *always* the decimal separator. The effective locale determines how the symbols are output, so for example using the above `fr_FR` locale, in output the grouping separator becomes a space (" ") and the decimal separator becomes a comma (",").

Where a 0 (zero) appears, the result always includes the appropriate digit, and where a # (hash) appears, the result only includes a digit if there is a non-zero digit to the left. The symbols before the decimal point define the minimum number of digits (if you use "0" instead of "#"), and define the grouping symbol, but do *not* limit the number of digits. After the decimal point, the number of symbols (either

[1] http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html

"#" or "0") determines the maximum number of digits shown after the decimal point, rounding up or down the truncated digits using the half-even rule.

Using the above format with the `fr_FR` locale `12345.678` will be formatted as "12 345,68", and with the default locale will be "12,345.68".

Date variables

Date processing in templates can be problematic, due to limitations in the standard Java API that FreeMarker depends on. Specifically, Java uses the same `java.util.Date` object for three different concepts: *date (with time of day)*, *date (without time)*, and *time (without date)*. There is no way for FreeMarker to know which usage is intended based on the object type. Furthermore, care must be taken with time zones, because if the time zone is not UTC, then a variable containing the time (without the date) can be rendered "incorrectly" (as the time zone offset may be applied).

So, before trying to include a date value in a template, you tell FreeMarker which usage is expected for the variable, using built-ins (see the section called “?datetime”, the section called “?date”, and the section called “?time”).

You can then apply the required text formatting to the variable (restricting formatting to specific parts of the date, such as the localized name for the month or the day of the week, if necessary) using the other built-ins for this type (see the section called “Built-ins for dates”).

You can also define default formatting using settings directives, as follows (the quoted values are replaceable examples):

```
[#setting datetime_format="yyyy-MM-dd HH:mm:ss"]
[#setting date_format="EEEE, MMMM d"]
[#setting time_format="HH'h'mm"]
[#setting time_zone="Europe/Paris"]
[#setting locale="fr_FR"]
```

The symbols used to define date and time formats are specified by the `SimpleDateFormat`[2] class, and time zones are specified by the `TimeZone`[3] class.

The locale setting determines how text values are output: for example (using the above date format), with `fr_FR` a date could be output as "mercredi, juillet 31", but with `en_GB` the same date would be output as "Wednesday, July 31".

With the above time format example, note that the "h" separating hours and minutes must be quoted with single quotes (as the "h" character, when unquoted, is interpreted as the AM/PM time symbol). At 10:15 UTC, with the above time format and time zone, this would be output (without quotes) as 12h15 (when the current date is in daylight savings time in central Europe).

Boolean variables

Boolean variables can be either `true` or `false`. Unlike the other scalar types defined by FreeMarker, by default it is *not* possible to use a boolean variable directly in a template to create text. You must use a either a built-in (see the section called “Built-ins for booleans”) or define a "settings" directive. The reasoning for this is to force template authors to choose appropriate text in the template instead of "true" or "false", such as "on" or "off", "yes" or "no", "available" or "unavailable", "1" or "0", and so on.

The settings directive can be defined as follows:

```
[#setting boolean_format="vrai,faux"]
```

Using the above setting, booleans can be used in the template without built-ins (here, `true` would be output as "vrai", and `false` would be output as "faux"). You can define the setting and then override it with built-ins per usage of boolean variables, as appropriate.

[2] http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
[3] http://docs.oracle.com/javase/6/docs/api/java/util/TimeZone.html

Using sequences and hashes in templates

Sequences are lists of items referred to using a single template variable, usually when the number of items is not known in advance. A list of records in a table is a typical example. *Hashes* are also collections of items, but unlike sequences, each item (the *value*) is referred to using a *key*. A hash could for example use SKUs (stock-keeping units) as keys, and full product records as values. The items contained by sequences and hashes can be any of FreeMarker's scalar types, Java objects, or other nested sequences and hashes.

Templates process sequences by defining a section in the template to be applied to items in the sequence, using the `[#list]` directive. Hashes cannot be directly processed; they must be converted to sequences using either the `?keys` or `?values` built-ins (see the section called “Built-ins for hashes”). The traversal order of sequences can be modified using additional built-ins (see the section called “Built-ins for sequences”).

The syntax for stepping through a sequence was introduced when presenting FreeMarker (see the section called “Template file structure”). Going further, FreeMarker provides some other useful features when processing sequences.

```
[#assign edges = ["top","right","bottom","left"]]❶

[#list edges as e]❷
[#nt]❸ padding-${e}❹: 10px;  /* ${e_index}❺ */
[/#list]

margin: [#list edges as e]5px[#if e_has_next]❻ [#else];[/#if][/#list]
```

This will produce the following output:

```
padding-top: 10px; /* 0 */
padding-right: 10px; /* 1 */
padding-bottom: 10px; /* 2 */
padding-left: 10px; /* 3 */

margin: 5px 5px 5px 5px;
```

❶ The `[#assign]` directive lets us create a variable from within the template, without it having been pushed into the template by OpenAGE Viewer, so we use it here to illustrate the example, as a simple sequence of strings referred to using a new template variable named "`edges`". See the section called “Using the #assign directive”.

❷ The `[#list]` directive lets us define a section (ending with `[/#list]`) that will process the `edges` variable, and within the section (repeated for each item in the sequence), we refer to the item as "`e`" (following the "`as`" keyword in the assign directive). You can nest different `[#list]...[/#list]` sections within each other.

❸ The `[#nt]` directive applies to the line in which it appears, and means "no trim". In other words, we want FreeMarker to include the end-of-line after processing each iteration over items in the sequence. Without it, the output would appear on the same line.

❹ Each instance of "`e`" is a scalar string variable from the sequence, so we include it as-is directly in the output using the `${e}` syntax.

❺ Here, we use an automatic FreeMarker-specific feature: an implicit variable, based on the name of the loop variable with the "`_index`" suffix. This scalar number contains the 0-based iteration counter, so that you do not need to define and track this useful information separately.

❻ In the second loop over the list, we generate output in the same line; we want every item to be separated with a space, except the last one, which should be following by a semicolon (";"). We achieve this using another automatic FreeMarker-specific feature: another implicit variable, based on the name of the loop variable with the "`_has_next`" suffix. This scalar boolean contains `true` if there are more items to loop over, otherwise `false`, so that you do not need to define and track this useful information separately.

With hashes, looping over all items requires additional syntax, as follows:

```
[#assign edges = {"top":20,"right":10,"bottom":15,"left":50}]

margin: [#list edges?keys as k]${edges[k]}px[#if k_has_next] [/#if][/#list];
```

This time, we declare the collection within curly brackets ("`{...}`", and not the square brackets used with the preceeding sequence example) and each *key* is followed by a colon (":") and the *value* (values are not necessarily strings). The loop will iterate over keys in the order in which the are defined (note that this is not necessarily the case for all hashes added by Java code in OpenAGE Viewer or in OSGi plug-ins[4]). We loop over the hash's keys using the `?keys` built-in, and retrieve values by accessing the hash variable `edges` followed by the key variable within square brackets. This produces the following output:

```
margin: 20px 10px 15px 50px;
```

Note that FreeMarker has an important limitation: hashes only work well when hash keys are strings (text type). Any Java objects pushed into the template's data context and that implement the `java.util.Map` interface without using string keys, must be manipulated as Java objects and *not* as FreeMarker hashes.

Using Java objects in templates

The default data provided by OpenAGE Viewer is almost always based on Java objects (see the section called “Default data provided for templates”). The properties of these objects may be other objects, or be sequences or hashes, or scalar variables. More precisely, FreeMarker introspects Java objects using a JavaBeans[5]-style syntax (without going as far as indexed or bound properties), and then wraps as either sequences, hashes, scalar variables, or as other Java objects in their own template wrappers.

You can therefore also add your own Java objects (using an OSGi plug-in) into any template, and the following method signature patterns will be recognized as properties:

```
public T getSomeProperty();       // accessible as variable.someProperty
public boolean isSomeCondition(); // accessible as variable.someCondition
public T get(String x);           // accessible as variable.someDynamicProperty
```

The first pattern is the most common: a `public` method, with no parameters and starting with "get", returning any type of object (or primitive datatype) in place of *T*. The second pattern is a special case variant defined by the JavaBeans specification that allows boolean properties to start with "is" instead of "get". The third pattern is similar to the first in intent, except that the property name is passed as parameter "`x`" and the method is called "get"; this allows the Java object to support dynamic properties (such as field names from a database table) that are not known in advance, so that the template author can benefit from a more concise syntax.

You *cannot* access any other method as a property, and FreeMarker applies some security filters to make some special methods (such as those defined by `java.lang.Object`) inaccessible to template authors. It is not possible to access fields of an object directly, even if they are declared `public`.

It is bad practice (and a security risk) to attempt to call arbitrary methods of a Java object, to perform actions (such as sending an e-mail) or to modify state (such as updating a file or a database). Such actions should be securely performed in code prior to providing data to the template.

Using the #assign directive

The `[#assign]` directive (already seen in the section called “Using sequences and hashes in templates”) lets you define or derive variables in the template, without needing to push them into the template using Java code.

[4]Consider using `java.util.LinkedHashMap<String,V>` if you need to add an order-preserving hash from an OSGi plug-in.
[5] http://docs.oracle.com/javase/tutorial/javabeans/writing/properties.html

For example, when generated HTML within a loop or a macro, you might find it useful to define the HTML ID of a form `<input>` element:

```
[#assign fieldId="textbox${item_index+1}"]
<label for="${fieldId}">${fieldTitle?html}</label>
<input id="${fieldId}" name="${fieldId}" value="" size="40"/>
```

In the above example, we use the implicit `item_index` variable from the `[#list]` directive (which is 0-based, so we add 1) which will result in the first value for *`fieldId`* being "textbox1". We then proceed to use this calculated value in three places in the output, instead of recalculating it for each usage.

Using FreeMarker built-ins

FreeMarker is not limited to merging variable values into templates, it also allows functions to be used in templates. Just as spreadsheet software adds flexibility with functions, you can too. FreeMarker distinguishes two types of functions: those provided by the application (see the section called “Template functions”), and those provided by FreeMarker itself, called *built-ins*. Built-ins have their own specific syntax, and can be chained easily (a built-in can be applied to a template variable, and a second built-in can then be applied to the result of the first built-in, and so on).

For example, assuming that the variable `person.lastName` contains the text "d'Arcy", then the following template excerpt:

```
${person.lastName?upper_case?html}
```

...will result in the following output:

```
D&#39;ARCY
```

This works as follows: the template engine resolves the `lastName` property on the `person` variable, which yields the text "d'Arcy". That text is then passed to the `?upper_case` built-in, which transforms it into the text "D'ARCY". Then, the `?html` built-in receives the text "D'ARCY" and transforms it into "D&#39;ARCY". Finally, the template receives the text "D&#39;ARCY" and writes it directly into the output page.

In the above example, the order if the built-ins is irrelevant, but this is not always the case. When using built-ins, always think about what will happen as each built-in processes input and yields output at each step.

As FreeMarker does not allow `null` values to be output directly into a template (it considers this to be an error, to err on the side of caution), there is one special built-in that can be applied to any variable: `?has_content`. This built-in returns `false` if the value to which it is applied is `null`, and also returns `false` if it is applied to an empty string, sequence or hash; in all other cases, this built-in returns `true`.

You can also test variables to determine if they are of a specific FreeMarker variable type, using the following built-ins:

`?is_string`	Returns `true` if the variable is a FreeMarker string.
`?is_number`	Returns `true` if the variable is a FreeMarker number.
`?is_date`	Returns `true` if the variable is a FreeMarker date and/or time.
`?is_boolean`	Returns `true` if the variable is a FreeMarker boolean.
`?is_sequence`	Returns `true` if the variable is a FreeMarker sequence.
`?is_hash`	Returns `true` if the variable is a FreeMarker hash.

Note that the following description of FreeMarker built-ins in this document is *not* exhaustive. For a complete reference, refer to the online FreeMarker manual[6].

Built-ins for text

The following built-ins can be applied to scalar string variables, string properties of Java objects, and to the result of any other built-in or template function that produces a string.

?html

Converts a string into HTML format, so that page layout is not corrupted when the string contains characters that have special meaning to a web browser. Apart from helping to prevent corrupted layout, it is a critical step in securing your web page. **You must apply this systematically to all string variables when generating HTML.**

The following replacement rules are applied:

`<` is replaced with `&lt;`
`>` is replaced with `&gt;`
`&` is replaced with `&amp;`
`"` is replaced with `@quot;`
`'` is replaced with `&#39;`

?js_string

Converts a string into JavaScript format, so that scripts on the page work as expected, even when the string contains characters that have special meaning in JavaScript. Apart from helping to prevent script errors, it is a critical step in securing your web page. **You must apply this systematically to all string variables when generating JavaScript.**

The following replacement rules are applied:

`"` is replaced with `\"`
`'` is replaced with `\'`
`/` is replaced with `\/"`
characters under `0x20` are escaped

?length

Gets the number of characters, including whitespace, in the string to which this built-in is applied.

?upper_case

Gets the upper case equivalent of the string to which this built-in is applied.

For example:

```
${"Viele Grüße"?upper_case}
```

...will produce the following output in a German locale:

```
VIELE GRÜSSE
```

?lower_case

Gets the lower case equivalent of the string to which this built-in is applied.

?contains

Determines if the string to which this built-in is applied contains the specified substring, using case-sensitive matching rules.

[6] http://freemarker.org/docs/ref_builtins.html

This built-in requires a single parameter, and produces a boolean result. For example, assuming that "`v`" is a string variable:

```
[#if v?contains("order")]Found[#else]Not found[/#if]
```

...then when `v` is "Your order invoice", this will evaluate as `true`, but when `v` is "Your quote", this will evaluate as `false`. It will also evaluate `false` if `v` does not contain a case-matching occurrence, as would be the case if the value of `v` was for example "BORDER".

?index_of

Gets the 0-based index of the *first* occurrence of the specified string within the string to which this built-in is applied.

This built-in accepts up to two parameters, and produces a number as a result:

```
index_of( value, offset )
```

`value`	Required string. This is the substring to search for within the string to which this built-in is applied. Searching is case-sensitive, and performed from left-to-right.
`offset`	Optional number. If specified, should be greater than 0 and less than the length (minus one) of the string to which this built-in is applied. Searching starts at this offset (and can therefore return a match at this offset position, or after the offset), and continues rightwards.

The 0-based index of the first match (from the start of the string, or inclusively from the offset) is returned; when no match can be found, -1 is returned instead. For example:

```
${"abcabc"?index_of("bc")}
${"abcabc"?index_of("bc",2)}
${"xyz"?index_of("bc")}
```

...will produce the following output:

```
1
4
-1
```

?last_index_of

Gets the 0-based index of the *last* occurrence of the specified string within the string to which this built-in is applied.

This built-in accepts up to two parameters, and produces a number as a result:

```
last_index_of( value, offset )
```

`value`	Required string. This is the substring to search for within the string to which this built-in is applied. Searching is case-sensitive, and performed from right-to-left.
`offset`	Optional number. If specified, should be greater than 0 and less than the length (minus one) of the string to which this built-in is applied. Searching starts at this offset (and can therefore return a match at this offset position, or after the offset), and continues leftwards.

The 0-based index of the last match (from the end of the string, or inclusively from the offset) is returned; when no match can be found, -1 is returned instead. For example:

```
${"abcabc"?last_index_of("bc")}
${"abcabc"?last_index_of("bc",2)}
${"xyz"?last_index_of("bc")}
```

...will produce the following output:

```
4
1
-1
```

?substring

Gets part of a string, from a character index up to an (optional) character index or to the end of the string. The indices can be located using other built-ins (usually `?index_of` and `?last_index_of`).

This built-in accepts up to two parameters, and produces a string as a result:

```
substring( firstIndexInclusive, lastIndexExclusive )
```

`firstIndexInclusive`	Required number. Must be greater than 0 and less than the length (minus one) of the string to which this built-in is applied.
`lastIndexExclusive`	Optional number. If specified, must be greater than *`firstIndexInclusive`* and less than the length (minus one) of the string to which this built-in is applied.

For example:

```
1st: ${"Home Page Menu"?substring(0,4)}
2nd: ${"Home Page Menu"?substring(5,9)}
3rd: ${"Home Page Menu"?substring(10)}
```

...will produce the following output:

```
1st: Home
2nd: Page
3rd: Menu
```

?split

Derives a sequence by splitting the string around matches of a specified separator. The matching string is discarded; when two adjacent matches occur (with no text between), this adds an empty string to the sequence.

This built-in requires one parameter[7], and produces a sequence of strings as a result:

```
split( separator )
```

`separator`	Required string, such as " " or ",", used to identify boundaries in the string where splitting should occur.

For example:

```
[#list "one,two,,nine,ten"?split(",") as x]
[#nt] - "${x}"
[/#list]
```

...will produce the following output:

```
 - "one"
```

[7]The `?split` function actually accepts an optional second parameter, allowing regular expression matching, however this feature is beyond the scope of this manual. Refer to the official on-line FreeMarker documentation.

```
 - "two"
 - ""
 - "nine"
 - "ten"
```

?left_pad

Gets a string with padding to the left of string to which this built-in is applied, such that the length of the returned string is at least the specified length. If applied to a string with a length that is already greater than or equal to the specified length, then no padding occurs and the returned string is the same as the string to which the built-in was applied. By default, spaces are used for padding, but an alternative padding character may be specified.

This built-in accepts up to two parameters, and produces a string as a result:

```
left_pad( padToLength, padWith )
```

padToLength	Required number. The minimum required length for the string when padding. If the string to which this built-in is applied is longer than this, then it is returned as-is (no padding, no truncation).
padWith	Optional string. If this is a single character, it is simply repeated as required to pad the string to the required length. If it contains more than one character, then it is repeated as a pattern (the overall padding length remains the same, the padding is simply composed of a repeating pattern).

For example (using enclosing parentheses to show the effect of padding):

```
(${"Pad me"?left_pad(7)})
(${"Naboo"?left_pad(7,"-")})
(${"Tatooine"?left_pad(7,"-")})
(${"Dashed"?left_pad(12,"= ")})
```

...will produce the following output:

```
( Pad me)
(--Naboo)
(Tatooine)
(= = = Dashed)
```

?right_pad

Gets a string with padding to the right of string to which this built-in is applied, such that the length of the returned string is at least the specified length. If applied to a string with a length that is already greater than or equal to the specified length, then no padding occurs and the returned string is the same as the string to which the built-in was applied. By default, spaces are used for padding, but an alternative padding character may be specified.

This built-in accepts up to two parameters, and produces a string as a result:

```
right_pad( padToLength, padWith )
```

padToLength	Required number. The minimum required length for the string when padding. If the string to which this built-in is applied is longer than this, then it is returned as-is (no padding, no truncation).
padWith	Optional string. If this is a single character, it is simply repeated as required to pad the string to the required length. If it contains more than one character, then it is repeated as a pattern (the overall padding length remains the same, the padding is simply composed of a repeating pattern).

For example (using enclosing parentheses to show the effect of padding):

```
(${"Pad me"?right_pad(7)})
(${"Naboo"?right_pad(7,"-")})
(${"Tatooine"?right_pad(7,"-")})
(${"Dashed"?right_pad(12,"= ")})
```

...will produce the following output:

```
(Pad me )
(Naboo--)
(Tatooine)
(Dashed= = = )
```

?trim

Gets a string with no leading or trailing whitespace.

For example (using enclosing parentheses to show the effect of trimming):

```
(${"  hello world  "?trim})
```

...will produce the following output:

```
(hello world)
```

Built-ins for numbers

The following built-ins can be applied to scalar numeric variables, numeric properties of Java objects, and to the result of any other built-in or template function that produces a number.

?c

Converts a number variable to a string in *computer format*, without applying any effective number format settings. In other words, there will be no grouping separator, no decimal rounding, no left-padded zeros, and no right-padded zeros. For example, assuming that "`pi`" is a template variable containing the value of the `double` constant `java.lang.Math.PI`:

```
${pi?c}
${(pi * 1000000)?c}
```

...then the output will be:

```
3.141592653589793
3141592.653589793
```

?string

Converts a number to a string using either the default format and locale settings for the template, or using a format specified as a parameter.

This built-in therefore accepts one optional parameter, and produces a string as a result:

```
string( format )
```

`format` Optional format string. If defined, uses the formatting symbols specified by the `DecimalFormat`[8] class. Refer to the section called "Number variables" for more information.

[8] http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html

You can also specify named formats:

`currency`	This formats the number using the current and default formatting rules for the effective locale. This can also be written as: `?string.currency`
`percent`	This multiplies the value by 100 then formats it as a percentage, using the effective number format for the decimal separator. Percentages between 0 and 100% should therefore be provided in the range 0.0 to 1.0. This can also be written as: `?string.percent`

For example:

```
${12.345?string.currency}
${0.77?string.percent}
${42?string("#000.0#")}
${92400?string("#,###")}
```

...produces the following output with the `fr_FR` locale:

```
12,34 €
77 %
042,0
92 400
```

...and produces the following output with the `en_GB` locale:

```
£12.34
77%
042.0
92,400
```

?round

Rounds a number variable. If the fractional part ends with .5, then this rounds towards infinity (either positive or negative, depending upon whether the number is positive or negative).

?number_to_date

Converts a number with `long` (64-bit integer) precision to a date. This may be useful in some situations with Java objects.

This produces a FreeMarker *date (without time)* variable, which can be output using a chained built-in. For example:

```
${1375354728519?number_to_date?iso_local}
```

...will convert *`1375354728519`* (a number of milliseconds since the Unix reference time) to a date variable, then (using the chained `?iso_local` built-in) produce the following output (if the time zone is UTC):

```
2013-08-01
```

?number_to_time

Converts a number with `long` (64-bit integer) precision to a time of day. This may be useful in some situations with Java objects.

This produces a FreeMarker *time (without date)* variable, which can be output using a chained built-in. For example:

```
${4585000?number_to_time?string("HH:mm:ss z")}
```

...will convert `4585000` (a number of milliseconds from midnight) to a time variable, then (using the chained `?string(format)` built-in) produce the following output (if the time zone is UTC):

```
01:16:25 UTC
```

?number_to_datetime

Converts a number with `long` (64-bit integer) precision to a date with time of day. This may be useful in some situations with Java objects.

This produces a FreeMarker *date with time* variable, which can be output using a chained built-in. For example:

```
${1375354728519?number_to_datetime?iso_local}
```

...will convert `1375354728519` (a number of milliseconds from midnight) to a date (with time) variable, then (using the chained `?iso_local` built-in) produce the following output (if the time zone is UTC):

```
2013-08-01T10:58:48Z
```

Built-ins for dates

The following built-ins can be applied to scalar date variables, date properties of Java objects, and to the result of any other built-in or template function that produces a date (or data that can be coerced to a date).

?date

Converts a Java `Date` object to a FreeMarker *date (without time)* variable.

?time

Converts a Java `Date` object to a FreeMarker *time (without date)* variable.

?datetime

Converts a Java `Date` object to a FreeMarker *date with time* variable.

?string

Converts a FreeMarker date (either *date (without time)*, *time (without date)*, or *date with time*) to a string using either the default format and locale settings for the template, or using a format specified as a parameter.

This built-in therefore accepts one optional parameter, and produces a string as a result:

```
string( format )
```

`format`	Optional format string. If defined, uses the formatting symbols specified by the `SimpleDateFormat`[9] class.

9 http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html

For example:

```
${.now?datetime?string("d/M/yyyy HH:mm:ss")}
```

...can produce the following output:

```
1/8/2013 13:24:31
```

?iso_utc

Converts a date to a ISO format string in the UTC timezone.

For example, even after setting the time zone to something other than UTC:

```
[#setting time_zone="Europe/Paris"]
${.now?datetime?iso_utc}
```

...could (on August 1st, 2013, at 13:09:21 CET) result in output such as:

```
2013-08-01T11:09:21Z
```

?iso_local

Converts a date to a ISO format string in the local timezone.

For example, when setting the time zone to something other than UTC (here, Central European Time):

```
[#setting time_zone="Europe/Paris"]
${.now?datetime?iso_utc}
```

...could (on August 1st, 2013, at 13:09:21 CET) result in output such as:

```
2013-08-01T13:09:21+02:00
```

Built-ins for booleans

The following built-ins can be applied to scalar boolean variables, boolean properties of Java objects, and to the result of any other expression, built-in or template function that produces a boolean. Boolean values can be either `true` or `false`, often used in practice to model "yes or no" and "on or off".

?c

Converts a boolean variable to *computer format*, either "true" or "false".

?string

Converts a boolean to a specified string, based on the value. For example, assuming that "`v`" is a boolean variable:

```
${v?string("Oui","Non")}
```

...then output when "`v`" is `true` will be "Oui", and when `false` will be "Non".

Built-ins for sequences

The following built-ins can be applied to sequences, properties of Java objects that return instances of `java.util.List`, and to the result of any other built-in or template function that produces a list.

?size

Gets the number of items in the sequence, or 0 if the sequence is empty.

?first

Gets the first item in a sequence, or triggers a template processing error if the sequence is empty.

?last

Gets the last item in a sequence, or triggers a template processing error if the sequence is empty.

?join

Concatenates all items into a single string, attempting automatic string conversion for values in the sequence if required. Any `null` items in the sequence are ignored.

This built-in accepts up to three parameters, and produces a string result:

```
join( separator, emptyValue, listEnding )
```

separator	Required. This is a string containing the text to add between items in the sequence.
emptyValue	Optional. This is a string, displayed when the sequence is empty.
listEnding	Optional, must follow preceeding parameters. This is a string containing text to add after the last item in the sequence.

For example:

```
[#assign choices=["Previous","Next","Finish","Cancel"]]
[#assign nothing=[]]

${choices?join("/","-",".")}
${nothing?join("/","-",".")}
```

...will produce the following output:

```
Previous/Next/Finish/Cancel.
-
```

?seq_contains

Determines if a sequence contains the specified value.

This built-in requires a single parameter, and produces a boolean result.

```
seq_contains( scalarValue )
```

scalarValue	The value to search for in the sequence to which this built-in is applied. The specified value must be a scalar type (it cannot be an object, hash, or sequence).

For example:

```
[#assign colors=["red","green","blue"]]

${colors?seq_contains("red")?string("Found","Can't find")} red.
${colors?seq_contains("yellow")?string("Found","Can't find")} yellow.
```

...will produce the following output:

```
Found red.
```

```
Can't find yellow.
```

?reverse

Gets a sequence containing the same items, but in reverse order. Produces a new view of the sequence without modifying the sequence to which it was applied.

?sort

Gets a sequence containing the same items, but in ascending order (for descending order, apply the `?reverse` built-in to the result of this built-in). This can only be applied to sequences containing scalar items, all of the same type. Produces a new view of the sequence without modifying the sequence to which it was applied.

?sort_by

Gets a sequence containing the same items, but in ascending order of values of the specified hash key (for descending order, apply the `?reverse` built-in to the result of this built-in). This can only be applied to sequences where each item is a hash with a key matching the specified name.

This built-in requires a single parameter, and produces a new view of the sequence without modifying the sequence to which it was applied.

```
sort_by( keyName )
```

`keyName`	The name of the key that must exist for each item in the sequence, where each item in the sequence is a hash.

For example:

```
[#assign cities=[
 {"city":"Paris","dept":75},
 {"city":"Marseille","dept":13},
 {"city":"Bordeaux","dept":33}
]]

[#list cities?sort_by("city") as v]${v["city"]}[#if v_has_next], [/#if][/#list].
[#list cities?sort_by("dept") as v]${v["city"]}[#if v_has_next], [/#if][/#list].
```

...will product the following output:

```
Bordeaux, Marseille, Paris.
Marseille, Bordeaux, Paris.
```

?chunk

Splits a sequence into a collection of sequences with a specified maximum number of items. This can split a sequence into tabular format, with a fixed maximum number of columns and an arbitrary number of rows.

This built-in accepts up to two parameters, and produces a sequence of sequences as a result (one sequence per tabular row).

```
chunk( columnsPerRow, filler )
```

`columnsPerRow`	Required. The number of items per subsequence (the number of columns per row).
`filler`	Optional. The item to append to the last subsequence (one or more times) such that it has the same number of items as specified by *`columnsPerRow`*, when

the number of items in the sequence to which this function is applied is not an exact multiple of `columnsPerRow`.

For example:

```
[#assign alphabet=["alpha","beta","gamma","delta","epsilon"]]

[#list alphabet?chunk(3,"--") as row]
[#nt][#list row as col]${col} [/#list]
[/#list]
```

...will produce the following output:

```
alpha beta gamma
delta epsilon --
```

Built-ins for hashes

The following built-ins can be applied to hashes, properties of Java objects that return instances of `java.util.Map`, and to the result of any other built-in or template function that produces a map or hashtable with keys and values. In practice, you will almost always convert a hash to a sequence for processing using either `?keys` or `?values`.

?keys

Gets a sequence containing all the keys defined by the hash. Some hashes will return keys in insertion order, some will return keys sorted into ascending or descending order; many other hashes will return keys in an arbitrary (seemingly random) order.

This built-in is the most common way to iterate over all entries in the hash, for use with the `[#list]` directive.

?values

Gets a sequence containing all the values contained in the hash. The values are most often returned in an arbitrary order. The returned sequence can be used in the same ways as any other sequence.

Template macros

Template macros are fragments of templates that can be called from one or more places elsewhere in a template (or even from other template files) to produce a section of output. Macros can have parameters, including dynamic parameters and parameters with default values. Template files can define any number of macros.

Let's start with a simple example, with no parameters:

```
[#macro hello]
[#rt] Hello!
[/#macro]

[#list 1..3 as x][@hello/][/#list]
```

The above example defines the macro body using the `[#macro macroname]` directive. It then defines a number sequence with a range from 1 to 3 (inclusive), which therefore loops three times over the macro invocation (using the `[@macroname/]` syntax within a `[#list]` directive. This produces the following output:

```
 Hello! Hello! Hello!
```

Note the use of the `[#rt]` directive, or "right trim", which tells the macro to ignore the trailing whitespace on the line, and the line return, so that output continues on the same line.

Macros with parameters

The following example illustrates how a macro can be defined and called with named parameters, including a second optional parameter with a default value:

```
[#macro quote q a="Anonymous"]
<blockquote>
<em>${q?html}</em>
 -- ${a?html}
</blockquote>
[/#macro]

[@quote "Time flies like an arrow.  Fruit flies like a banana."/]
[@quote "The Guide is definitive. Reality is frequently inaccurate." "D. Adams"/]
```

This produces the following HTML output:

```
<blockquote>
<em>Time flies like an arrow.  Fruit flies like a banana.</em>
 -- Anonymous
</blockquote>
<blockquote>
<em>The Guide is definitive. Reality is frequently inaccurate.</em>
 -- D. Adams
</blockquote>
```

The above output illustrates usage of the macro with different parameters, and in the case of the first invocation, usage of a default value when the second parameter was not specified.

The next example combines named parameters and dynamic parameters:

```
[#macro img src extra...]
<img src="${src?html}"
 [#list extra?keys as attr]
 ${attr}="${extra[attr]?html}"
 [/#list]
    />
[/#macro]

[@img src="logo.png" width=100 height=50 alt="Logo"/]
```

The macro `img` is invoked with one named parameter (`src`) and three dynamic parameters (`width`, `height`, and `alt`), where the dynamic parameters are not all of the same type. The macro signature identifies the dynamic parameter as "`extra...`", where "..." indicates that the parameter is in fact a hash (named `extra`) that will collect the remaining parameters, and which can be iterated over (in an arbitrary order) using the `?keys` built-in. This produces the following output:

```
<img src="logo.png"
    height="50"
    alt="Logo"
    width="100"
    />
```

Including other templates

Template files can *include* the content of other template files using the `[#include]` directive. Inclusion is dynamic, and the name of the file to include is a runtime parameter (it is not necessarily a fixed string). Files can be included using a path to any resource in the same template package, or even refer to resources in any other installed template package.

The syntax of this directive is as follows:

```
[#include path parse=true]
```

path	Required. Can either be a quoted literal string, such as "include.ftl", or an expression that produces a string result. The path can be relative or absolute.
parse	Optional. If specified, must be either `true` or `false`. When parsed (the default), it must be a valid FreeMarker file, and the included file can access variables in the current template. When false, the file is treated as plain text.

Some usage examples:

```
[#-- a FreeMarker template from the current directory --]
[#include "footer.ftl"]

[#-- a static text file containing a copyright notice --]
[#include "blurb/copyright.txt" parse=false]

[#-- a shared navigation header, from another package --]
[#include "/sharedthemepack/corporate/sitenavigation.ftl"]

[#-- a dynamic include,based on a "language" variable --]
[#include "content_"+language+".ftl"]
```

Importing macros from other templates

Template files can *import* the macros defined by other template files using the `[#import]` directive. Unlike the `[#import]` directive, it is not designed to copy the contents of the file; it treats the imported file as a library of macros that can be called.

The syntax of this directive is as follows:

```
[#import path as namespace]
```

path	Required. Can either be a quoted literal string, such as "library.ftl", or an expression that produces a string result. The path can be relative or absolute.
`namespace`	Required. This is the name of a hash that will contain all the macros imported from the template file specified by the *path* parameter. Unlike the previous parameter, this is not a string expression (the namespace is fixed, not dynamic).

A hypothetical example of usage:

```
[#-- Import two hypothetical macro libraries --]
[#import "library/forms.ftl" as forms]
[#import "../sharedthemepack/corporate/layout.ftl" as layout]

[#-- Use functions, prefixed with hashes, to build page --]
[@layout.header title="Login"]
[@forms.username id="principal"]
[@forms.password id="credentials"]
[@forms.button type="submit" label="${dict.login}"]
[@layout.footer office="Courbevoie"]
```

The above example shows how a template could be assembled using imported macro libraries. Here, one library provides the general corporate page layout (header, footer), which can be rendered by calling functions with parameters. Then, we use another library that hides the verbosity of form layout (fields, labels, standard attributes, alignment, and action buttons).

Template functions

OpenAGE Viewer provides a default set of template functions, and makes it possible to plug in additional template functions. Template functions generally transform existing data (for example, formatting text, numbers, and dates) or can calculate new content. They are also commonly used to create context-aware links between elements (for example, creating hypertext links for navigation that apply to the currently-connected user).

In the following sections, each default template function is described as follows:

```
function❶ ( param:Type❷ , param:Type❸ [, param:Type❹] )
```

❶ The function name appears before the list of parameters (also referred to as *arguments*). In the following function descriptions, the name of the function is used in place of the literal string "function" above.

❷ Most functions (but not all) expect at least one parameter. In all cases, the function name is shown followed by parentheses. When the function expects one or more parameters, these are shown within the parentheses; when there are multiple parameters, each parameter is separated by an intermediate comma. The name (here, "param") has no meaning to the function, it is simply shown for reference. In the same way, the ":Type" suffix is not expected, but is shown to indicate the data type (such as `String`, for text).

❸ Example of how a second required parameter can be specified in the argument list, separated from other parameters with a comma.

❹ Optional parameters are shown within square brackets. These brackets are shown in examples to identify optional parameters but *must not* be used within templates. This indicates that the parameter may be processed by the template function if provided, but is not required in all cases.

When used with the FreeMarker template system, functions can be combined with other FreeMarker functions, variables, and built-ins. For example:

```
<p>${nl2br(page.title?html)}</p>
```

The `${...}` syntax defines where the block of FreeMarker content occurs; within that block, FreeMarker interprets the text as commands and as variable references. The `nl2br` function will be passed the value of the hypothetical data `page.title` *after* it is transformed safely to HTML by FreeMarker's `?html` built-in (as the aim of `nl2br` is to convert new line characters into `<br/>` tags, the HTML conversion must be performed *first*, otherwise the output would be `&lt;br/&gt;`).

Default template functions

nl2br

The `nl2br` template function has the following syntax:

```
nl2br(text:String)
```

This function converts new line characters into HTML `<br/>` elements, so that the original data can store plain text without needing to store HTML markup. Multiple consecutive new line characters create multiple `<br/>` tags, with intermediate `&nbsp;` entities being generated as necessary.

For example, given the following input text:

```
This is a subtitle.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr...
```

...the `nl2br` function will produce the following output text:

```
This is a subtitle.<br />
&nbsp;<br />
Lorem ipsum dolor sit amet, consetetur sadipscing elitr...
```

This function can be used in any context.

addr

The `addr` template function has the following syntax:

```
addr(relativeUri:String|URI)
```

This function converts a string or a relative URI into an instance of `IAddress`, an interface defined by the API for working with addresses relative to (and not including) the server path, for use with other template functions or for accessing component parts of the address. It will neither add nor remove the server path, and must not contain the protocol, hostname, port, or the query string. The specified string must start with a forward slash ("/").

The resulting `IAddress` instance is recognized by other template functions as being relative the configured server path, and not as relative to the server root.

This function can be used in any context.

sid

The `sid` template function has the following syntax:

```
sid(url:String|URI|URL|IAddress)
```

This function encodes the session ID into the absolute or relative URL passed as a parameter to the template function. The `url` parameter may be specified as the output of another link-producing function. It should be used for all internal links when the default session cookie settings are effective (refer to Options for `server.using-session-cookies`).

For example, assuming the function is invoked when processing a request for a user with an active session (with a session token of "`abcd1234`"), then the following call to the `sid()` template function with an example URL:

```
sid('/report.pdf?orientation=landscape')
```

...would produce:

```
/report.pdf;jsessionid=abcd1234?orientation=landscape
```

No transformation occurs when this function is invoked when there is no active session.

The "`url`" parameter can be of Java type `String`, `URI`, `URL`, or `IAddress`.

This function can only be used in the context of processing an HTTP call, as it must be able to access the currently-connected user's session information.

uriHome

The `uriHome` template function has the following syntax:

```
uriHome()
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the current user's home page, which by default is the default dashboard. The generated URI automatically includes the session ID, if necessary.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI and determine the currently-connected user.

## uriDashboard

The `uriDashboard` template function has the following syntax:

```
uriDashboard()
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the current user's default dashboard page. The generated URI automatically includes the session ID, if necessary.

If the URI cannot be determined using the `IPageRouter` service, this returns the hash location `#uriDashboard`. Hash location updates can be handled using the `onhashchange` event of modern web browsers.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI and determine the currently-connected user.

## uriList

The `uriList` template function has the following syntax:

```
uriList(list:IListPage|IListView [, route:String] [, page:int [, pageSize:int]])
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the specified list page. The generated URI automatically includes the session ID, if necessary.

The optional *`route`* parameter can be used to select a non-default route, if such as route has been configured. The default route can be selected by omitting this parameter.

The optional *`page`* parameter specifies the page number as a positive integer, for paginated lists. The default page number is 1.

The optional *`pageSize`* parameter specifies the number of items per page as a positive integer, for paginated lists. The default number of items per page is 50. This optional parameter can only be specified if the *`page`* parameter is specified before it.

If the URI cannot be determined using the `IPageRouter` service, this returns the hash location `#uriList`. Hash location updates can be handled using the `onhashchange` event of modern web browsers.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI and determine the currently-connected user.

## uriForm

The `uriForm` template function has the following syntax:

```
uriForm(form:IFormPage|IItem|IKey|ISource [, route:String])
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the specified form page. If the first parameter is an instance of `ISource`, the form page can be used to create a new record; in all other cases, the form page is designed for viewing and/or editing an existing record. The generated URI automatically includes the session ID, if necessary.

The optional *`route`* parameter can be used to select a non-default route, if such as route has been configured. The default route can be selected by omitting this parameter.

If the URI cannot be determined using the `IPageRouter` service, this returns the hash location `#uriForm`. Hash location updates can be handled using the `onhashchange` event of modern web browsers.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI and determine the currently-connected user.

uriSignOn

The `uriSignOn` template function has the following syntax:

```
uriSignOn()
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the first valid configured definition of a password-based page design. If no such authentication service is available (for example, with a public site or a site using SSO), then this function will be unable to generate a valid URI. The generated URI automatically includes the session ID, if necessary.

If the URI cannot be determined, this returns the hash location `#uriSignOn`. Hash location updates can be handled using the `onhashchange` event of modern web browsers.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI.

uriSignOff

The `uriSignOff` template function has the following syntax:

```
uriSignOff()
```

This function generates a URI (relative hyperlink) that can be embedded into an HTML `<a>` element to link to the first valid configured sign-off URI of a password-based page design. If no such authentication service is available (for example, with a public site or a site using SSO, or an invalidation definition), then this function will be unable to generate a valid URI. The generated URI automatically includes the session ID, if necessary.

If the URI cannot be determined, this returns the hash location `#uriSignOff`. Hash location updates can be handled using the `onhashchange` event of modern web browsers.

This function can only be used in the context of processing an HTTP call, as it must be able to construct an appropriate URI.

udict

The `udict` template function has the following syntax:

```
udict(dictionary:String|UUID|IDictionary [, key:String [, formatParameters...]])
```

This function can either return a reference to a translation dictionary (an instance of `ITemplate-Dictionary`), or (more usefully) directly retrieve a text translation identified by a specific translation key. If the text is a template, and appropriate formatting parameters are specified, then they will be merged with the template. Furthermore, if any formatting parameters are themselves translatable (are instances of `ITranslated`, such as `ICell` or `ISource`), then they will be translated first before being merged with the template.

The translation is selected by first checking the request for an authenticated user (defining a locale preference), then falling back to any information provided using HTTP request headers (specifically, `Accept-Language`). If it is still not possible to select a translation, a neutral locale will be assumed. However, if an instance of `ITemplateDictionary` is used as the first parameter, then this will define the locale instead.

This function can only be used in the context of processing an HTTP call, as it must be able to determine the locale to use to select translations.

isTextType

The `isTextType` template function has the following syntax:

```
isTextType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `ITextType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isIntegerType

The `isIntegerType` template function has the following syntax:

```
isIntegerType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IIntegerType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type contain signed numbers (with no decimal point) with up to 20 digits (the exact number of digits will usually depend on database constraints). The default precision is 10 digits.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isDecimalType

The `isDecimalType` template function has the following syntax:

```
isDecimalType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IDecimalType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type contain signed numbers, and can have numbers after the decimal point. They are suitable for use in financial applications. The range of numbers that can be stored in such objects is in principle arbitrary, but in practice limited by database constraints (varying from one database to another). Generally, this can potentially store more digits than `IIntegerType` objects.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isFloatType

The `isFloatType` template function has the following syntax:

```
isFloatType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IFloatType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type contain signed numbers and have a decimal point. They can store very large ranges of numbers (and are therefore suited to scientific applications), and are memory-efficient (compact) and processor-efficient (fast). However, this comes at the cost of accuracy (such objects *are* very accurate, but are *not* absolutely accurate), and therefore such objects should not be used in financial applications.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isDateTimeType

The `isDateTimeType` template function has the following syntax:

```
isDateTimeType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IDateTimeType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type contain both date *and* time information. Care should be taken when working with multiple time zones (for example, databases generally are inadequate at storing time zone offsets and working with daylight savings rules).

Such objects are *not* equivalent to objects implementing or referring to `IDateType` or `ITimeType`.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isDateType

The `isDateType` template function has the following syntax:

```
isDateType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IDateType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Such objects are *not* equivalent to objects implementing or referring to `IDateTimeType`.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isTimeType

The `isTimeType` template function has the following syntax:

```
isTimeType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `ITimeType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Such objects are *not* equivalent to objects implementing or referring to `IDateTimeType`.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isBinaryType

The `isBinaryType` template function has the following syntax:

```
isBinaryType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IBinaryType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type usually store uploaded files.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isImageType

The `isImageType` template function has the following syntax:

```
isImageType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IImageType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type usually store picture files.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isBooleanType

The `isBooleanType` template function has the following syntax:

```
isBooleanType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IBooleanType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type usually store "yes or no" information (note that they can also be undefined, storing neither "yes" nor "no").

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isUuidType

The `isUuidType` template function has the following syntax:

```
isUuidType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IUuidType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

There are no template objects of this type when using the default OpenAGE connector. Such objects may exist with other connector types, and are usually limited to unique computer-generated identifiers.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

isArrayType

The `isArrayType` template function has the following syntax:

```
isArrayType(variable:IData|ICell|IType)
```

This function determines if a given variable implements the `IArrayType` interface (or refers to it), so that a template can process the associated data appropriately. When the `variable` implements (or refers to) this interface, this function returns `true`. If the `variable` is `NULL` or not one of the specified variable types (`IData`, `ICell`, or `IType`), then this returns `false`.

Template objects of this type contain embedded data sets, and can be used to store tabular data within a form.

TODO: describe properties of this type, and the associated data interface.

This function can be used in any context, and can be used in `if...else` blocks.

Chapter 6. Customizing page navigation

Default navigation

TODO: describe sign-on, sign-off, home and dashboard, master, detail.

Overriding default navigation

TODO: describe custom routes, path segment translation, path segment markers, hard-coded path context variables.

Chapter 7. Creating an OSGi plug-in

Quick introduction to OSGi bundles

TODO: describe basics, then link to other resources

Creating a page processor callback

TODO: provide some simple examples

Creating a page asset bundle

TODO: provide some simple examples, based on JavaScript libraries or CSS grids.

Creating template functions

TODO: examples of HTTP template functions, and non-HTTP template functions (usable for e-mail).

Creating content without templates

TODO: describe IAddressableService usage, for example a chart renderer.

Overridding page routing

TODO: describe IPageSelector, IPageRouteSelector, IPageRouteResolver.

Filtering HTTP requests

TODO: describe IFilteringService, and IPageGuardPredicate.

Part III. Reference

Appendix A. Configuration files

active-bundles.xml

This file defines which OSGi bundles should be activated at startup, and which bundles should be excluded from automatic startup. It is created automatically at startup if not present, but is not automatically updated thereafter. Bundles are referred to using their symbolic name. If a bundle is not included in this file, it will be automatically started; consequently, this file may be edited to list only those that should be excluded from automatic startup (by setting the associated entry value to *`false`*). Changes to this file take immediate effect (unless the `system.config` bundle is stopped).

The default file will be similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="system.api">true</entry>
<entry key="system.config">true</entry>
<entry key="system.connector">true</entry>
<entry key="system.defaultdesign">true</entry>
<entry key="system.design">true</entry>
<entry key="system.formatting">true</entry>
<entry key="system.freemarker">true</entry>
<entry key="system.fs">true</entry>
<entry key="system.http">true</entry>
<entry key="system.jmx">true</entry>
<entry key="system.locale">true</entry>
<entry key="system.themes">true</entry>
<entry key="system.views">true</entry>
</properties>
```

For applications that do not require the default navigation system nor the default visual theme, this file could be replaced with the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="system.defaultdesign">false</entry>
<entry key="system.design">false</entry>
</properties>
```

The above example excludes two specific bundles, by setting the entry value to *`false`*. All other bundles will be started automatically. If these bundles are not required for your application, you should deactivate them *before* configuring a connector (as this will prevent these bundles from needlessly creating additional configuration files when the connector is activated). Going further, for applications that do not require the standard OpenAGE connector, the following entry could also be added:

```
<entry key="system.connector">false</entry>
```

This will effectively prevent installation the connector setup interface (see the section called “Configuring the default connector”), and make the standard connector unavailable to plugins. If the connector is not available, you can implement your own connector or construct an application without a connector (for example, for directly accessing a database without leveraging OpenAGE's features).

Note that bundles that are excluded from automatic startup may be resolved transitively when they export packages that are imported by one or more bundles that are included in automatic startup.

system.connector.xml

The role of this file is to configure options for the standard connector implementation.

This file is not available by default; all of the options that it contains have default values. It is created automatically if any default value is overridden using the API or using a JMX console. It can also be created and updated manually, with any changes taking immediate effect (without requiring an application restart). All available options are presented in the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="setup.interface.enable">true</entry>
<entry key="network.connector.pool.size">16</entry>
<entry key="network.connector.clock.driftLimit">20</entry>
<entry key="network.connector.connectionTimeout">250</entry>
<entry key="network.connector.socketTimeout">5000</entry>
<entry key="network.data.cache.diskThreshold">32768</entry>
<entry key="network.data.compress.auto">true</entry>
<entry key="network.data.format.binary">true</entry>
<entry key="authn.fail.logLevel">WARN</entry>
<entry key="authn.trust.keySpec">keySpec</entry>
</properties>
```

Options for `setup.interface.enable`

`true`	Enables the default HTML connector setup interface (see the section called “Configuring the default connector”) when no connector (standard or any other implementation) is available. This is the default setting.
`false`	Disables the default HTML connector setup interface. If a connector is required and has not yet been configured, it must be configured using either a configuration file (see the section called “system.connector_endpoint.xml”), or by using the API or using a JMX console.

Options for `network.connector.pool.size`

`size`	A positive integer, greater than or equal to 1 (with 16 being the default value). Defines how many concurrent HTTP connections can be used to connect to the OpenAGE server.

Options for `network.connector.clock.driftLimit`

`limit`	An integer, greater than or equal to 0 (with 20 being the default value, and 0 being recommended). Defines (as a number of milliseconds) how much measurable clock desynchronization is permitted between servers, when OpenAGE is not running on the same server as OpenAGE Viewer. Clock drift is more of a problem than latency; poor performance (due to network or server load and capacity) is obviously not desirable, but clock drift can cause connector data caches to become inconsistent and unreliable. When significant clock drift (i.e.: above this defined limit) is detected (detection is automatic), the connector cache will be automatically deactivated to improve reliability and consistency, but at the cost of reduced performance. If this happens, a warning will be written to `system.log` (see the section called “The "logs" directory”).

Options for `network.connector.connectionTimeout`

`timeout`	An integer, greater than or equal to 20 (with 250 being the default value). Defines (as a number of milliseconds) the network connection timeout used by the connector.

Options for `network.connector.socketTimeout`

`timeout`	An integer, greater than or equal to 20 (with 5000 being the default value). Defines (as a number of milliseconds) the network socket timeout used by the connector.

Options for `network.data.cache.diskThreshold`

`threshold`	An integer, either equal to 4096 or a multiple thereof (with 32768 being the default value). Defines (as a number of bytes) the threshold that will trigger the connector to buffer data on disk instead of in memory. Low values reduce performance and consume extra disk space, whereas high values improve performance but increase memory pressure (beyond a certain point, this will also reduce performance as it will force excessive garbage collection and cache invalidation, and in some cases cause the operating system to make heavy use of memory paging).

Options for `network.data.compress.auto`

`true`	The connector will automatically compress buffered, cached, and transmitted data with this setting, at the cost of a small processing overhead. This is the default setting.
`false`	The connector will not compress data with this setting, reducing CPU load (but with little measurable difference given that network performance will remain a bottleneck). This setting makes it easier to inspect transmitted data for development and debugging purposes.

Options for `network.data.format.binary`

`true`	The connector will use an optimized binary format for sending and receiving data, and will compress UUIDs. This is the default setting.
`false`	The connector will use a plain text format for sending and receiving data (with the exception of binary objects, such as file data).

Options for `authn.fail.logLevel`

`WARN`	The connector will log warnings (in `system.log`) when an authentication attempt fails. This is the default setting.
`INFO`	The connector will log information messages when an authentication attempt fails, instead of warnings.
`DEBUG`	The connector will log debugging messages when an authentication attempt fails, instead of warnings. As *`INFO`* is the recommended logging verbosity level for production systems, choosing *`DEBUG`* here will filter out such messages to avoid an excessive amount of such messages when users frequently enter passwords incorrectly.

Options for `authn.trust.keySpec`

`keySpec`	When trusted authentication is setup as part of an SSO solution, the hexadecimal-encoded key specification (generated either by OpenAGE or by OpenAGE Viewer) is set as the entry value for this option. The API does not permit authentication without a password unless a valid key specification is configured for both ends of the connector.

Note that deleting this file has no immediate effect: it is permitted to delete this file as part of a deployment workflow, pending deployment of an updated version. Deleting the file will only have an effect if the application is restarted without any replacement file being deployed.

system.connector_endpoint.xml

The role of this file is to associate a fixed identifier to a connector, so that application code can avoid hard-coded connector endpoints (hostnames, etc.); without such identifiers, this would be a potential source of configuration errors when deploying code (a developer should not normally directly develop an application using a live production server) or when updating infrastructure (changing hostnames or ports, for example).

This file is not available by default. It is created automatically if a connector is defined using the setup interface (see the section called “Configuring the default connector”), using the API, or using JMX. It can also be created and updated manually, with any changes taking immediate effect (without requiring an application restart). The following example illustrates how this file can be used:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="d55c21e6-9a41-4c19-8bee-11ed1c817390">http://host:8080/openage/</entry>
</properties>
```

Each entry corresponds to an OpenAGE connector; normally, this file will contain at most one entry. Entry keys must be unique UUIDs, and not arbitrarily copied-and-pasted (use tools such as **uuidgen** to create one if necessary), and values must be well-formed connector endpoints, as shown above. When deploying an application for the first time, copy the file and change as appropriate the connector endpoint (to refer to your production OpenAGE server, instead of your development server) without modifying the UUID (this is the only time that the UUID should be copied).

Note that deleting this file has no immediate effect: it is permitted to delete this file as part of a deployment workflow, pending deployment of an updated version. Deleting the file will only have an effect if the application is restarted without any replacement file being deployed.

system.connector_alias.xml

The role of this file is to configure aliases for connector endpoints. It is recommended to create this file on production servers, so that accidental overwriting of a production connector hostname by a developer does not suddenly make a production server try to connector to the developer's OpenAGE instance. It should not be necessary to use this file on a developer system (with a rational deployment workflow).

This file is not available by default. It is created automatically if a connector alias is defined using the API. It can also be created and updated manually, with any changes taking immediate effect (without requiring an application restart). The following example illustrates how this file can be used:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="http://dev:8080/openage/">https://prod:8443/openage/</entry>
</properties>
```

In the above example, if `system.connector_endpoint.xml` is deployed and refers to the endpoint "http://dev:8080/openage/", it will be interpreted instead as "http://prod:8080/openage/", without needing to fix the incorrectly-deployed file.

Note that deleting this file has no immediate effect: it is permitted to delete this file as part of a deployment workflow, pending deployment of an updated version. Deleting the file will only have an effect if the application is restarted without any replacement file being deployed.

system.http.xml

The role of this file is to configure options for the HTTP service implementation.

This file is not available by default; all of the options that it contains have default values. It is created automatically if any default value is overridden using the API or using a JMX console. It can also be created and updated manually, with any changes taking immediate effect (without requiring an application restart). All available options are presented in the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="redirect.code">303</entry>
<entry key="request.debug.enabled">false</entry>
<entry key="session.serializable">true</entry>
</properties>
```

Options for `redirect.code`

`code`	A valid HTTP redirect code, defaulting to 303, with 302 and 307 also accepted. The default value is recommended for modern HTTP clients.

Options for `request.debug.enabled`

`true`	When `DEBUG` logging is enabled, the `http.log` file will contain entries for all requests handled by the server, including the mapped diagnostic context, the remote address, the HTTP method, and the requested URI. When `TRACE` logging is enabled, this behavior is automatic. Enabling this feature may significantly increase the verbosity and size of log files.
`false`	Avoids adding additional entries in `http.log` to identify all incoming requests. This is the default setting.

Options for `request.remote_addr`

`auto`	Enables automatic behavior for detecting and reporting the remote (client) IP address. When selected, the value of the `X-Real-IP` header (or supported equivalents) will be used when present (and valid), instead of the value of the `REMOTE_ADDR` CGI variable, *if* the remote address corresponds to the address of a local loopback interface. This targets common reverse proxy server configurations, and is the default setting.
`accept`	Defines the option to always accept the value of a header-defined remote address, without requiring usage of a loopback interface. This targets configurations where a reverse proxy is used, but not via the local loopback address.
`reject`	Defines the option to always ignore the value of a header-defined remote address, even if present. This aims to prevent IP address spoofing, for when localhost is potentially untrusted.

Options for `session.serializable`

`true`	Requires that session attributes implement `java.io.Serializable`. Serialization is required to enable scalable deployments, for persistent and distributed sessions. This is the default setting.
`false`	Enables storage of arbitrary Java objects in the HTTP session, but at the cost of limited scalability and limited durability (if the bundle is stopped, active sessions will be discarded in this mode).

Note that deleting this file has no immediate effect: it is permitted to delete this file as part of a deployment workflow, pending deployment of an updated version. Deleting the file will only have an effect if the application is restarted without any replacement file being deployed.

Index