Amalfi 2.1

User Manual

GAEL-AMALFI-SUM-001-03-04

December 2011

Document Change Log

ISSUE	DATE	DESCRIPTION
2.0-draft	2011-05-31	This new version is inherited from the version 1.7 of Amalfi 2.0 Software Users Manual (ref. GAEL-P236-SUM-001).
2.1	2011-06-16	- Deletion of section 1.3 : it was empty - Section 2.1 : update of versions of Amalfi packages - Section 2.4 : update of installation section with new screenshots and improvements like update an existing installation - Section 4.6 : specifying that if you submit an item without answering server, a local queue is launch. - Adding section 4.7 - Section 6.1.4.1.3 : adding TLS and emailReportName keys to E-mail broadcaster service - Section 6.1.4.1.5 : adding fileReportName key to File broadcaster service - Adding Appendix A — Best Practices - Updating Appendix B — Example of XML configuration file - Updating Appendix C — XML-Schema describing 2malfi configuration file (normative)
3.0	2011-07-22	Document fully rewritten and reorganized. Adding Report normative XML schema in appendix.
3.1	2011-07-22	- Adding Server "General behaviour" p34. - Added sample command line focusing standalone sublmission p40 - Enhance description of the "`—addon`" sample p41. - Reference the built-in inspectors in the inspection creation use case in chapter "Use Case: Creating a new add-on for Amalfi." P80 - Add a reference to the IPFJobSubmitter Interface control document in the Amalfi Server section.
3.2	2011-09-23	- Added server synopsis and its description

		- Added Amalfi database prerequisites - Added expected java version - Added expected Mysql version. - Added alternative user defined configuration description
3.3	2011-10-11	Change the document identifier from GAEL-P264-SUM-001 to GAEL-AMALFI-SUM-001
3.4	2011-12-01	- Adding Encrypter function to encrypt password with DES inside your configuration. - Changing rpm target to /usr/local/amalfi/amalfi-xx.xx.xx

Table of Content

Welcome to Amalfi

Amalfi application performs controls inside earth observation data to produce quality reports. It is composed of a set of highly configurable applications that allow to be deployed over a network or in standalone mode. Handling a new earth observation data is fully configurable. This manual aims to exhaustively describes, the way of doing these configurations.

About this Manual

This book is focused on the description of all features of the Amalfi software components and their installation. In addition to the descriptions of software functionalities, this book also introduces general concepts that guided the design of the Amalfi software, and that may enhance the understanding of the features breakdown and boundaries. Finally, this book addresses advanced use cases that intend to support the understanding of, where the software should be deployed, how it could be tuned for security and performance issues, and how it could be configured to support more inspections and input data types.

This book is dedicated to Data Experts, Data production Operators or any users interested in data Quality Control, either in routine operations or for occasional sessions.

This book also provides System Designers and Administrators with information about how Amalfi can be deployed, fine tuned and secured over their networks and operating environments.

Finally, this book may also serve Project Managers in their decision processes, in particular, to determine whether Amalfi features are of interest to achieve their project goals, and assess the effort of integration and training Amalfi would require.

It is expected that most readers will have some familiarity with computerized applications and with the Operating System and Desktop Manager they have selected. Basic knowledge about networks is also required for those readers that intend to deploy Amalfi over multiple computers. Should advanced topics be required, it is highly recommended to have a standard knowledge of W3C technologies including XML, XML Schema, XQuery, RDF and OWL.

This book is organized in a bottom-up fashion, starting from general concepts and techniques that governed the design of Amalfi software, turning then to the software component installations, their feature descriptions, and ending up with a series of advanced topics that provide readers with tricks and tips for fine tuning and troubleshooting the installed software.

Chapter 1 – Introducing Amalfi	Introduction to Amalfi general concepts and techniques
Chapter 2 – Installing Amalfi	Installation guidelines
Chapter 3 - Amalfi Compass	Amalfi Compass user manual
Chapter 4 – Amalfi Server	Amalfi Server user manual
Chapter 5 – Amalfi Database	Amalfi Database user manual
Chapter 6 – Amalfi Configuration	Complete description of Amalfi configuration

Chapter 7 – Amalfi add-ons	Describes add-ons and how to create one
Chapter 8 – Deploying Amalfi	Show a sample of a network deployment of Amalfi
Appendix – 1	Understanding background concepts of Amalfi
Appendix – 2	System Requirements
Appendix – 3	Sample configuration file
Appendix – 4	Configuration file XML schema (normative).
Appendix – 5	Amalfi Report XML schema (normative)
Appendix – 6	License Terms

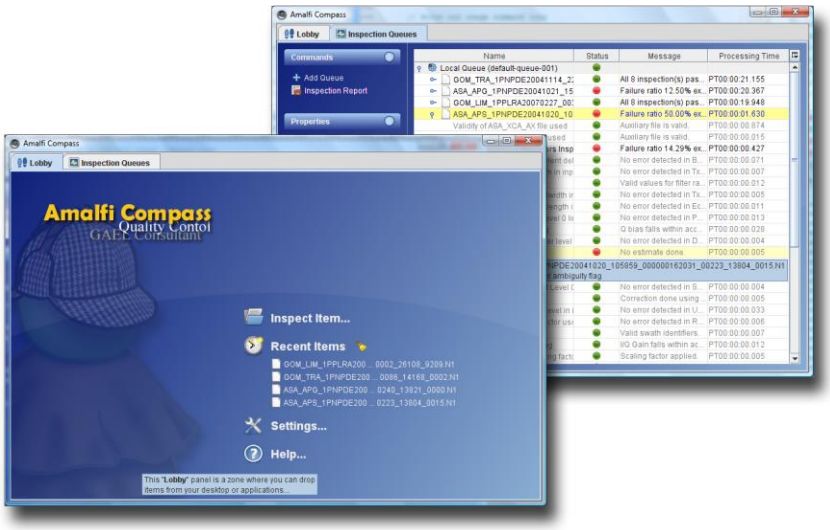
Additional Resources

http://www.gael.fr/amalfi

Introducing Amalfi

About Amalfi

Once Amalfi installed on your computer, you are equipped with an automated inspection machine. Either in a few clicks, if you are using a graphical environment, or a single command line from a terminal, you are ready to inspect your data and setup a Quality Control report: Amalfi software have been specially designed to carry out intricate things from simple interfaces.



If you are running Amalfi deployed over a network, you are able to monitor the Quality Control activity about the production of all your sites simultaneously, trigger inspections remotely, extract statistical figures and status accounting about the quality of your production, get reports about inspected data, still in a few clicks or command lines.

For readers interested in a more practical top-down approach, we recommend to skip the following conceptual and theoretical sections of the present chapter and jump to Chapter 2 - Installing Amalfi.

Understanding the Basics

Amalfi application defines a set of new concepts that defines a specific Amalfi vocabulary. Before starting Amalfi description, it is required to well understand this vocabulary:

- *Item*: It is the earth observation product once decoded by Amalfi low level layer (called DRB API®). Note that one product can be constituted by multiple items.
- *Inspector*: It is the process that performs the inspection. This process is Java based class.
- *Inspection*: It is the unitary configurable (written in OWL language) part of the processing to perform data control.
- *Inspection plan*: Is a set of inspection gathered in a "plan". The plan is also defined in OWL/RDF language.
- *Inspection Queue*: Is the processing queue where will be executed inspections. This queue is configurable to manages threading and priorities of inspections executions.

- *Report*: Is the output of the inspections processing.
- *Topic*: Is the set of files that configures a new data support (usually a jar file or a directory that contains OWL descriptions).
- *Add-on*: Is the set of files that configures a new data support and its inspections. An "addon" is a topic plus the Amalfi inspections.
- *Configuration*: is the file used to configure Amalfi application. It configures the processing inspection queues, the add-on to be loaded, the services to be executed by the application.
- *Amalfi services*: Are modules integrated into Amalfi application that can be activated to broadcast reports in various outputs (databases, files), or to configure a file scanner… The services are attached to an Amalfi inspection queue.

Amalfi at a Glance

Amalfi Compass

Compass is the graphical user interface able to perform and monitor inspections into a local inspection queue or a remote server queue. It implements a set of interfaces to display report, display and browse items and access database if configured. Its setting panel gives access to the configuration of the entire Amalfi environment.

Amalfi Server

Amalfi server is a batch application dedicated to remotely execute inspections over a network using RMI technology. It also proposes a batch Client application to communicate with it. Amalfi Compass is also able to remotely monitor servers and submit items to be inspected.

Amalfi Database

When the database service is configured in the processing queue, the database (MySQL based) gathers all the Amalfi reports. The Amalfi database implements graphical user interfaces able to browse the inspections and to perform statistics other reports stored in this database.

Amalfi Add-ons

The Amalfi add-ons is a directory where are stored all the add-ons that supports items. The add-ons must be configured in the inspection queue to be taken into account, otherwise the item might not be recognized, and no inspection will be found for the supported add-on.

Encrypter

The Encrypter is a tool done to encrypt your passwords to put them not clearly inside your `amalfi-configuration.xml`. Encrypter use Data Encryption Standard, which is a block cipher that uses shared secret encryption.

Installing Amalfi

This section will help you installing Amalfi software on your systems. Thanks to its pure Java nature, Amalfi should run on most of traditional systems but we strongly recommend checking the appendix System Requirements (p. 103) at the end of this manual. According to your system characteristics Amalfi comes up with a generic installer driven by an interactive Graphical User Interface and with an RPM set that may be more suitable if you are running under LINUX RedHat or compatible environments.

This section covers the following:

Generic Graphical Installation Wizard

All software components of Amalfi can be installed through a single installer. This installer is named:

amalfi-suite-<version>-installer.jar

From a Desktop Manager as Microsoft Windows or Linux/Gnome, run the installer by double-clicking on the installer's icon: automatic launch of the installer requires the Java™ software already installed and configured. If the Desktop Manager does not launch the installer automatically but proposes a list of software to associate to the file, select Java™, if available. Otherwise, open a command line terminal and follow the next paragraph.

From the command line e.g. from MS-DOS or a UNIX terminal, run the following:

```
java -jar amalfi-suite-<version>-installer.jar
```

An installer Wizard pops-up and guides you up to the complete installation of Amalfi. The content and features of the Wizard pages are described hereafter, although most users could afford all steps without further information.

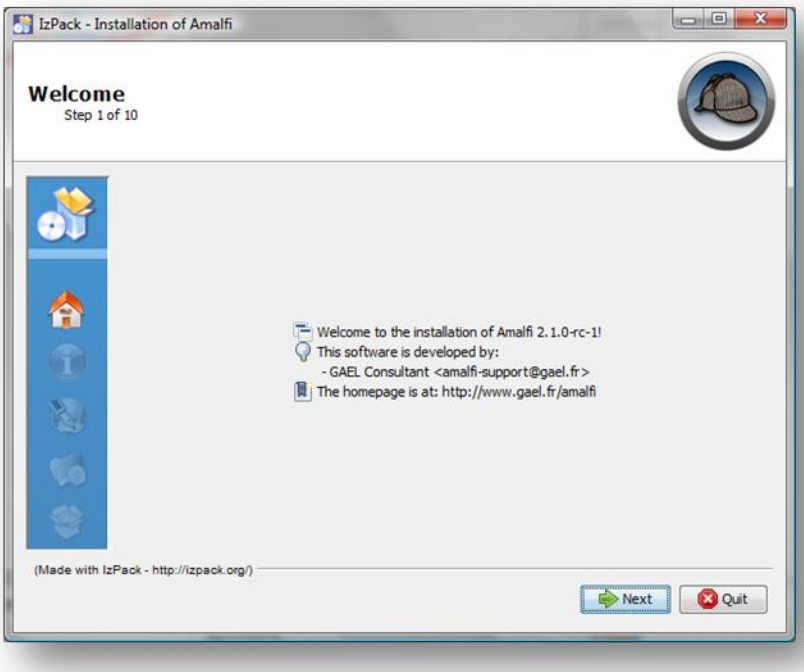


fig. 1 - Amalfi Installer - Welcome panel

The *Welcome Panel* reminds general information about the software due to be installed. It is strongly recommended to check the version of the software before proceeding.

fig. 2 - Amalfi Installer – Information panel

The *Information Panel* reminds more detailed information about the software due to be installed.

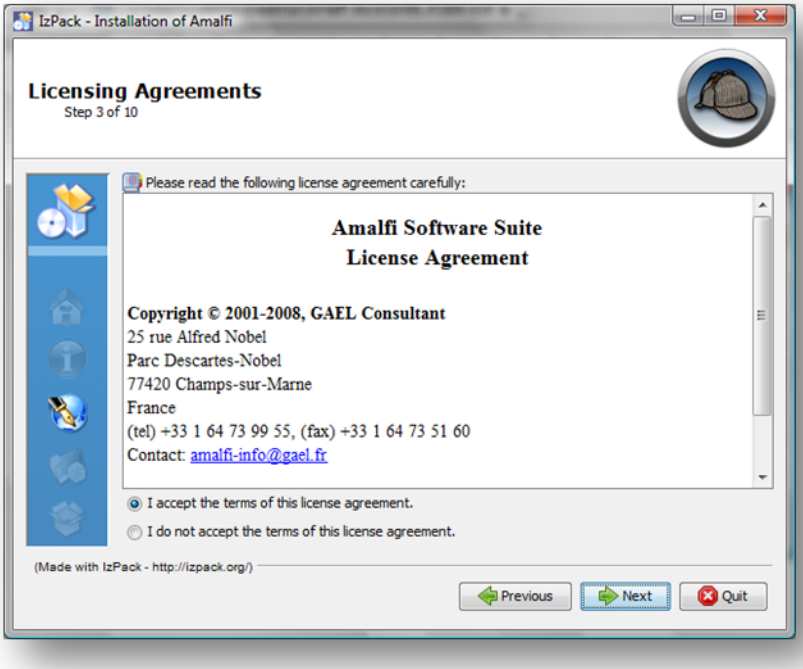


fig. 3 - Amalfi Installer - License panel

Please read carefully the license agreement of the *License Panel*. You shall agree with all the terms and conditions of use before proceeding. Otherwise, you should stop the installation with the *Quit* button.

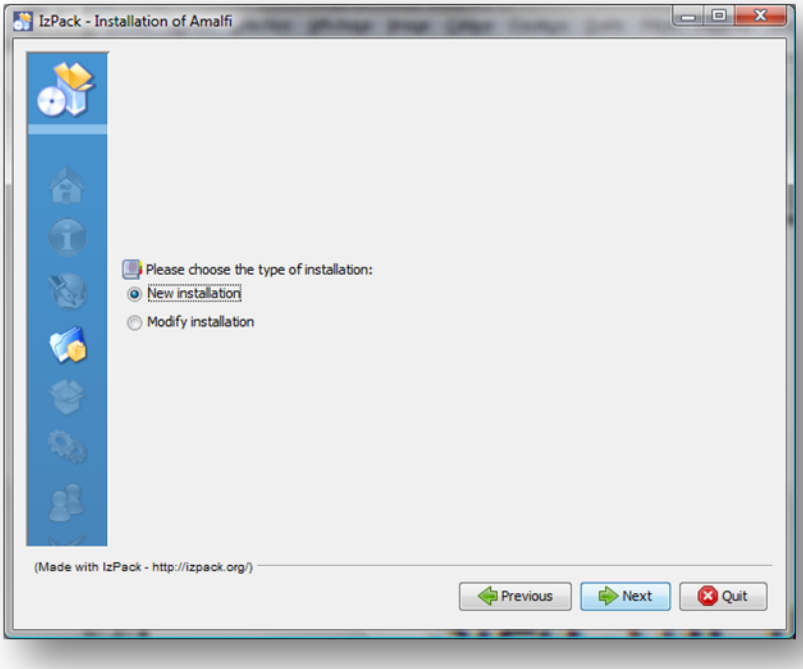


fig. 4 - Amalfi Installer – Type of installation panel

The *Type of Installation Panel* lets you choose between making a new installation of Amalfi Software and updating an existing one.

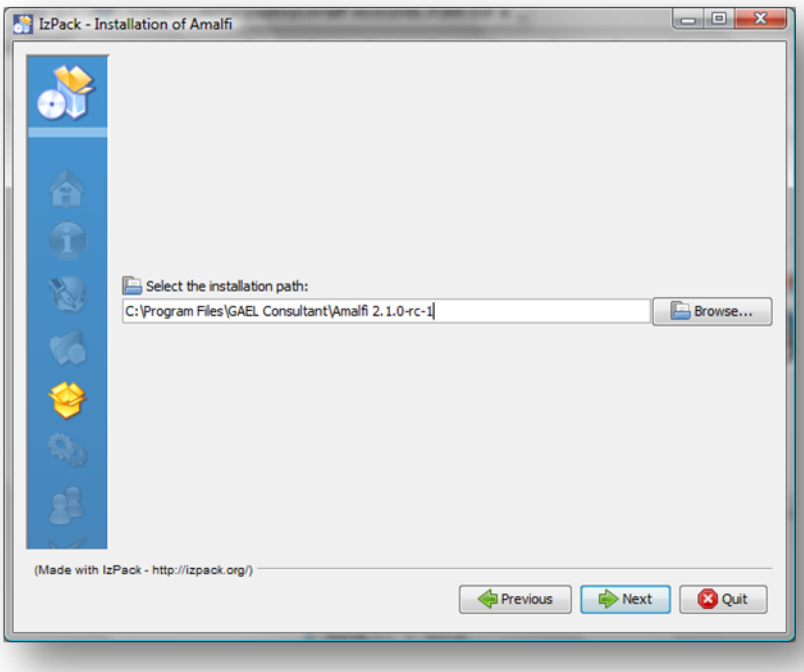


fig. 5 - Amalfi Installer - Target path panel

The *Target Path* Panel allows you specifying where the Amalfi software should be installed: the *Target Path* automatically proposes a location according to the standard rules and policy of the Operating System.

Important: On Windows Vista Operating System, the default location proposed in this panel may lead to access failures at runtime, this although the Wizard terminates without warning. This drawback comes from a well known problem with some Java™ versions that have not already been aligned to Vista security policy. A work around recommended for any installation on Windows Vista is to select a location under the installing user directory: usually the `C:\Users\xxx\` directory.

If you are installing a new instance of Amalfi, if the selected location does not already exist, a dialog window will prompt you to confirm that this location has to be created.

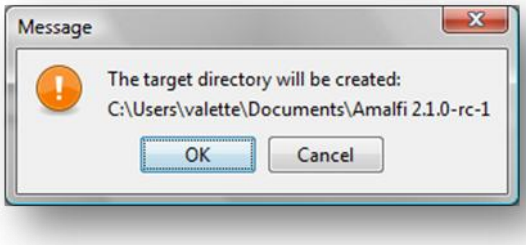


fig. 6 - Amalfi Installer - Target path creation dialog

On the contrary, if the target location already exists, a different dialog box will prompt you to confirm that Amalfi software has to be installed there.

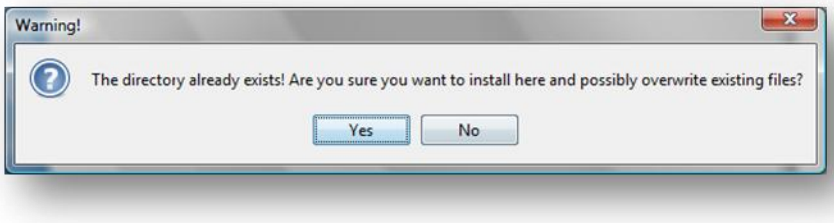


fig. 7 - Amalfi Installer - Target path override warning dialog

Overriding a directory may lead to a loss of data, and in particular, overriding a previous Amalfi installation (without using update function) of a different version may create hazardous behaviours during runtime. Conversely, cautiously overriding an Amalfi installation of the same version can restore corrupted files, or augment the installation with new components that where not installed (all modified files would however be restored to the default values).

In some rare cases where a target location may be missing, have been simultaneously deleted and cannot not be created, the installer may raise the following dialog will prompt you to reselect or rephrase the target location.

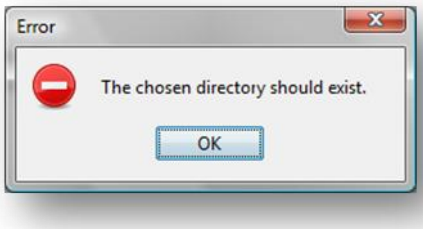


fig. 8 - Amalfi Installer – Not existing Amalfi installation

As introduced earlier, the Amalfi software is broken down into a set of components that have different purposes. Those components have been grouped in installation packs that can be enabled or disabled in the *Selection Panel* of the installation Wizard.

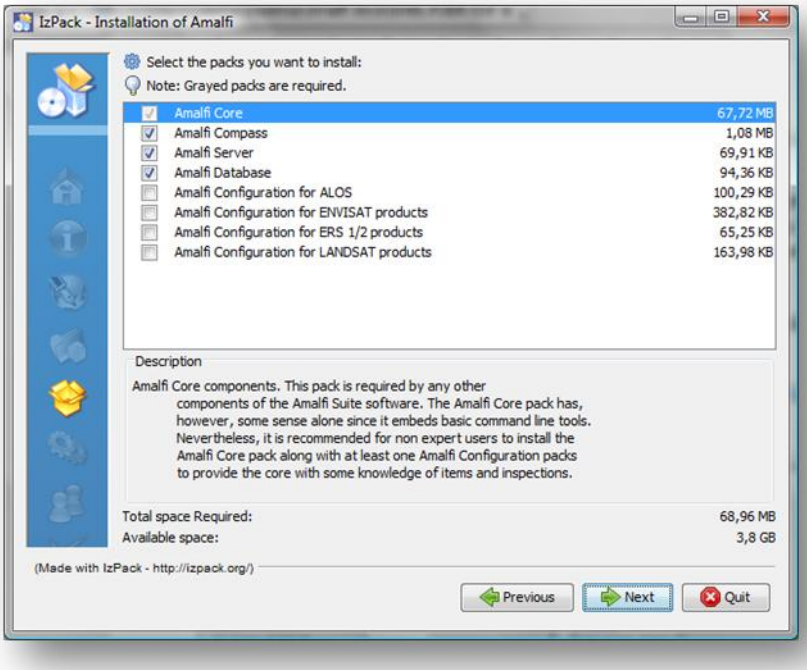


fig. 9 - Amalfi Installer - Installation pack Selection Panel

In case of installation update, the *Selection Panel* automatically pre-selects the already installed packs and you should unselect those that you may want to remove.

Whether for fresh installations, updates and during your editing, the panel show up with pre-selected packs and keeps track of the consistency of the selections according to the actual software dependencies that may exist between the packs.

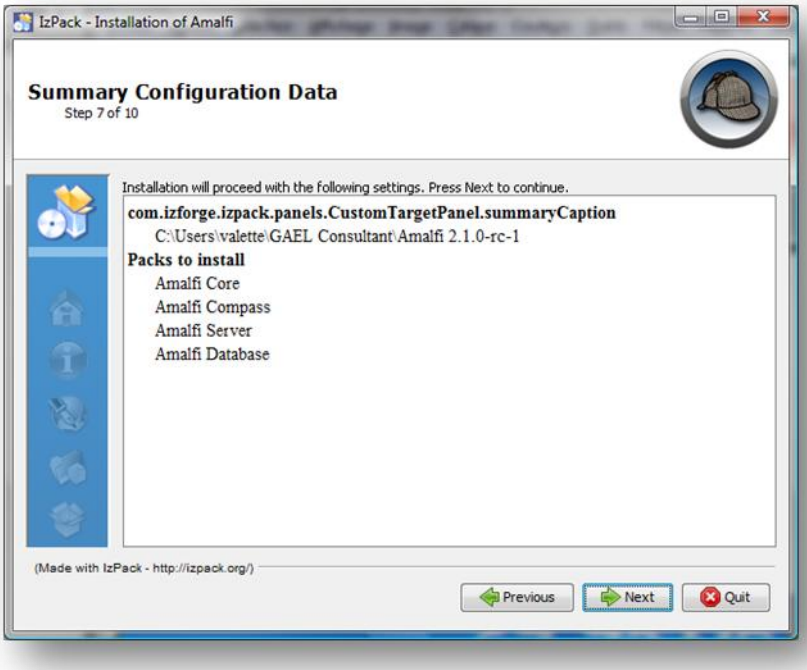


fig. 10 - Amalfi Installer - Installation pack Confirmation Panel

Accepting the information reported in the *Summary Panel* will proceed to the installation: file copy activity is reported according to each pack being installed and according to the global installation progress.

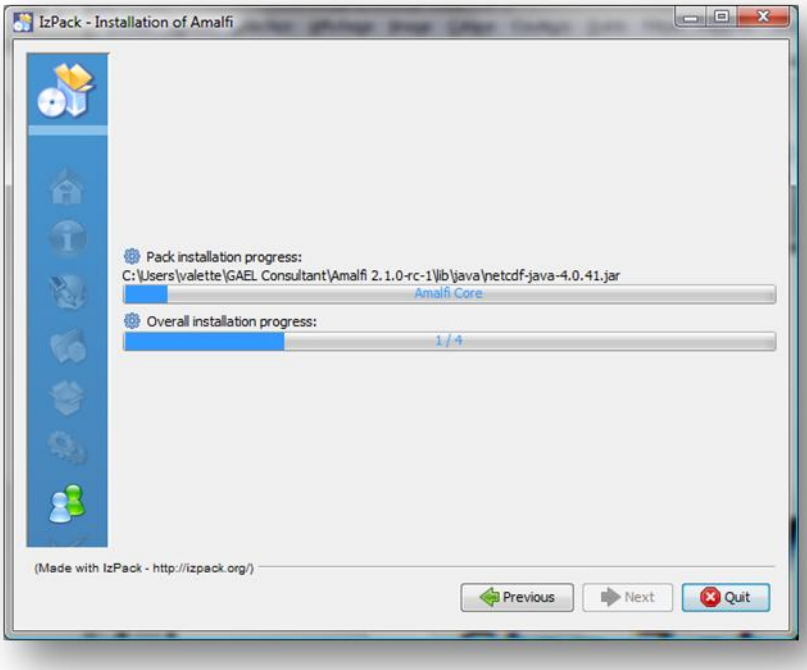


fig. 11 - Amalfi Installer - Installation progress panel

During an update, an already installed pack will be replaced if the installer can provide a newer version or can repair a corrupted file. Newer files are identified according to their version numbers

and corrupted files according to the file sizes. If the installed file sizes differ from those known by the installer they are to be replaced.

During installation, if the configuration file `etc/amalfi-configuration.xml` exists, a dialog box will prompt you to confirm that you want to overwrite it. You should answer with caution if you have already modified this configuration file that may include important resources that could be discarded.

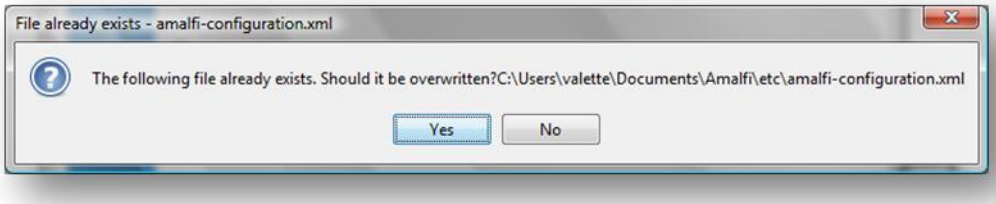


fig. 12 - Amalfi Installer – Configuration file overwrite dialog

Once all files installed, the Wizard proposes to create shortcuts that helps launching main Amalfi components as the Amalfi Compass, the online documentation and uninstaller. The shortcuts can be placed on the Desktop and/or in the main pull-down menu of the interface e.g. the Start menu on Windows environments. Those shortcuts can be installed in the scope of the current user or for all users configured on the Operating System. This latter option is generally applicable if you have system administrator's rights or can be granted to.

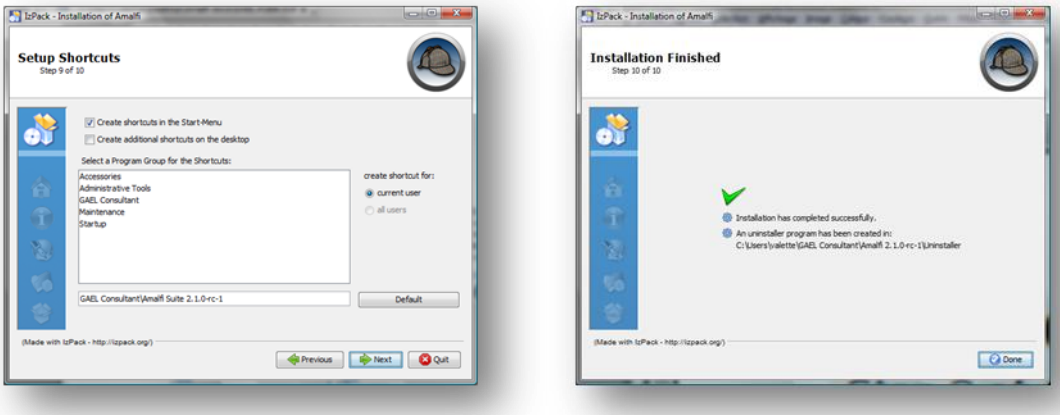


fig. 13 - Amalfi Installer - Desktop shortcut setup panel

Important: When installing the Amalfi Database module, it is required to have an already installed MySQL database. MySQL database can be downloaded from: http://www.mysql.com. See appendix System Requirements (p. 103) for more information about compatible versions with your system.

At this step, selected packs have been installed and the Amalfi software is ready to use!

Linux RedHat RPM Installation

All software components of Amalfi may alternately be installed through specific RedHat Linux installation packages. They are delivered in RPM (RedHat Package Manager) form that process installation of Amalfi for RedHat Linux x86 and compatible systems. The RPM packages are the following:

- **amalfi-softwares-<version>.rpm:** that contains the entire Amalfi software suite.
- **amalfi-addons-<version>.rpm:** suite that contains a set of add-ons, including ENVISAT, ERS, ALOS and LANDSAT supports.

Dependencies

As many other RPM packages those two RPMs depend on third party packages that are usually installed by default on RedHat systems or available from RedHat RPM distribution libraries. The two main dependencies are the following:

- **Java JRE v1.6+:** http://java.com/en/download/manual.jsp
- **Mysql server v5.0+:** http://dev.mysql.com/doc/refman/5.0/en/linux-installation-rpm.html

Installation Instructions

To install RPM packages, it is first required to install the Java JRE, following vendor instructions as reported at http://java.com/en/download/help/linux_x64rpm_install.xml, and MySQL software as detailed in http://dev.mysql.com/doc/refman/5.0/en/linux-installation-rpm.html.

The installation of Amalfi RPM packages can be performed by running the following instructions from the prompt of typical command shells:

```
rpm –i amalfi-softwares-XX.XX.XX.rpm
rpm –i amalfi-addons-XX.XX.XX.rpm
```

You are now done and Amalfi is now ready to use on your system!

Installation directories

RPM packages automatically install the Amalfi software into `/usr/local/amalfi/amalfi-XX.XX.XX` directory. The sub-directories are the following:

- **etc:** Contains the main `amalfi-configuration.xml` configuration file described in section Amalfi Configuration (p. 46);
- **docs:** Contains the online documentation and a version of this User Manual;
- **lib/java:** Contains the set of java libraries to run the Amalfi software;
- **examples:** Contain a set of useful samples and helpers to use and configure Amalfi;
- **resources:** Contains some non-Java the resources necessary to run Amalfi software;
- **addons:** Contains some add-ons extending the dataset and inspections recognized and applicable by the installed Amalfi instance. This directory is not mandatory as far as the add-ons it contains are not fully referenced by the `etc/amalfi-configration.xml` configuration file as explained in the section Amalfi Configuration (p. 46).

ESA Sentinel-1 Satellite Specific Installation

An additional RPM package is specifically made available for the European Space Agency (ESA) Sentinel-1 satellite to further configure Amalfi to the design of this satellite Payload Data Ground Segment (PDGS):

- **amalfi-addon-sentinel-1-<version>.rpm:** That contains the ESA Sentinel-1 product definition and base inspection sets.

It can be installed further those described in the previous sections with similar commands:

```
rpm –i amalfi-addon-sentinel-1-XX.XX.XX.rpm
```

You are now done and Amalfi is now ready to inspect Sentinel-1 products on your system!

Amalfi Compass

The Amalfi Compass is the Graphical User Interface of Amalfi. This component allows submitting items for inspection, to either a local or a remote inspection queue. The Amalfi Compass also provides features for monitoring one or more inspection queues, browsing item contents and viewing inspection reports. If you were used to one of the Amalfi's predecessors, the Amalfi Compass supersedes the former QUISS, QCC or AQC tools.

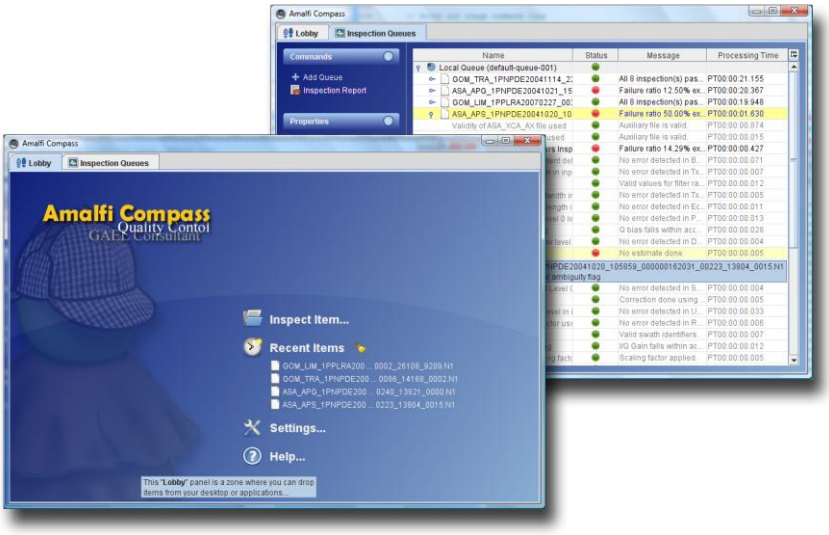


fig. 14 - Amalfi Compass

The Amalfi Compass is mainly dedicated to the data processing operators, product experts or other users that wishes to occasionally assess the quality of digital data.

Running Amalfi Compass

If you have installed the application shortcuts during the installation procedure, you can run the Amalfi Compass by activating the corresponding icons on the Desktop or in the main menu e.g. the Start menu under Windows environment.

To run Amalfi Compass from the command line e.g. from MS-DOS or a UNIX terminal, run the following:

```
java -Xmx1024m -jar <intall-dir>/lib/java/amalfi-compass-<version>.jar
```

where `<install-dir>` is the location where Amalfi software has been installed, and `<version>` depends on the installed version of the software.

Graphical User Interface

The interface is minimalist and shows icons, buttons and options only when they are necessary. You should notice that the application runs without menu bar and classifies the functionalities through a series of panels organized in a single tabbed pane.

The graphical user interface is composed of a single window containing two or more tabs. The main tab, called lobby view provides the main commands necessary to perform inspections. The second tab graphically monitors the connected inspections queues. Other possible tabs may be opened during the application use, in order to show and browse an item, or to display a report. The following chapter details these tabs.

Lobby Panel

The main interface displayed at start of Amalfi Compass is the so called *Lobby Panel*. This area has been divested of all secondary options to be focused on the essentials: launching an inspection, configuring the software or learning more about it.

fig. 15 - Amalfi Compass - Lobby panel

The following sections describe the four main functions made available from the *Lobby Panel*.

Inspect Item

First function of the Lobby panel is dedicated to the selection of an item. Clicking this link open a file chooser to select the item to be inspected.

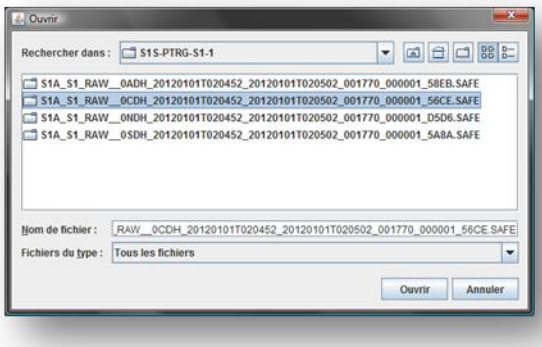


fig. 16 - Amalfi Compass – Inspect Item file chooser

It is always possible to load a new item with dragging it from the desktop (Windows or Linux/gnome tested), and dropping it anywhere in the Compass application frame.

Recent Items

"Recent items" is a list of the items that has been previously opened. This list allow user to start an inspection already performed.

Settings

Clicking setting section open the settings panel detailed at p.30.

Help

Clicking the help section opens a new tab to display the online help of the application, detailed at p.31

Inspection Queues Panel

The "Inspection Queues" panel deals with monitoring and controlling of the inspection activities performed by one or more inspection queues. The inspection queues can be either local queues (attached to the current Amalfi Compass process) or remote queues exposed by Amalfi Servers.

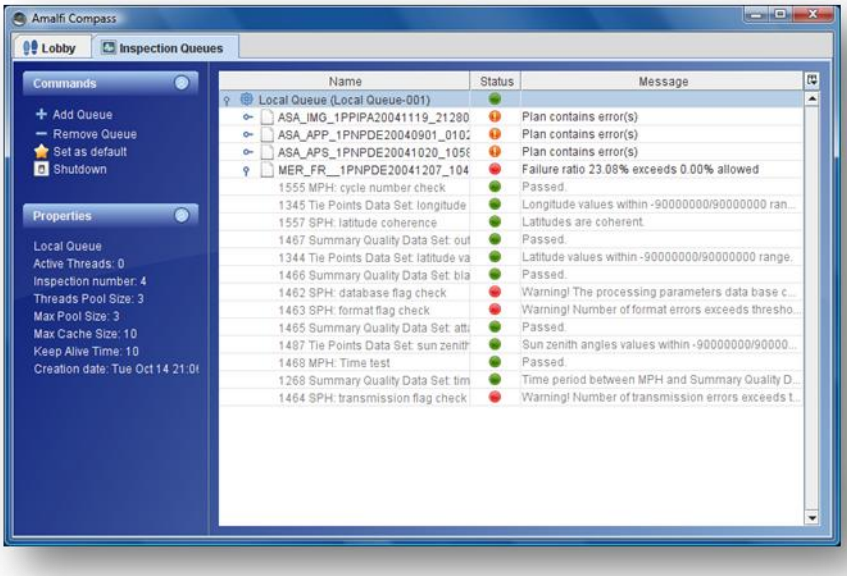


fig. 17 - Amalfi Compass - Inspection queue panel

For convenience, it is possible to show or hide columns of the table from the pull down menu that pops up when the button at the top right corner of the table, as shown in the figure below.

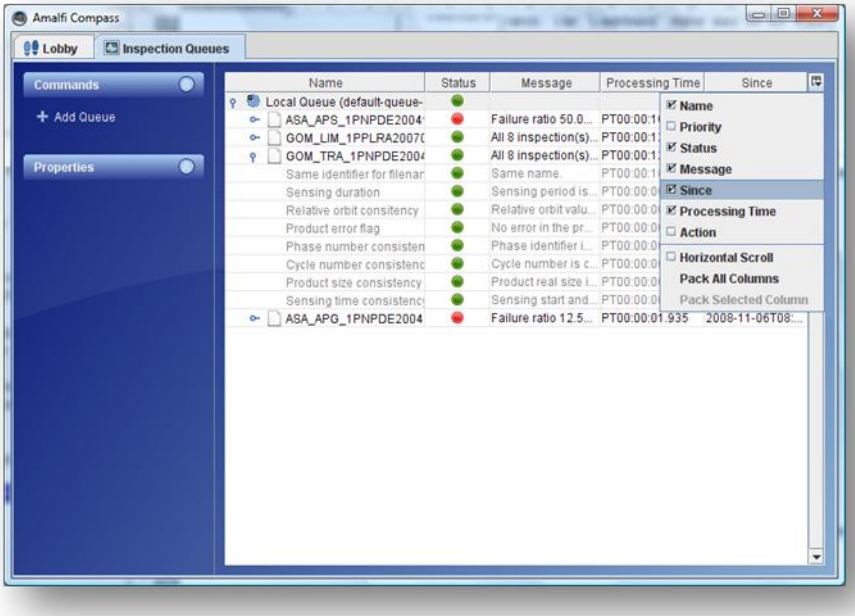


fig. 18 - Amalfi Compass - Enabling more columns

This menu also allows re-computing column sizes according to predefined rulings, and enabling horizontal scrolling.

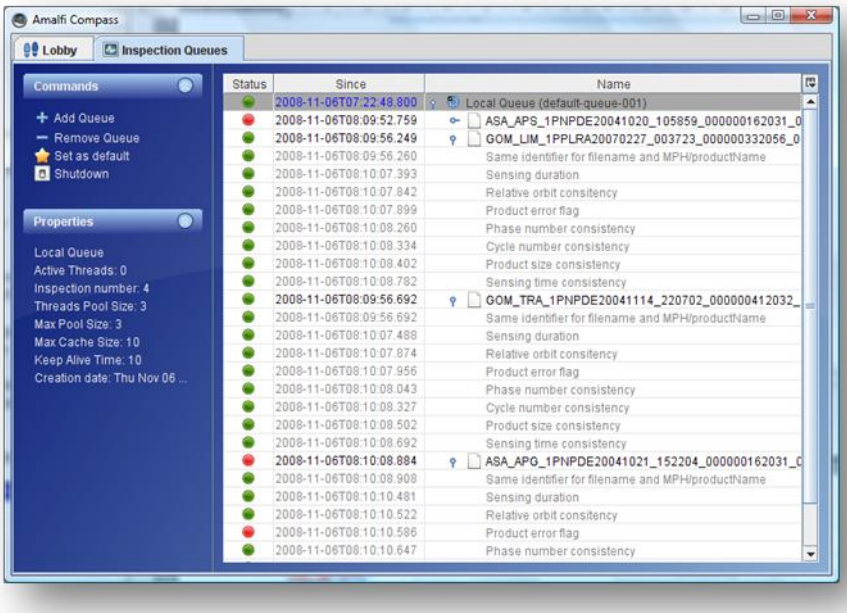


fig. 19 - Amalfi Compass - Reordering/resizing columns

As shown in the figure above, the columns can re-ordered or resized. Re-ordering is performed by dragging the columns header from left to right, and the resizing by dragging the vertical separators between the columns.

Tabled tree columns

The main graphical element of the panel is a tabled tree that lists the inspection queues and their attached inspections as child nodes. If inspections are plans with sub-inspections, they are also reported as child nodes of their parent inspection.

The meaning of each column of the table is summarized in the table below:

Column Name	Summary
Name	The name has different values according to the row types: – for inspection queues, the name of this inspection queue – for simple inspections, the name of the inspection – for inspection plans, the name of the inspected item followed by the name of the plan between parenthesis Note: the special case of inspection plans has been selected to visualize in a single look the items subject of inspection; otherwise, the table would have been filled with inspection names only missing the target items, or, on the contrary filled with a repetition of the same item names;
Priority	The priority level of the inspection or nothing for inspection queues.
Status	The status has different values according to the row types: – for inspection queues, denotes whether the queue is active or is shutdown – for inspections, the status of the inspection as specified in the "table 2 - Inspection processing status" below
Message	Inspection result messages or error message for inspection queues
Since	The start date of the inspection queue, or the one of the inspection
Processing Time	The processing time has different values according to the row types: – for inspection queues, the time elapsed since the queue has been started – for inspections, the execution duration of the inspection
Action	Reserved for future use

table 1 - Inspection queue tabled tree columns

The Status column is computed from the inspection processing status and the inspection result. The special meaning of the icons used as values of this column is summarized in the following table:

Icon	Status	Summary
	Pending	Once an inspection created, its processing status is set to a "Pending" status. This denotes that the inspection has been successfully initialized and wrapped around the corresponding item(s) and that it is ready to be run. At this stage the inspection has an initialized creation date but no inspector result and consequently no corresponding report.
	Running	This processing status denotes that the inspection is currently running: it always follows a "Pending" status. At this stage, the inspection has an initialized creation and a running dates.
	Cancelled	This processing status denotes that the inspection has been cancelled.
	Error	This processing status denotes that an error occurred during the inspection: an error does not mean that the item failed the inspection. See Message column for textual information about the origin of the error.
	Passed	This processing status denotes that the processing has been successfully performed and that the item passed all inspections.
	Failed	This processing status denotes that the processing has been successfully performed and that the item failed the inspections. Refer to the Message column for further information about the failure.
	Undefined	The "Undefined" denotes that processing status could not be determined at this stage of the processing. This case should never occur or very rarely. The "Undefined" status may be set when in between two transitional states or because communication with a remote processing server has been lost.

table 2 - Inspection processing status

Commands task pane

The task pane at the top left of the panel, provides contextual actions depending on the tabled tree selection.

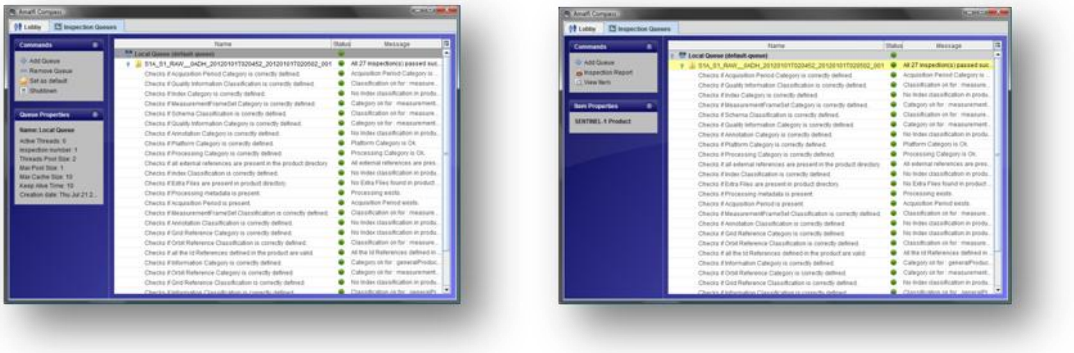


fig. 20 - Amalfi Compass - Contextual task panels

The contextual actions are summarized in the table below, and illustrated by the figures of fig. 20 - above:

Action	Context	Summary
Add Queue	Always	Add a new inspection queue in the tabled tree. See below for further information about how this action behaves.
Inspection Report	Inspection	Opens a new *Report Panel* about the select inspection.
View Item	Inspection	Opens a new item panel allowing display and browsing inside the selected item.
Remove Queue	Queue	Removes the selected inspection queue after a confirmation dialog.
Set as default	Queue	Mark the selected inspection queue as default inspection queue. The selected inspection queue icon shall change accordingly and all future inspection submissions will be sent to this inspection queue.
Shutdown	Queue	Shutdown the selected inspection queue.

table 3 - Amalfi Compass - Command task panel actions

The "Add Queue" action opens an "Add inspection queue..." dialog box as the one below, that needs some explanations.

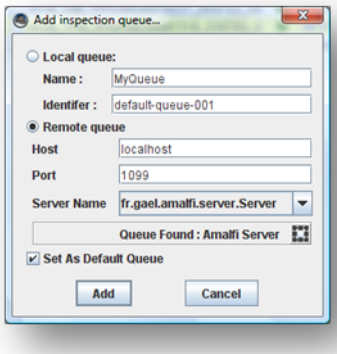


fig. 21 - Amalfi Compass - Add inspection queue dialog

The dialog allows the creation of a new local inspection queue (running in the same process as the Amalfi Compass) or creating a connection to a remote inspection queue: your selection of one of these solutions is governed by the two radio buttons "Local queue" and "Remote queue".

The creation of a local queue requires a free textual name of the queue to be created and a compulsory identifier. The choice of the identifier is important since it will be matched to the resource entries of the current configuration. Please, refer to the section "*Amalfi Configuration*" below for further information about the configuration of Amalfi.

The connection to a remote server is a little bit more complicated. It requires the target hostname, the network port number to be used: these parameters are initialized for looking up to the local host with the default port used of an Amalfi Server. Any change is kept between multiple sessions of the Amalfi Compass. Once those parameter set, they automatically looked up and, if any server could be found, the names of the available inspection queues are proposed in a pull-down menu (in the illustration above, the remote queue is named `fr.gael.amalfi.server.Server`, the default name of inspection queues exposed by an Amalfi Server).

Finally, you may select if the new inspection should become the default one and press the "Add" button.

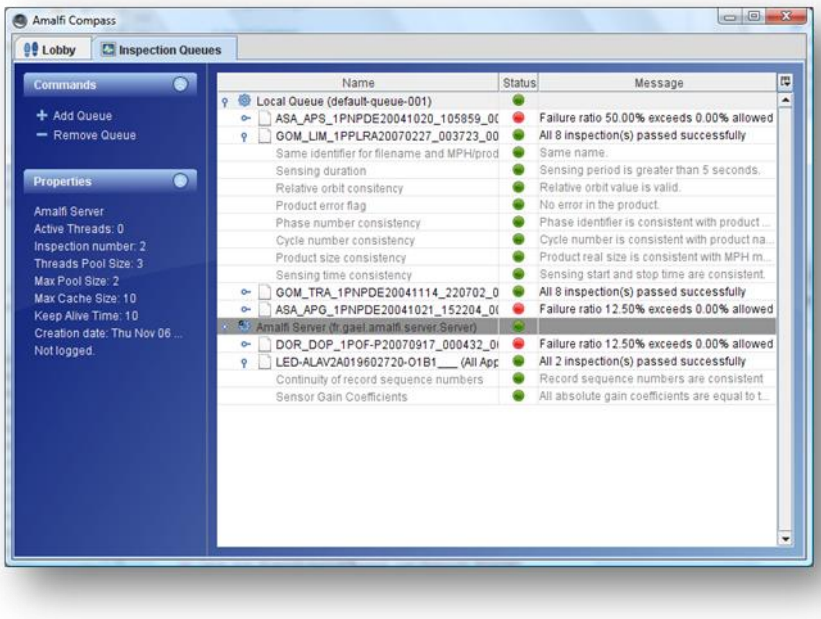


fig. 22 - Amalfi Compass - Multiple inspection queues

The result of this action is a new entry in the tabled tree that corresponds to the newly created inspection queue or to the connected remote inspection queue as shown in the fig. 22 - above. The number of inspection queues that can be monitored is unlimited.

Report panel

The report panel displays the pdf report of an inspection.

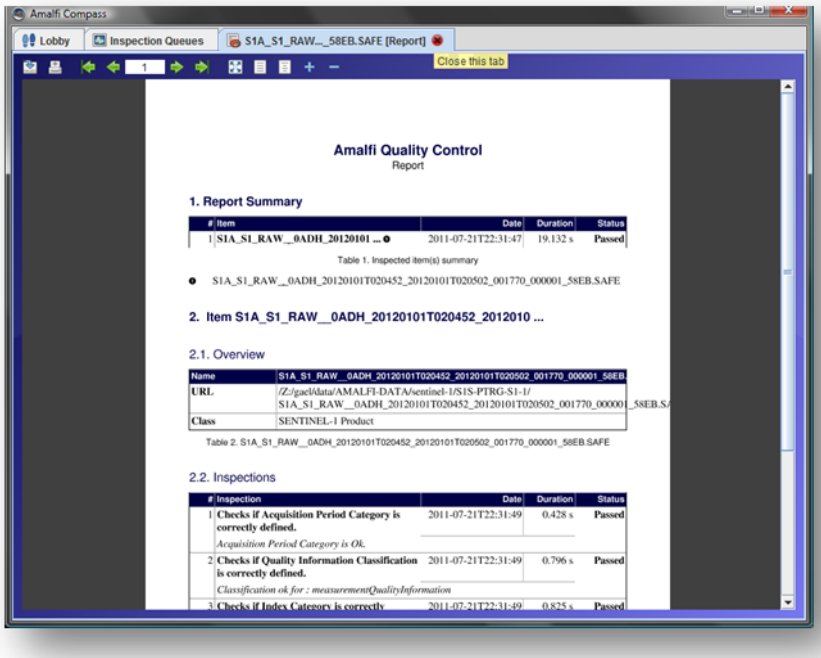


fig. 23 - Amalfi Compass – report panel

With the report panel, you can browse, print or save the inspection report in pdf format.

Item View panel

The item view panel gives access into the loaded item displaying a summarized description of the item on top, and an interactive tree on bottom. This tree is a view of the product that has been inspected.

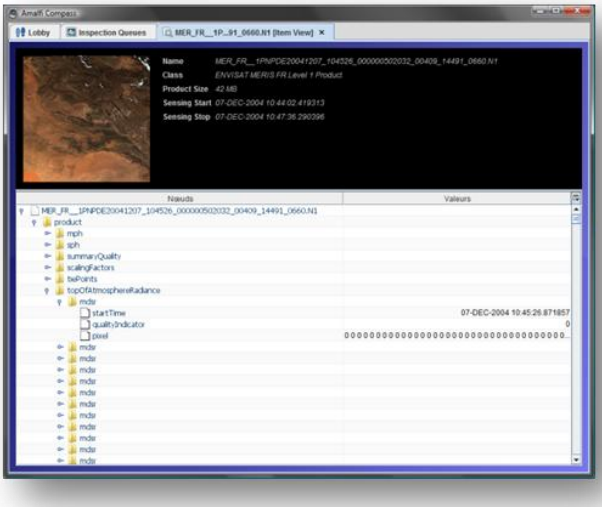


fig. 24 - Amalfi Compass – view item panel

Settings Panel

The settings panes allow users to configure the Amalfi Suite components such as "Users", "Groups" or "Resources". A complete description of the configuration file is available in next Administrating chapter. The Setting panes provide users with a user friendly manner to edit loaded Amalfi configuration. When modifications are applied, it will update or create the customized configuration stored in `${user.home}/.amalfi/etc/amafi-configuration.xml` (p. 46).

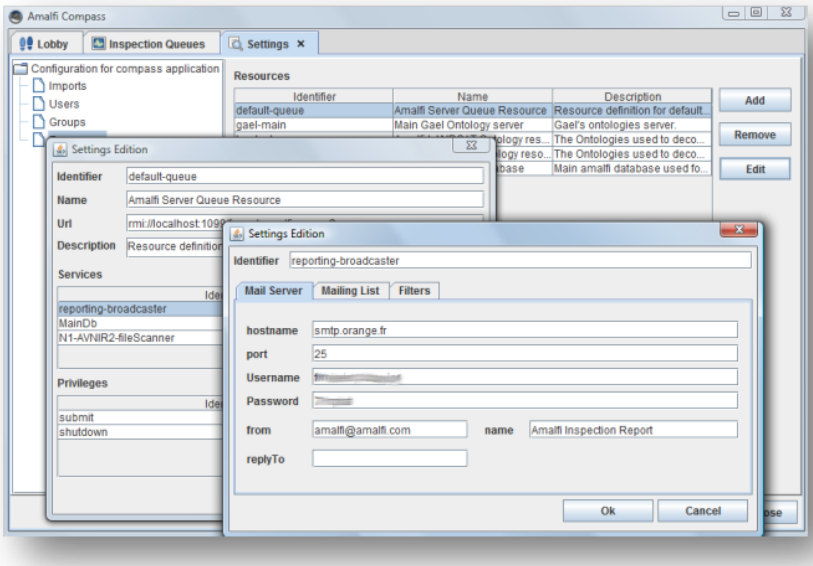


fig. 25 - Amalfi Compass - Settings panel

Online Help Panel

Amalfi compass is able to display this document via "Help" hyperlink of lobby view.

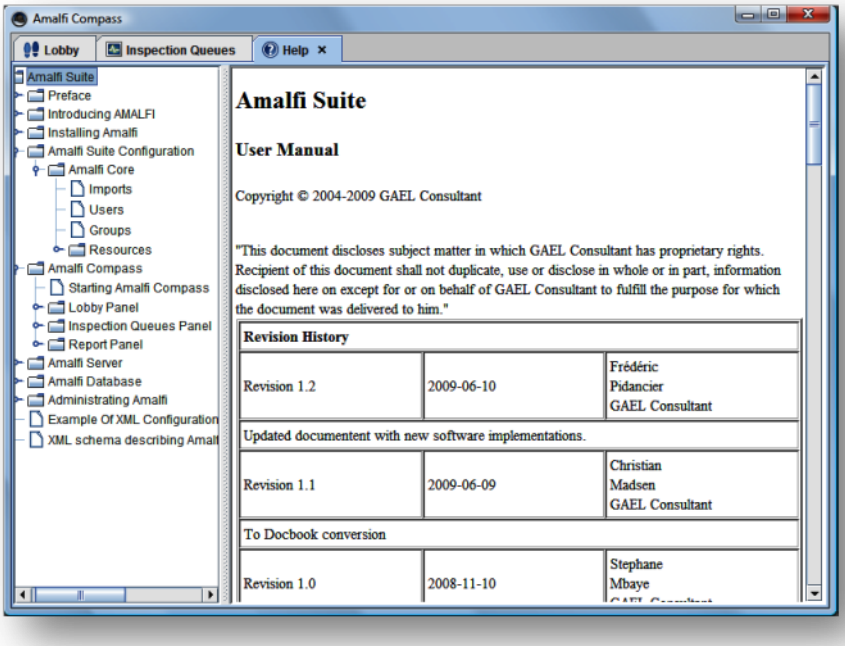


fig. 26 - Amalfi Compass - Help panel

Running Inspections

To achieve the main goal of the Amalfi Compass, which is to run a data inspection, press the "Inspect Item..." active area and select a file or a directory from the proposed file chooser. Pressing the "Ok" button submit the selected file for inspection to the default inspection queue. Then switch to the "Inspection Queues" panel by activating the corresponding tab at the top of the main window, to monitor the inspection activity and get the inspection results. Refer to the section §0 below for further information about this "Inspection Queues" panel.

Using Recent Inspections

Because you may need to repeat some operations several times, the Amalfi Compass keeps track of a series of items recently submitted: they are reported as a list of items below the "Recent Items" area of the panel. This list is ordered so that the most recent item is coming first. The list works in a "first in first out" mode, where the oldest item is evicted when a new recent item appears.

fig. 27 - Amalfi Compass - Purge of recent item list

This list of recent items is specially designed for operators that have to repeat frequently the submission of the same mount point (for example a DVD or a CDROM drive). It could also be useful when you need tuning the application and test it against the same series of items.

The list of recent items can be purged by pressing the broom icon on the right side of the "Recent Items" title.

Producing Inspection Reports

XML Reports

The XML Reports are textual dumps of the inspection results: they contain all information that where available from the "Inspection Queues" panel for an inspection row of the tabled tree component. XML report is the native low level format for reports processing. This report is stored as-is in the database and database Browser also authorize visualizing the report.

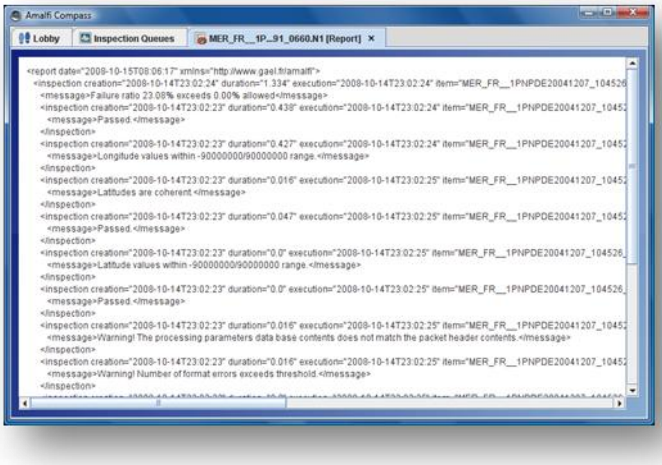


fig. 28 - Amalfi Compass - Inspection XML report

PDF reports

PDF report is a graphical processed version of the XML report.

Two kinds of reports are available:

- The first is the normal full report, including full information about processed inspections.
- The second is the digest report, also called Label (in Amalfi-1), only reporting a summary of the full report.

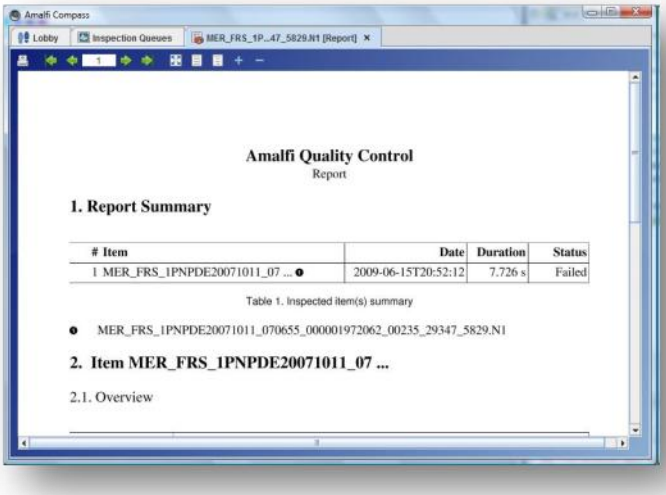


fig. 29 - Amalfi Compass - Inspection PDF report

Amalfi Server

This section deals with the Amalfi server command line interface. Not detailed in this document, another interface is proposed for interfacing file driven ground segment processing facilities. This interface, called "*IpfJobSubmitter*", is fully described in document "Amalfi-2/Payload Data Ground Segments - Interface Control Document" referenced "GAEL-P264-ICD-001-01-02".

The Amalfi Server application acts both as a server and as a client for interacting with other Amalfi Servers, e.g. submitting an item for inspection, restarting or stopping the server, starting or stopping services, etc.

Server Command Line Interface

Return code

The Amalfi Server returns 0 to the caller if the application ran successfully and any other value otherwise (a priori 1). “Successfully” does not mean that the output of any requested task is positive but that no unrecoverable error or exception occurred during the execution. For example, a successful completion of an inspection request does not mean that the inspected item passed the inspection but that an inspection report as actually been produced. On the contrary, an inconsistent setting of command line parameters, an unreachable server, or an invalid authentication, are considered as errors and lead to a non null returned code.

Logging System

The Amalfi Server application makes use of the Apache logging system Log4j and can therefore be configured to redirect logging messages in many ways without recompiling the application e.g. logging to standard output, to system logger, rolling logs, XML formatted logs, etc. Please refer the http://logging.apache.org/log4j Web site for further information.

General behaviour

When server is not present or unreachable, the submission client processes its own standalone inspection queue to successfully perform the inspection.

Inspections can be performed asynchronously only when a server is reachable, otherwise a synchronous submission is performed until the end of the inspection treatment.

When an output file is explicitly requested in the command line, the inspection is systematically synchronous, and the execution ends when report file has been produced.

Synopsis

```
java –Xmx1024m -jar <install-dir>/lib/java/amalfi-server-<vv.vv>.jar [ARGS]
```

This command line starts the java machine with reserving 1024 Mo of heap space to run the server. The application accepts arguments `[ARGS]` that are mainly dedicated to specify the server to be created or contacted, and related parameters such as user identification or network port number.

In the following examples, the Amalfi Server calling command has been substituted with the `server` abbreviation in order to reduce the line extents and make easier the reading.

Command Options

This table provides information about each of the command options for submitting jobs.

Command option	Description
--help	Displays help.
--version	Displays version.
--host	Defines the remote host
--port	Defines the port to be configured
--identifier	Defines the identifier of the inspection queue.
--name	Defines the name of the inspection queue.
--user	The queue connection username
--password	The queue connection password
--submit	Asynchronously submits an item
--submit-synchronous	Synchronously submits an item
--addon	Specify the add-on to perform inspections.
--start	Starts the server
--shutdown	Stops the server
--maximum-cache-size	Defines the maximum cache for inspection reports.
--maximum-thread-pool-size	Defines the number of simultaneously running threads.
--output	Defines the output report pattern
--status	Displays the status of the server.

table 4 - Amalfi server command line overview

Informative Arguments

The special arguments are processed first, whatever their positions in the command line. In this case, the command terminates without processing the others arguments.

```
--help
```

Prints out a documentation and usage about the Amalfi Server. The help is printed to the standard output and not as an INFO message throughout the logger.

```
--version
```

Prints out the version identification of the Amalfi Server package. The version is printed to the standard output and not as an INFO message throughout the logger.

Server identification and user authentication arguments

The server identification arguments are processed first and in the order of their appearance. All of these arguments shall not be repeated in the command line, even with the same values.

```
--host <host-name | host-ip-address>
```

The host name or IP address running the Amalfi Server to be contacted. --host switch is not mandatory and, if not provided, the local host is considered.

```
--port <port-number>
```

The network port-number on which the Amalfi Server to be contacted is listening. --port switch is not mandatory and, if not provided, the default port number is 1099.

The default port number 1099 is inherited from Java™ RMI technology that is used for the listening of remote object lookups and that is currently the baseline of the Amalfi Server remote protocol.

```
--identifier <queue-identifier>
```

The identifier of the server. This identifier will be used for distinguishing this server instance from other ones running on the same host.

If not provided, the default identifier is 'fr.gael.amalfi.server.Server'. The selection of the identifier is important since it will be matched to the resource entries of the current configuration: it modifies the configuration and thus the behavior of the queue e.g. for security considerations or in term of attached services (mailing, reporting, etc). Please, refer to the section §6.1 below for further information about the general configuration of Amalfi.

```
--name <queue-name>
```

The name of the server. If not provided, the default name is 'Amalfi Server'.

```
--user <user>
```

The user identification that authorizes the command execution on the target Amalfi Server.

The --user is not a mandatory argument, and if not provided, the command will be processed as an anonymous login. It is the configuration of the target Amalfi Server that governs the authentication and authorization policy.

```
--password <password>
```

The password to be associated to the user identification. The --password switch has no default value and can be provided only with the --user. In case --user has been provided with no --password switch,

the password will be required from the command line. To avoid interactive prompts that could block automated system, an empty string shall be provided as password e.g. --password ''.

If the Amalfi Server is responsible for the privacy control of the password while conveying through the network, the privacy control of the password provided on the command line is left to the caller. Writing the plain text password of an uncontrolled terminal or in a script with open permissions may lead to security issues. It is strongly encouraged to use the prompt to enter password in an interactive mode or to carefully set the access permissions to the scripts that use the --password switch.

Server command arguments

The server command arguments are processed next to those specified in the previous sections. They are considered in their order of appearance in the command line.

`--start`

Creates a new Amalfi Server on the local host i.e. --host is in this case ignored, and waits for remote commands: the server command arguments following --start are never processed. The started server can be safely stopped as any other process on the running Operating System e.g. pressing <Ctrl-C> key or killing the underlying Java™ Virtual Machine process. Example of output at server start:

```
[INFO] Starting server (Thu Nov 06 09:55:35 CET 2008)
[INFO] Loading configuration from "xxx/amalfi-configuration.xml"
[INFO] Server started.
```

`--status`

Prints outs the status of the specified server and those of the inspections it is running or those that are still in its cache. Example of status:

```
Server: "Amalfi Server" (fr.gael.amalfi.server.Server:1099)
RMI Registry: localhost:1099 (connected)
Queue: "Amalfi Server" (fr.gael.amalfi.server.Server) - connected
 - Status: running since Thu Nov 06 09:55:36 CET 2008
 - Thread Pool:
    - Size: 2
    - Core Size: 3
    - Max. Size: 5
    - Keep Alive Time: 10s
    - Active count: 0
    - Larget Size: 2
 - Inspected Items Count: 2
 - Inspection Cache: (2 over a max. of 10)
```

```
1 - MPH Inspection Plan (DOR_DOP_1POF-P20070917_..._2040.N1)
        \->Failed (Failure ratio 12.50% exceeds 0.00% allowed)
2 - All Applicable Inspections Plan (Automatic)
    (LED-ALAV2A019602720-O1B1___)
        \->- Passed (All 2 inspection(s) passed successfully)
```

`--submit <item-url>`

Submits a request to the target server for inspecting the item located at <item-url>. This submission is asynchronous, so it does not provide any result apart the successfulness of the submission. The result of the submission, i.e. the inspection result, could be accessed through a successive call with the status argument, and after an unpredictable delay depending on the target server load.

The <item-url> argument shall point to the item and shall be expressed from the target server point of view. If the corresponding URL is a relative location to the item, this item shall therefore be located on the host running the target item.

The <item-url> argument shall be a valid URL and shall, therefore, include a scheme protocol. Supported protocols are file, and ftp. Depending on the FTP server capability, the ftp scheme may require downloading the input file or directory tree.

If you try to submit an item and no server answers, a local queue is started just for submitting this item and is closed after.

`--submit-synchronous <item-url>`

Submits an inspection request to the target server but in synchronous mode. In this mode, the current call will wait for the result of the submitted inspection and will then print out an XML report on the standard output.

`--output <pattern>`

Defines a pattern of the file where will be stored the output report. If "—output" parameter is used, submission becomes synchronous, and the report is not produced on stantard out, but in the file matching the given pattern.

The pattern is the full path where the report will be written. This field is filtered to dynamically customize the output report name following these rules:

- "%d": current date expressed in ISO 8601 standard format up to the second
- "%i": item identifier
- "%f": product filename

  "[A-z-_.]*" static strings

For example, pattern "/usr/data/%i-report-%d.xml" may produce the following report name:

```
EN01_MER_FR__1P_20041207T104402_20041207T104736_14491-report-20110624T173715.xml
```

`--shutdown`

Shuts down commands in the server allow remotely stops the queue, but keep server instance alive to request server cache, and other passive operations. The command waits for the termination of the queue and for the termination of all inspections currently running: other passed or pending inspections are discarded.

```
--maximum-cache-size
```

Assigns the maximum number of inspections that should be kept in the starting server cache, or update this value on a target server.

```
--maximum-thread-pool-size
```

Assigns the maximum number of threads to be used for processing inspections in the starting server, or update this value on a target server.

Server inspection loading

When a server is starting, it can load inspections from configuration file or from addons files specified in the server command line argument.

```
--addon [<add_ons_path>]*
```

Loads given inspection(s) <add_ons_path> spaces-separated files. The add-ons already configured (via amalfi-configuration.xml file, see p.46) for the application are ignored. This switch has no effect, and a warning message is raised when submission is performed to a remote Amalfi server.

As described in the chapter "Adding or modifying an Add-on" p.79, addon are taken into account if they are present in the application CLASSPATH. The add-ons passed via configuration file and command line parameters are also placed in the application CLASSPATH by the application.

Connection to a remote server

```
server --host my.domain.org --port 8080
```

The previous example illustrates a connection to an Amalfi Server running of “my.domaing.org” host and listening port 8080. This command only prints something if the connection failed.

Connection a remote server with specified user identification

```
server --host my.domain.org --user wallas
```

This example still reaches an Amalfi Server running of “my.domaing.org” host but this time listening on the default port e.g. 1099, and providing user identification. Because no password has been provided, it is requested from the standard input.

Connection a remote server with specified user identification and password

```
server --host my.domain.org --user wallas --password secret
```

This example still reaches an Amalfi Server running of “my.domaing.org” listening on the default port e.g. 1099, and providing user identification but the password is now provided on the command line, avoiding any interactive prompt.

Submission of an item located on the Server local file system

```
server --submit file:///mnt/cdrom/
```

In this example, the Amalfi Server will queue the item located on the file:///mnt/cdrom directory on the local file system: this kind of call has particular sense when launched from the same host as the target Amalfi Server. The item type will be determined according to the automatic ruling and configuration of the server. The inspection(s) will also be selected automatically according to the recognized item type. The determination of both the item type and the applicable inspections will not depend on the caller but on the target Amalfi Server.

If no server is configured on local host, a standalone queue will be started to locally perform requested inspection. In this case, a warning is raise on the console.

Submission an item through FTP

```
server --submit ftp://www.domain.org/path-to-item
```

Same example as in the previous section, but the item is to be accessed through FTP. This kind of call has particular sense when the item is located on a host different from the Amalfi Server. Item type and applicable inspection determination would be conducted as in the previous example.

The URL to the item shall be expressed from the target server point of view. Supported protocols are file, and ftp. Depending on the FTP server capability, the ftp scheme may require downloading the input file or directory tree.

Synchronous Standalone Submissions of an item

The standalone behaviour is active when no server is reachable. In the following standalone examples, we consider that no server has been locally started on the system with default settings (id="`fr.gael.amalfi.server.Server`", port=1099, host="`localhost`").

```
server --submit-synchronous file:///mnt/cdrom/
```

In this example, the server performs a synchronous submission. Once the treatment of the inspection finished, the report is printed in the standard output because no output file is specified.

```
server --submit file:///mnt/cdrom/
       --output /user/data/%i-report-%d.xml
```

This example executes a synchronous execution (even if "`--submit`" usually run asynchronously) because an output is specified. The processing shall wait the end of the processing, that the report is available to produce the requested output file.

Submission of an item using a specific add-on

```
server --addon drbx-cortex-topic-envisat-1-0-rc-12.jar \
               /home/user/amalfi/add-ons/my_envisat/ \
       --submit file:///mnt/cdrom/MER_FR__1PXX.N1
```

This example demonstrates how to execute an inspection plan configured in two separate pathes (cortex-topic-envisat-1-0-rc-2.jar file that contains the drb schemas and /home/user/amalfi/add-ons/my_envisat that contains Amalfi inspection definitions).

Amalfi Database

Amalfi Suite contains a database module. This module is used to store inspection processing information into a persistent support. To use Amalfi database, it is necessary to select database module during Amalfi installation, and follows the installation instructions.

Prerequisites

Amalfi database application requires application MySql Version 5.0 or higher being installed and available. Amalfi database module includes facilities to fully setup the necessary database elements required to run Amalfi database applications and services. (See section "Database installation" p.45 for details)

Database module

Database module contains multiple kinds of Amalfi features:

- A Database resource for Amalfi
- Amalfi database tools (browser, statistics)
- Amalfi database administration tools

Database Amalfi resource

Declaring the database resource

The database resource aims to declare the database for the Amalfi applications. The configuration shall be performed with defining this database access into the configuration file (See "Databases resources" p.57). Once configured all the Amalfi resources are allowed to connect this database.

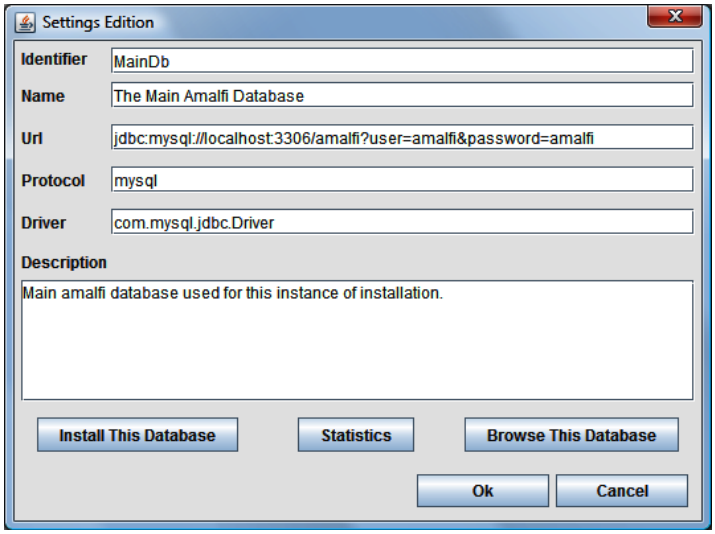


fig. 30 - Amalfi Database resource configuration panel – Compass Settings

Declaring an Amalfi queue database broadcaster service

Once the database resource configured, the inspection queue can also be configured to store inspection result into this database. The mechanism is called "database broadcaster service". For manual configuration See "Database broadcaster service" p.54. Compass also proposes a GUI to update this configuration.

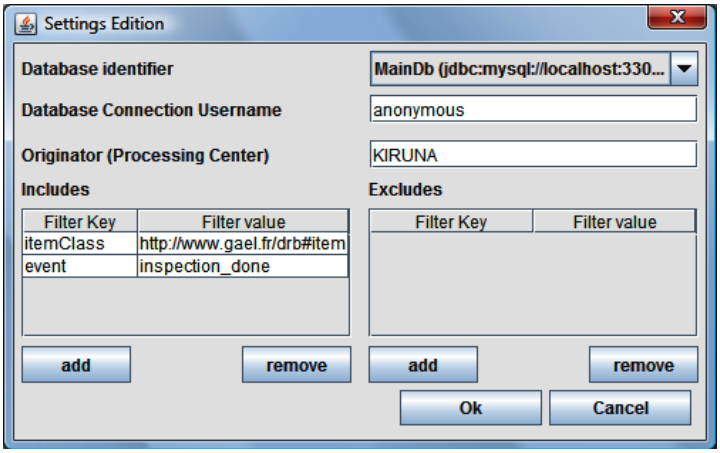


fig. 31 - Amalfi Database service configuration panel

Database tools

Database browsing

Usually database systems propose basic shell to browse inside database. Amalfi implements a dedicated user interface to access and navigates inside the amalfi database tables.

To execute database browser, it is necessary to have installed the database module and execute the following command:

```
java -jar lib/java/amalfi-database-X-X.jar
                        fr.gael.amalfi.database.DatabaseBrowser
```

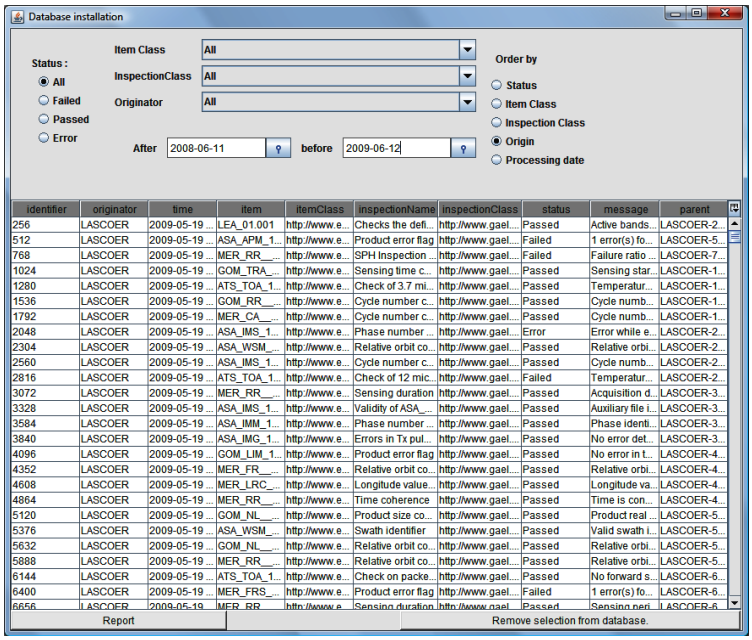


fig. 32 - Amalfi Database browsing panel

Database statistics

Amalfi database module proposes a tool able to customize and to compute database statistics. The customization is performed according to different themes available via upper left combo box, while upper right two combo boxes allow limiting the statistics period.

Statistic GUI panel can be accessed with executing the following command:

```
java -jar lib/java/amalfi-database-X-X.jar
      fr.gael.amalfi.database.statistics.StatisticsGUI
```

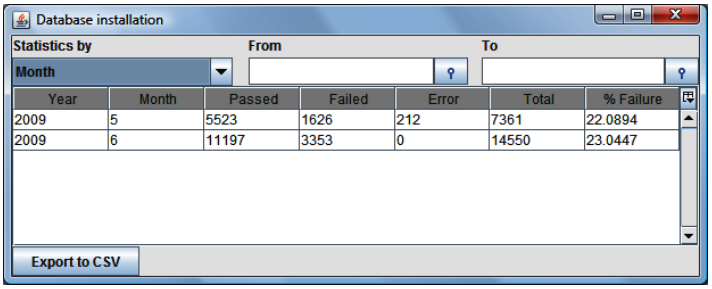


fig. 33 - Amalfi Database statistics panel

Database administration

Purge

Manual database cleanup is possible via database browser operations. In a more operational mode, the scheduler (see "Scheduler (Cron)", p.59) also proposes a set of jobs dedicated to database management.

The scheduler job allows light purge, or full purge. Light purge means that the report entry will be removed from database but inspection information stays available for statistics. Full purge means

that all the references to the inspection will be removed and it will not be possible to perform statistics after this action (see "`fr.gael.amalfi.database.service.DatabasePurge`" p.61).

Databases synchronizations

Scheduler also proposes a synchronisation job, able to resynchronize one database with another (see "`fr.gael.amalfi.database.service.DatabaseSynchronize`" p.62).

Database installation

Amalfi database installation comes with a specific database installation GUI that can be launched from Compass Setting panel or from the following command line:

```
java -jar lib/java/amalfi-database-X-X.jar
     fr.gael.amalfi.database.Installation
```

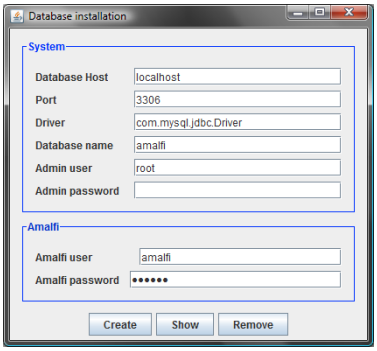


fig. 34 - Amalfi Database statistics panel

The database installation panel fully installs the tables required by Amalfi. If a granted administration user is provided, it also creates the administrative actions such as user and database creation.

Amalfi Configuration

Amalfi Suite is a highly configurable System. A unique configuration file is used to setup all the elements of Amalfi Suite. First section of this chapter describes the configuration of each resources of the suite. In an other hand, Amalfi can be enriched with add-ons. Add-ons are pluggable packs that can be connected into Amalfi to add inspection and mission's extensions capabilities. The second section describes these add-ons, how they are architectured.

Details of the configuration file

Amalfi Core is the kernel of Amalfi suite. All the module of Amalfi Suite uses this mandatory module to get configuration information. Thanks to this architecture, all the modules can be configured with the same configuration file. This file is XML encoded, and its syntax is governed by the XML schema defined at "*Appendix – Configuration File XML Schema (normative)*".

The configuration file used by all the Amalfi modules is located in installation directory path `etc/amalfi-configuration.xml`. This file can be edited manually with a simple text/xml editor.

This chapter defines each section of configuration able to modify Amalfi suite behavior. Configuration is split into 4 main sections:

- Imports : importing external configurations.
- Users : configure users parameters.
- Groups : configure groups parameters.
- Resources : configure Amalfi resources.

In special cases, when Amalfi is installed in a not writable directory, the Amalfi configuration file can be customized with creating the "`.amalfi/etc/amalfi-configuration.xml`" file into user directory (aka `${user.home}/.amalfi/etc/amalfi-configuration.xml` directory). If exists, this file fully override the default configuration file existing in the installation directory that is ignored. When using Compass setting edition section, you are creating or modifying this file.

Imports

External configurations can be imported using this keyword. Imported configuration never overrides existing configured resource.

```
<import url=http://user:pass@server/path/to/config/file
   cache="file:/path/to/local/cache"/>
```

Key	Synopsis
`@url`	The URL to the configuration file to be imported.

Key	Synopsis
@cache	The URL path where to store a copy of the imported configuration.  This path is used to store file to speed up accesses, and in case of network unavailability.

table 5 - Import attributes

Amalfi Compass is able to browse imported configuration in read only mode.

Users

The user section in configuration file allows edition of user parameters such as its name, e-mail, or password connection. These users are used in Amalfi for various authentications and getting permissions (inspection queues, database privileges). Default and mandatory user is `anonymous`, who has no password. The configuration allows an unlimited number of users:

```
<users>
  <user id="anonymous">
    <name>Anonymous User</name>
  </user>
  <user id="queue-administrator">
    <name>Inspection Queue Administrator</name>
    <mail>queue.administrator@amalfi.com</mail>
    <password algorithm="MD5">
      3cc8588bc6fb10cf0fd5907832f1cd8d
    </password>
  </user>
</users>
```

Key	Synopsis
@id	The unique identifier of the configured user.
name	The real name of the user.
mail	The e-mail address of the user.
password	The password connection for this user (used in server only).
password/@algorithm	The password encoding algorithm (- , SHA1, MD2, MD5...).

table 6 - User attributes

Groups

Close to the Unix *groups*, Amalfi groups can gather a set of users or a set of groups. It allows factorizing the users in a common category organized by *group*.

```
<groups>
  <group id="queue">
    <member>queue-submitter</member>
    <member>queue-administrator</member>
  </group>
  <group id="users">
    <member type="group">queue</member>
    <member type="group">database</member>
    <member type="user">anonymous</member>
  </group>
</groups>
```

Key	Synopsis
`@url`	The unique identifier of the configured group.
`member`	The identifier ot a member of this group. Member section allows the attribute `type` able to be configured as followed: – `user` the member is a user identifier – `group` the member is a group identifier Default is `user`.

table 7 - Group attributes

Resources

This configuration section describes how to setup amalfi resources. All the configuration are applied to these resources thanks to the mandatory attribute called `id` and the configuration type `xsi:type`. The existing resources are:

- Inspection Queues : `xsi:type` shall be `QueueResource`
- Databases: `xsi:type` shall be `DatabaseResource`
- Scheduler: `xsi:type` shall be `CronResource`
- Add-on definitions: `xsi:type` shall be `OntologyResource`
- Add-on repositories: `xsi:type` shall be `OntologyRepositoryResource`

The identifiers `id` must be unique to avoid confusion between resources.

Inspection Queue resources

This resource defines a configuration for an inspection queue. Inspection queue is the most complex resource of the amalfi configuration, because is shall be possible to manage privileges and a set of services to start.

The simplest inspection queue shall define an identifier, a name and basic privileges as defined in the section of code here after. Otherwise it is possible to setup services (See section §0, “Services”) which will be started during the queue initialization if present in configuration.

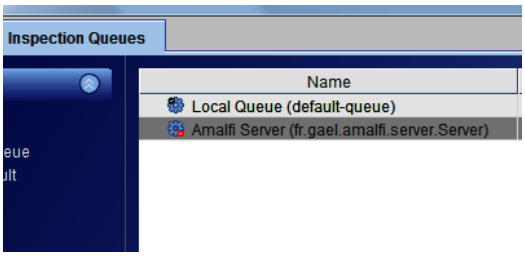
```
<resource xsi:type="QueueResource" id="default-queue">
  <name>Default Inspection Queue</name>
  <description>
    This default inspection queue is mandatory in the configuration
    to manage not configured queues.
  </description>
  <url>
    rmi://localhost:1099/fr.gael.amalfi.server.Server
  </url>
  <!-- Definition of privileges: who could access this queue? -->
  <privileges>
    <privilege id="submit">
      <policy>
        <deny> <all/> </deny>
        <allow>
          <user>anonymous</user>
        </allow>
      </policy>
    </privilege>
    <privilege id="shutdown">
      <policy>
        <deny> <all/> </deny>
        <allow>
          <user>anonymous</user>
        </allow>
      </policy>
    </privilege>
  </privileges>
```

Key	Synopsis
`@id`	The unique identifier of the configured group.
`@xsi:type`	Always `QueueResource`.
`name`	The name of the resource.
`description`	A description of the resource.
`url`	An informative URL for the resource (not used in inspection queue).
`privileges`	List of user privileges to setup queue. (See section §0, “Privileges”).
`services`	List of user services to start in this queue. (See section §0, “Services”).

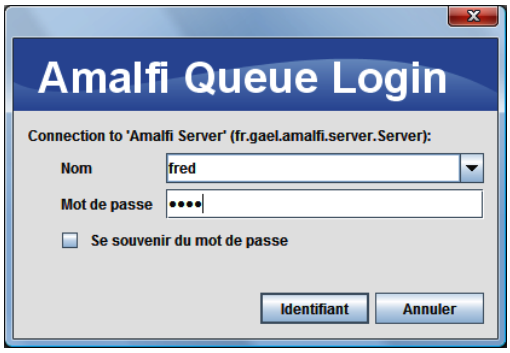
table 8 - Inspection queue resource attributes

Privileges

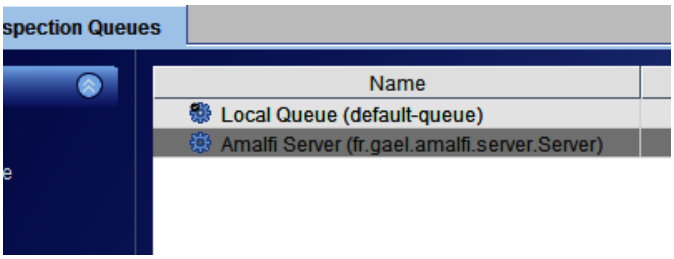
Privileges are only used in remote inspection queues such as amalfi server. Default local queue initially started in Amalfi Compass is not constrained by these privileges.



Connected to a server, but not logged-in.



Login panel of Compass.



Connected and logged-in: submission can be performed.

fig. 35 - Authentication steps with Amalfi Compass

The snapshot here before presents the Amalfi queue connection steps. To configure privileges, has shown in previous section 6.1.4.1, “Inspection queue resource”. It is necessary to add in the inspection queue resource a list of privileges. Privilege shall respect the following syntax:

Key	Synopsis
`@id`	A set of privilege have already been defined: – `submit`: this privilege policy is to be applied when submitting item in this queue. – `shutdown`: this privilege policy is to be applied when queue is stopped. – `query`: this privilege policy is to be applied when querying database.
`policy`	Policy defines a grant or deny accesses for this identified privilege: – `allow`: Allow access to followed defined user/groups/all. Selecting "all" remove all other configured user or group – `deny`: Deny access to followed defined user/groups/all. Selecting "all" remove all other configured user of group. Deny got the priority above allow.

table 9 - Privileges attributes

Services

Inspection Queue is able to carry out a set of services. If defined in the services section of an inspection queue resource, the service is automatically started at queue initialization. Two main categories of services are available:

- `xsi:type="ReportBroadcasterService"` Performs action according to an event in the queue.
- `xsi:type="FileScannerService"` Wake-up the queue when an external even appears.

First, the report broadcaster is able to broadcast inspections into mails or database. Second, the FileScanner monitor path to submit inspection.

E-mail broadcaster service

The e-mail broadcaster is a service to be attached to an inspection queue, that is able to send inspection report by e-mail.

```
<service id="reporting-broadcaster"
  xsi:type="ReportBroadcasterService">
  <includes>
    <itemClass>http://www.esa.int/envisat#product</itemClass>
    <inspectionStatus>error</inspectionStatus>
    <inspectionStatus>failed</inspectionStatus>
  </includes>
  <broadcaster type="email">
    <mailer>
      <server>
        <hostname>smtp.domain.org</hostname>
        <port>25</port>
        <username>smtp-user</username>
        <password encrypted=”false”>smtp-password</password>
        <TLS>true</TLS>
      </server>
      <from name="Amalfi Inspection Report">amalfi@amalfi.com</from>
    </mailer>
    <to type="group">administrators</to>
    <to type="email">frederic.pidancier@gael.fr</to>
    <cc type="user">wallace</cc>
    <bcc type="email">bond@sis.gov.uk</bcc>
    <emailReportName>%i-report-%d</emailReportName>
  </broadcaster>
</service>
```

First, as all the broadcaster services, it is possible to configure the filter to select when and why the broadcaster will be executed. The filters are optional but can be configure as the following table:

Filter key	Description	Samples
`itemClass`	The DrbItem Class	`http://www.gael.fr/drb#item`
`inspectionStatus`	The status of the inspection	– `pending` – `running` – `done` – `cancelled` – `passed` – `failed` – `error` – `undefined`
`event`	The inspection or queue significant events	– `inspection_ran` – `inspection_done` – `inspection_cancelled` – `queue_shutdown` – `queue_inspection_added` – `queue_inspection_removed`

table 10 - Filters keys

The e-mail broadcaster presented here before, is filtered to send report only for Envisat products, and in the case the inspection fails or throws an error. Second, the broadcaster is configured to send e-mail. So type is configured to `email`, and broadcaster is configured as following:

Key	Synopsis
`@type`	Always `email`.

Key	Synopsis
mailer	This is the mail server configuration with following internal parameters: – server/hostname: The smtp server host – server/port: The smtp server port (default 25) – server/username: The smtp connection user name – server/password: The smtp connection password – server/password/encrypted: Is this password encrypted? – server/TLS: Is your server using TLS encryption? (default false) – from: Defines the e-mail address of the mail sender. – from/@name: Defines the name of the sender.
To	Identifier of the user/group/email to send the report to.
cc	Identifier of the user/group/email to send the report as copy (Cced).
bcc	Identifier of the user/group/email to send the report as hidden copy (Bcced)
[to/cc/bcc]/@type	Type of sender identifier. If sender identifier type is user, email will be sent to the user related from the user section of the configuration file. If sender identifier type is group, email will be sent to all the users member of the group. And If the sender identifier type is email the identifier passed is an e-mail. Default is email.
emailReportName	Specify how to format the report's name that will be sended. You can use some parameters like : – %d: current date expressed in ISO 8601 standard format up to the second – %i: item identifier

table 11 - E-mail broadcaster service attributes

Database broadcaster service

This broadcaster is able to store inspection result into a database when the inspection is performed.

Similarly to email Broadcaster, Database broadcaster is automatically executed according to the broadcaster filters has defines in the previous section §0, “E-mail broadcaster service”.

The database broadcaster references a database resource as defined in section §0, “Databases resources”. If not configured, the service will not run.

```
<service id="reporting-broadcaster"
  xsi:type="ReportBroadcasterService">
  <includes>
    <itemClass>http://www.esa.int/envisat#product</itemClass>
      <event>inspection_done</event>
  </includes>
  <broadcaster type="database">
    <database identifier="MainDb">
      <connectionUserId>smith</connectionUserId>
      <originator>ESRIN</originator>
    </database>
  </broadcaster>
</service>
```

The here before service broadcaster configuration sets the inspection queue to store inspection report into configured database called `MainDb` using submitter `smith`, and defining originator for this queue as `ESRIN`.

Key	Synopsis
`@type`	Always `database`.
`database/@identifier`	The database resource identifier name (as defined in `@id` of section §0 “Databases resources”).
`database/connectionUserId`	The user name used to check database privileges. This user identifier references the user section of the configuration file described here before at section §6.1.2, “Users”.
`database/originator`	A free text string to identify the processing center. This string is use in database to perform statistics by processing center.

table 12 - Database broadcaster service attributes

File broadcaster service

This broadcaster is able to store inspection report into a specified directory.

Similarly to email broadcaster and database broadcaster, file broadcaster is automatically executed according to the broadcaster filters has defines in the previous section §0, “E-mail broadcaster service”.

The file broadcaster is defined with an url specifying the path to the directory where stores the reports.

```
<service id="file-broadcaster" xsi:type="ReportBroadcasterService">
  <includes>
    <itemClass>http://www.esa.int/envisat#product</itemClass>
      <event>inspection_done</event>
  </includes>
  <broadcaster type="file">
    <file><url>file:/C:/data/report</url></file>
    <fileReportName>%i-report-%d</fileReportName>
  </broadcaster>
</service>
```

The here before service broadcaster configuration sets the inspection queue to store inspection report into configured directory `url`.

Key	Synopsis
`@type`	Always `file`.
`file/url`	The path to the directory where saving reports. If this path does not exist, it will be created.
`fileReportName`	Specify how to format the report's name that will be sended. You can use some parameters like : – %d: current date expressed in ISO 8601 standard format up to the second – %i: item identifier

table 13 - File broadcaster service attributes

File scanner service

The file scanner service is a service able to scan local directory files, according to filters, and submit candidates to the attached inspection queue.

```
<service id="N1-AVNIR2-fileScanner" xsi:type="FileScannerService">
  <baseUrl>file:/anywhere/repository</baseUrl>
  <nameMatch>.*\.N1</nameMatch>
  <nameMatch>ALAV2.*</nameMatch>
  <scanningPeriod>PT10S</scanningPeriod>
  <recursive>true</recursive>
</service>
```

Configuration for this service shall respects the following attributes:

Key	Synopsis
`@xsi:type`	Always `FileScannerService`.
`baseUrl`	Path to the directory to be scanned.
`nameMatch`	Pattern to define files to be retained inside the scan (multiple nameMatch is possible).
`scanningPeriod`	Delay period between scans (XML Duration format).
`recursive`	scan shall be recursively performed in sub-folders. (`true|false`).

table 14 - File scanner service attributes

Databases resources

Database resources are used by inspection Queue to store inspection reports (See here before queue configuration service “Database broadcaster service” p54. Database module carries out a set of application to manipulate these configured databases (See "Amalfi Database" p42). This resource is the root configuration for all Amalfi modules that uses databases. It defines the physical location of database. The following shows a piece of configuration of a database resource:

```
<resource xsi:type="DatabaseResource" id="MainDb">
  <name>The Main Amalfi Database</name>
  <description>
    Main amalfi database used for this instance of installation.
```

```
  </description>
  <url>jdbc:mysql://localhost:3306/amalfi</url>
  <login>xxx</login>
  <password encrypted=”false”>xxx</password>
  <driver protocol="mysql">com.mysql.jdbc.Driver</driver>
  <privileges>
    <privilege id="submit">
      <policy> <allow> <group><all/></group> </allow> </policy>
    </privilege>
    <privilege id="query">
      <policy>
        <deny> <all/> </deny>
        <allow> <user>smith</user> </allow>
      </policy>
    </privilege>
  </privileges>
</resource>
```

In this resource, the most important element is the `url` that fully defines physical location and access right to the database.

Key	Synopsis
`@xsi:type`	Always `DatabaseResource`.
`descriptionn`	Textual description of the resource.
`url`	Valid Url to access database.
`login`	Login to access database.
`password`	Password to access database.
`password/@encrypted`	Defines if given password is encrypted or not.
`driver`	The driver to be used to access database (default is `com.mysql.jdbc.Driver`).

Key	Synopsis
driver/@protocol	The protocol to be used by the driver (default is mysql).
privileges	The privileges promoting an Amalfi user to get read/write access into the database. For details, See “Privileges”.p50.

table 15 - Database resources attributes

Scheduler (Cron)

Scheduler is a small application able to execute jobs at date and time defined by the user. The other name given to the Scheduler is *Cron* because it is very similar to Unix Cron daemon.

```
<resource xsi:type="CronResource" id="CronSceduler">
  <name>Amalfi Cron scheduler</name>
  <description>
    Scheduler used in amalfi to perform various periodic treatments
  </description>
  <url>http://www.gael.fr/amalfi#schduler</url>
  <cron compensation="0">

<!-- Performs Purge (Report content only "full"=false) last one year
(-P1Y) of inputs in database at a period of all the years in
January, 5th at 00h00 ("00 00 05 01 *") -->

<job id="purge" name="Performing database Purge" schedule="00 00 05
01 *" disabled="false"     startup="false">
    <class name="fr.gael.amalfi.database.service.DatabasePurge">
      <param name="url" value="jdbc:mysql://local:3306/amalfi"/>
      <param name="user" value="anonymous"/>
      <param name="db_login" value="amalfi"/>
      <param name="db_password" value="amalfi"/>
      <param name="db_encrypted" value="false"/>
      <param name="since" value="-P1Y"/>
      <param name="to" value="PT0S"/>
      <param name="full" value="false"/>
    </class>
  </job>
  </cron>
</resource>
```

A cron is a set of jobs to periodically execute at defined dates. Amalfi already implements a set of jobs to be used in the scheduler (See section 6.1.4.3.1, “List of available jobs classes”).

Parameters shall be defined as following:

<table>
<tr><th>Key</th><th>Synopsis</th></tr>
<tr><td>@xsi:type</td><td>Always CronResource.</td></tr>
<tr><td>@id</td><td>Resource identifier.</td></tr>
<tr><td>name</td><td>Resource name.</td></tr>
<tr><td>description</td><td>Resource description.</td></tr>
<tr><td>url</td><td>Informative resource URL.</td></tr>
<tr><td>cron/@compensation</td><td>Value used to compensate system time error if any.</td></tr>
<tr><td>cron/job</td><td>A Job is one of the process to perform. Each job has its own schedule and execution thread.</td></tr>
<tr><td>cron/job/@id</td><td>Identifier of the job.</td></tr>
<tr><td>cron/job/@name</td><td>Name of the job.</td></tr>
<tr><td>cron/job/schedule</td><td>The schedule to process the job using crontab(5) syntax:
<table>
<tr><th>Field #</th><th>Allowed</th><th>Values</th></tr>
<tr><td>1</td><td>minute</td><td>0-59</td></tr>
<tr><td>2</td><td>hour</td><td>0-23</td></tr>
<tr><td>3</td><td>day of month</td><td>0-31</td></tr>
<tr><td>4</td><td>month</td><td>1-12</td></tr>
<tr><td>5</td><td>day of week</td><td>0-6 (0 is Sunday)</td></tr>
</table>
table 16 - Scheduler string syntax</td></tr>
</table>

Key	Synopsis
cron/job/disabled	Disabled this job execution if true. (default is false).
cron/job/startup	Execute job when starting scheduler (outside the scheduled date) (default is false).
cron/job/class	This entry defines the physical class to execute in the job.
cron/job/class/@name	The name of the class to be executed. This name shall be a valid class name available in the scheduler classpath. Amalfi already implements a set of classes (See section “List of available jobs classes” p61 here below).

table 17 - Scheduler resource parameters

List of available jobs classes

fr.gael.amalfi.database.service.DatabasePurge

Perform database purges according to the following parameters:

Key	Synopsis
url	The url of the database to purge.
db_login	Login to access database.
db_password	Password to access database.
db_encrypted	Defines if given password is encrypted or not.
since	Duration/date since when the database shall be purged.
to	Duration/date to when the database shall be purged.
full	boolean (true|false) indicating if full purge or only removes report contents shall be performed.

table 18 - Database purge job class

In these parameters, duration/date can be mixed. Valid date format are:

- yyyy-MM-dd'T'HH:mm:ss,SSSZ
- yyyy-MM-dd'T'HH:mm:ss,SSS
- yyyy-MM-dd HH:mm:ss
- yyyy/MM/dd HH:mm:ss

- `EEE d MMMM yyyy, HH:mm:ss '(UTC'Z')'`
- `EEE, d MMM yyyy HH:mm:ss Z`

- `PnYnMTnHnMnS` (ISO 8601 duration format)

fr.gael.amalfi.database.service.DatabaseSynchronize

Perform synchronization of two Amalfi databases.

Key	Synopsis
`to_url`	Destination database URL.
`to_login`	Login to access destination database.
`to_password`	Password to access destination database.
`to_encrypted`	Defines if given password is encrypted or not.
`from_url`	Source database URL.
`from_login`	Login to access source database.
`from_password`	Password to access source database.
`from_encrypted`	Defines if given password is encrypted or not.

table 19 - Database synchronization job class

fr.gael.amalfi.database.service.SystemCommand

Allow executing system commands via the system shell. Valid parameters include:

Key	Synopsis
`dir`	Working directory.
`fork`	True, if the process should fork (i.e. not wait for the process to exit).
`param0`	The first parameter.

Key	Synopsis
param1	The second parameter.
param2...	So on and so forth.

table 20 - System command job class

Add-on's

Add-ons are pluggable modules able to fully describes a mission and provide inspection features, full description of add-on is available in the next chapter, "Amalfi Add-ons" p.68. The add-on configuration can be a jar or a path, otherwise, it shall be possible to reference a remote repository to retrieve remote add-ons. The simplest way to configure an add-on is to directly provide the path to the jar file, and its dependencies. This is call "jar" resolution type. Another way to univocally retrieve Add-ons consists in using the concept of project object model `POM` defined by apache/maven project (Maven project Site – http://maven.apache.org/).

A complete description of Amalfi add-on is available in section "Amalfi Add-ons", p.68.

Add-on resolution with jar method

This method consists in defining the list of files/directory necessary to manage the mission. Here a sample of such a configuration:

```
<resource tintype="ontology resource" id="landsat" type="jar">
  <name>Amalfi LANDSAT Ontology resource</name>
  <description>
    The Add-on used to decode and retrieve Amalfi tests for
    the overall LANDSAT products.
  </description>
  <url>http://www.gael.fr/drb#landsat</url>
  <configuration>
  <![DATA[
  <jar name="LANDSAT Inspections" followInternalDependencies="false"
  xmlns="http://www.gael.fr/classpath/resolver/jar/configuration">
    <path>
      file:/c:/Dev/confs/landsat/ceos/src/main/resources/
    </path>
    <dependencies>
      <dependency>
```

```
        file:/c:/Dev/confs/drbx/topics/landsat/src/main/resources/
      </dependency>
      <dependency>
   file:/c:/Dev/libs/drbx-cortex-topic-ceos-1-0-beta-5.jar
      </dependency>
    </dependencies>
  </jar>
  ]]>
  </configuration>
</resource>
```

Key	Synopsis
`@xsi:type`	Always `OntologyResource`.
`@type`	Always `jar`.
`@id`	Resource identifier.
`name`	Resource name.
`description`	Resource description.
`url`	Informative resource URL.
`configuration`	Resolver configuration parameters.
`configuration/jar/@name`	Name of jar resolver.
`configuration/jar/@followInternalDependencies`	In the case of the provided file is a jar file, under some condition it is possible to retrieve dependencies required by the jar. If this parameter is set to `true`, system will try to resolve dependency defined inside the jar.
`configuration/jar/path`	The path (url formatted) to the Add-on to be loaded.
`configuration/jar/dependencies`	Dependency list of paths (url formatted) of the Add-on dependency.

table 21 - Add-on resource parameters

This configuration is simple to use and to setup, but does not propose cache capabilities and data visioning management.

Add-on resolution with maven

The maven resolution method uses a server repository to resolve project model. Main information of a project model, are `groupId`, `artifactId` and `version` that allow to univocally give access to the requested add-on and its dependencies.

In contrary to the jar method, when updated the version of Add-on, all the dependencies will automatically be updated thanks to the object model dependencies.

Libraries are stored in cache that protected by MD5, and all the new loaded Add-on in the cache are secured with this encryption.

Configuration of such a resolution can be defined as follow:

```
<resource xsi:type="OntologyResource" id="envisat" type="maven">
  <name>Amalfi ENVISAT Ontology resource</name>
  <description>
    The Ontologies used to decode and retrieve Amalfi
    inspections for the overall ENVISAT products.
  </description>
  <url>http://www.gael.fr/drb#envisat</url>
  <configuration>
  <![CDATA[

  <project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>fr.gael.amalfi</groupId>
    <artifactId>amalfi-config-envisat</artifactId>
    <packaging>jar</packaging>
    <version>1-0-rc-3</version>
  </project>

  ]]>
  </configuration>
</resource>
```

With this method, configuration only contains a POM section. The maven repositories are configured in the Amalfi resource defined in the next section "Add-on maven repositories" p.66.

Key	Synopsis

Key	Synopsis
`@xsi:type`	Always `OntologyResource`.
`@type`	Always `maven`.
`@id`	Resource identifier.
`name`	Resource name.
`description`	Resource description.
`url`	Informative resource URL.
`configuration`	Resolver configuration parameters.
`configuration/maven/project`	This is the entry point for the POM definition. A complete definition of this data set is available at POM Reference (http://maven.apache.org/pom.html).

table 22 - Add-on resource parameters

Add-on maven repositories

Add-ons repositories are remote servers containing the up-to-date versions of add-ons. It is possible to configure multiple repositories to retrieve different add-ons, or in case of unavailability of one of these servers.

The following configuration fragment shows the definition of such a configuration:

```
<resource xsi:type="OntologyRepositoryResource" id="gael-main">
  <name>Main Gael Ontology server</name>
  <description>Gael's ontologies server.</description>
  <url>http://www.gael.fr/software/distributions</url>
</resource>
```

Key	Synopsis
`@xsi:type`	Always `OntologyRepositoryResource`.
`@id`	Resource identifier.
`name`	Resource name.

description	Resource description.
url	The url of the repository. It specifies both the location and the transport protocol to be used to transfer a built artifact (tested protocols are http, https, scp).
username	User name to connect this repository.
password	Password to connect this repository.
privateKey	Private key for SSH style connection.
passphrase	Pass phrase for SSH style connection.

table 23 - Add-on repository resource parameters

Amalfi Add-ons

The Ontology Add-ons relies on *DRB Cortex Extension* component to store and deploy the *Representation Information* to be associated to the data to be inspected. Without repeating the definition of *Add-ons* already introduced previously in this book, it is reminded here that the *Add-ons* are *JAR* files that contains information necessary to Amalfi for classifying the data types (*Item Classes*), defining the inspections applicable to these types and other secondary resources as image access definition, meta data extraction schemes, icons or style-sheets. All these resources are consistently defined in pieces of an *Ontology Model* described by *OWL* files.

Location of the distributed add-ons

All add-ons *JAR* files of an Amalfi installation are located in the "`addons/`" subdirectory of the base installation point.

According to an internal convention of GAEL Consultant, the names of the *JAR Add-ons* distributed along with Amalfi are following one of the two patterns:

- `amalfi-config-<addon-identifier>-<version>.jar`
- `drbx-cortex-topic-<addon-identifier>-<version>.jar`

The *JAR* files starting with `amalfi-config` are the starting points of *Amalfi Configuration*'s previously described in XXX: they contain the inspection definitions and other Amalfi specific resources. The other *JAR* files starting with `drbx-cortex-topic` are dependent *Topics* that generally provide the support a consistent set of data types. Add-ons and topics are deployed in separate jar files, but it is possible to merge them in a unique add-on jar.

A typical example of this subdirectory content is the following:

```
-rwx------     3428 08:28 amalfi-config-alos-1-0-rc-1.jar (*)
-rwx------    11148 08:28 amalfi-config-envisat-1-0-rc-1.jar
-rwx------     2222 08:28 amalfi-config-ers-1-0-rc-1.jar
-rwx------    85420 08:29 drbx-cortex-topic-alos-1-0-rc-1.jar (*)
-rwx------    11743 08:29 drbx-cortex-topic-ceos-1-0-beta-4.jar (*)
-rwx------   370232 08:29 drbx-cortex-topic-envisat-1-0-rc-1.jar
-rwx------    44119 08:29 drbx-cortex-topic-ers-1-0-rc-1.jar
```

In these examples, a series of *JAR* files have been highlighted by *(*)*: all of them illustrate a set of dependent *Topics* that provide support to a set of data type, in this case, data products originating from *ALOS* satellite.

Anatomy of an add-on JAR file

Add-on JAR files, either one from the *DRB Cortex Extension* distribution or from Amalfi specific levels have a free content layout. However, the following layout is encouraged and followed whenever possible: `my-cortex-topic.jar`

```
|-> META-INF/            <- Standard Manifest folder
   |->cortex-index.owl   <- entry point for ontology description
|-> fr/gael/...          <- Java classes (no sub-folder)
...
|-> xsd/fr/gael/...      <- All other resources classified under
|-> owl/fr/gael/...      <- sub-folders denoting their types
...
```

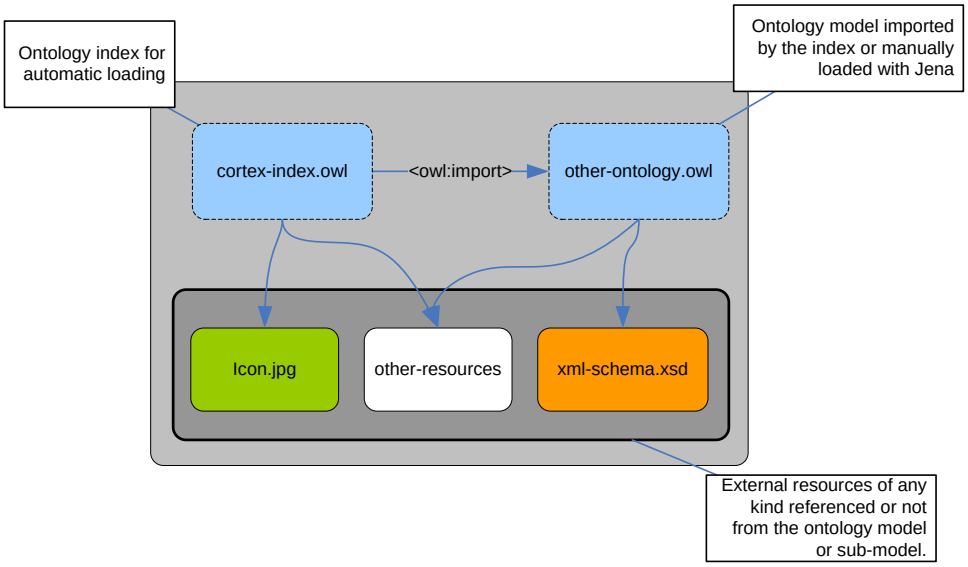


fig. 36 - Anatomy of a typical Add-on of DRB Cortex extension

DRB Cortex Extension is capable of recognizing automatically the *Addon*'s *Ontology Models* a soon as a corresponding `META-INF/cortex-index.owl` can be retrieved from the Java™ classpath i.e. generally embedded in a *JAR* archive but may also be included in a directory.

Classes of items

The pieces of *Ontology* defined in the *OWL* files of the *Topics* generally define a series of classes of information as for any *Ontology* but some have a specific meanings and lead to specific behaviours.

To make a long story short, let's take the following chunk of an *OWL* file as example:

```
<owl:Class rdf:about="&alos;prismLevel0Product">
  <rdfs:label xml:lang="en">
    ALOS/PRISM level 0 Product
  </rdfs:label>
  <rdfs:comment xml:lang="en">
    ALOS PRISM raw data generated by every down link segment and
    every band. This product is divided into an equivalent size to
```

```
    one scene.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="&alos;level0Product" />
  <drb:implementationIdentifier>file</drb:implementationIdentifier>
  <drb:signature rdf:parseType="Resource">
    <drb:xqueryTest>
      *[matches(name(),"AL_PSM_[0-9]{2}_(N|E|Q)_(R|N)_.*")]
    </drb:xqueryTest>
  </drb:signature>
</owl:Class>
```

Even if you are not used to *OWL*, *RDF* or even *XML* syntax, you should understand that this chunk of code deals with an *OWL* class identified *&alos;prismLevel0Product*. The *rdfs:label* and *rdfs:comment* markup inside the *owl:Class* markup, indicate that this class deals with information about an "ALOS/PRISM level 0 Product" i.e. data originating from *ALOS* satellite. The other markups may appear more cryptic but their meanings, as well as other possible markups not present in the example above, are summarized in the table below.

Markup	Cardinality	Description
`rdfs:label`	0..1	Human understandable title of the class: see [OWL] document for complete definition of this markup.
`rdfs:comment`	0..1	Textual information about the class: see [OWL] document for complete definition of this markup.
`rdfs:subClassOf`	*	Denotes the super-class of present class: see [OWL] document for complete definition of this markup and the description above for further information about how subclasses of `&drb:item` class behave.
`drb:implementation Identifier`	0..1	Forces the *DRB Implementation* (or *Data Model* binding) to be used with data matching the described type. See “Amalfi Data Model” p97 in appendices for more information about *Data Model* and *DRB Implementation*. Generally, the implementation to be used is automatically deduced by *DRB API®* but it is not always possible and this markup could resolve an ambiguity. Possible values for example, *file*, *xml* or *sdf*. Only subclasses of `&drb:item` class are affected by this markup.

Markup	Cardinality	Description
drb:signature	*	Provide a method for identifying if an actual item matches the present class. Several methods are available: they are all listed in the row below. Multiple methods can be provided, meaning that all of them shall be positive to match an item with the present class. Only subclasses of &drb:item class are affected by this markup.
Possible children of drb:signature markup (at least one of the following)		
drb:nameTest	*	This signature is positive if the item name is exactly the content of drb:nameTest markup.
drb:nameMatch	*	This signature is positive if the item name matches the regular expression[a] contained in the drb:nameMatch markup.
drb:namespaceTest	*	Same as drb:nameTest but applied to the namespace of the item.
drb:namespaceMatch	*	Same as drb:nameMatch but applied to the namespace of the item.
drb:xqueryTest	*	This signature is positive if the *Effective Boolean Value (EBV)* resulting from the evaluation of the *XQuery* expression contained in the drb:xqueryTest markup is true. The script is evaluated with the item as *Context Item*[b]. The example above shows a drb:xqueryTest signature that verifies that a given item has at least one child who's name matches a given regular expression i.e. "AL_PSM_[0-9]{2}_(N|E|Q)_(R|N)_.*"

table 24 - Main markups of a class of Item

[a] Please refer to http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html for a complete definition of the supported regular expression patterns.

[b] Please refer to [XQUERY] recommendation for further information about XQuery language and in particular the important concepts of *Effective Boolean Value* and *Context Item*.

The classes with the specific meanings mentioned above, are those classes deriving from the *&drb:item* class. They may be direct subclasses of this *&drb:item* class or derive from a unlimited number of subclasses that finally derive from this *&drb:item* class. All of these classes, later called *&drb:item* subclasses, could be considered as a bridge between the *Ontology Model* to the *Data Model*. This means that they will be automatically associated to the data to be inspected i.e. through the *drb:signature* markup described in the table above. Any *Ontology* resource related to these

subclasses is consequently automatically associated to the corresponding data: this is in particular the case of the inspection definitions described in the next section.

Service Inspection definitions

Inspections organisation can be described as the following schema:

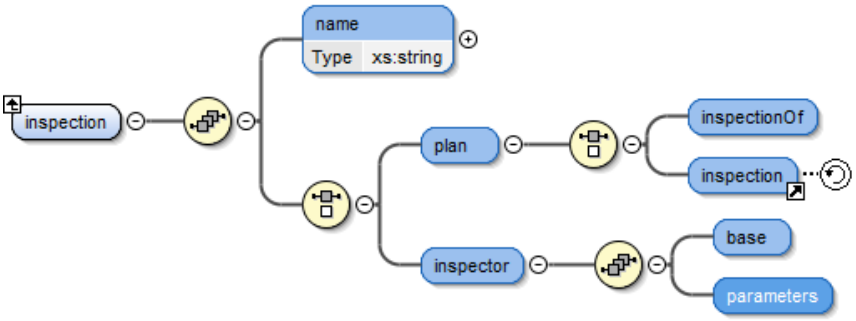


fig. 37 - Structure of Inspections.

The pieces of *Ontology* defined in the *OWL* files of the *Topics* may define simple inspection and inspection plans and may attach those latter to one or more classes of items of the Ontology. Again, let's take a chunk of an OWL file as example in lieu of a long theory:

```
<amalfi:inspection rdf:ID="sensingPeriod">
  <amalfi:name>MPH sensing time consistency</amalfi:name>
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;xqueryInspector"/>
      <amalfi:parameters rdf:parseType="Literal">

... truncated ...

      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

You should not be long to figure out that this example is specifying with an inspection identified as "sensingPeriod" and dealing with a so described "MPH sensing time consistency". It defines a simple inspection since it includes an `amalfi:name` and an `amalfi:inspector` markups as children. The `amalfi:inspector` markup accepts two children (indirectly bound through an `rdf:Description` markup): an `amalfi:base` that references the inspector to be applied as base algorithm (an XQuery inspector in this example), and an `amalfi:parameters` markup that configures the selected inspector. The content of this latter `amalfi:parameters` markup depends on the inspector.

Yet another chunk of code to illustrate how inspection plans are defined:

```
<amalfi:inspection rdf:ID="anExampleOfPlan">
  <amalfi:name>A data type inspection plan</amalfi:name>
  <amalfi:plan>
    <rdf:Description>
      <amalfi:sequence>
        <rdf:Description>
          <amalfi:inspection rdf:resource="&n1;sensingPeriod"/>
          <amalfi:inspection rdf:resource="&n1;discardedISP"/>
          <amalfi:inspection rdf:resource="&n1;sensingPeriod"/>
          <amalfi:inspection rdf:resource="&n1;reedSolomonCount"/>
          <amalfi:inspection rdf:resource="&n1;sensingPeriod"/>
        </rdf:Description>
      </amalfi:sequence>
       <amalfi:resultCompilation rdf:parseType="Literal">
        <amalfi:maxFailureCount>3</amalfi:maxFailureCount>
        <amalfi:maxFailureRatio>1</amalfi:maxFailureRatio>
        <amalfi:abortOnResult/>
      </amalfi:resultCompilation>
    </rdf:Description>
  </amalfi:plan>
</amalfi:inspection>
```

In this typical example of inspection plan, you should have recognized the same base description as for the previous simple inspection, and in particular that this plan is identified "*anExampleOfPlan*" and titled "A data type inspection plan". It defines an inspection plan since it includes an `amalfi:name` and an `amalfi:plan` markups as children. This `amalfi:plan` is composed of an `amalfi:sequence` that provides an ordered collection of references to inspection definitions of the *Ontology*, that can point to either simple inspections or inspection plans. This `amalfi:plan` is also composed of an optional `amalfi:resultCompilation` definition that specifies how the result shall be computed from the collection of results emanating from the inspections of the plan: the plan final result can actually be configured to fail when a given amount or ratio of failures have been identified while processing the inspections of the plan.

Built-in Inspectors

The previous section shows how to create an inspection. Inspection treatment is governed by a processor referenced in class `amalfi:base` and called inspector. Amalfi core implements a set of predefined inspectors:

- `amalfi:xqueryInspector`
- `amalfi:xmlSchemaInspector`

- amalfi:imageSaturationInspector
- amalfi:imageStripingInspector
- amalfi:userInspector

XQuery inspector

Description

"XQuery" is a general purpose inspector that interprets a functional script written in *XQuery 1.0* language.

The XQuery script is parsed and evaluated in the default environment using the inspected item as initial context. A `true` result denotes a *Passed* inspection and a `false` result denotes a *Failed* inspection.

The inspector manages XQuery results castable to a true/false value according to *Effective Boolean Value* (EBV) rules of the XQuery specifications.

XQuery errors raised during the parsing or evaluation phase are reported in the Amalfi result. The inspection process is aborted with the *Error* status and a diagnostic message.

Parameters

"XQuery" inspector accepts a single query script parameter and three optional messages for the inspection result. All parameters are formatted in XML.

• `amalfi:query` any XQuery script that return true if the inspection is passed and false if the inspection is failed. Other result types are converted to a boolean value using the standard EBV rule.

• `amalfi:message` the default message to be considered if none other failureMessage or successMessage has been provided. Otherwise, any of these other messages is preferred.

• `amalfi:successMessage` the message to be considered if the inspection passes . This message has a higher priority than the default message argument if both are provided.

• `amalfi:failureMessage` the message to be considered if the inspection fails. This message has a higher priority than the default message argument if both are provided.

All message parameters are XQuery scripts that return a result castable to a string value.

Usage

```
<amalfi:inspection rdf:ID="recordLength">
  <amalfi:name>Record Length</amalfi:name>
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;xqueryInspector"/>
      <amalfi:parameters rdf:parseType="Literal">
```

```
        <amalfi:query>record/length = 3060</amalfi:query>
        <amalfi:failureMessage>
          fn:concat("Bad record length ",
            xs:string(fn:data(record/length)),
            ": shall be equal to 3060 bytes.")
        </amalfi:failureMessage>
        <amalfi:successMessage>
          "Correct record length of 3060 bytes."
        </amalfi:successMessage>
      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

XML-Schema validator

Description

"XML-Schema" inspector validates the content of a structured item using an *XML-Schema* definition document (XSD). The XSD processor reports all elements with an invalid cardinality, value type or range.

Additionally, the XSD inspector accepts an optional XQuery script to restrict the validation on a user-defined selection.

Parameters

The format expected for the "XML-Schema" inspector parameters is an XML fragment composed of an unordered series of elements:

- `amalfi:query` an optional XQuery script that select the list of items to be validated. These items must be descendant of the inspection root item.

- `amalfi:failureLimit` the maximum number of schema failure results to be considered. The inspector will not report more errors than the specified limit.

- `amalfi:maxDepth` the maximum depth allowed during the schema validation. By convention, the root element of the target schema is located at depth 0. All descendant nodes at a depth strictly greater than the specified limit will be ignored.

Usage

Hereunder, the root element is named product and is located at depth 0. The specified parameters validate the mph and sph elements.

```
<amalfi:inspection rdf:ID="mphSphValidity">
  <amalfi:name>MPH/SPH Validity</amalfi:name>
```

```
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;xmlSchemaInspector"/>
      <amalfi:parameters rdf:parseType="Literal">
        <amalfi:query>product/(mph|sph)</amalfi:query>
        <amalfi:failureLimit> 10 </amalfi:failureLimit>
        <amalfi:maxDepth> 2 </amalfi:maxDepth>
        <amalfi:abortOnResult> true </amalfi:abortOnResult>
      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

Image saturation

Description

"Image Saturation" inspector checks that the saturation ratio of an input image is within the allowed range.

Parameters

The format expected for the "Image Saturation" inspector parameters is an XML fragment composed of an unordered series of elements:

- `amalfi:background` an optional value flag to exclude pixels outside the image area. If not specified all pixels are inspected.

- `amalfi:minThreshold` minimum allowed pixel value (limit for low saturation). Low saturation cannot be detected if this threshold is equal to the lower bound of the pixel range (0 for unsigned integer pixel values).

- `amalfi:maxThreshold` maximum allowed pixel value (limit for high saturation). High saturation cannot be detected if this threshold is equal to the higher bound of the pixel range (255 for 8 bits unsigned).

- `amalfi:maxLowSaturation` the maximum low saturation ratio allowed (in per thousand).

- `amalfi:maxHighSaturation` the maximum high saturation ratio allowed (in per thousand).

Usage

The configuration below may be used to detect saturated pixels in 8-bit image file such as Landsat products. The value zero is reserved for background pixels and the allowed pixel range is within 1 to

254 (inclusive). Pixel values lower or greater than these limits are saturated. For example, a pixel having value 255 is highly saturated.

```
<amalfi:inspection rdf:ID="imageSaturation">
  <amalfi:name>Image Saturation</amalfi:name>
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;imageSaturationInspector"/>
      <amalfi:parameters rdf:parseType="Literal">
        <background>0</background>
        <minThreshold>2</minThreshold>
        <maxThreshold>254</maxThreshold>
        <maxHighSaturation>100</maxHighSaturation> <!-- 10% -->
        <maxLowSaturation>1</maxLowSaturation> <!-- 0.1% -->
      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

Image striping inspector

Description

"Image Striping" inspector checks that an input image is not corrupted by Striping noise. This inspection is relevant for imagery items acquired by push-broom sensors such a Landsat TM/ETM+ satellite instruments.

The Striping inspection is based on a mathematical process called *Fourier Transform* that detects spectral artifacts in all bands of the image. It reports rectangular area (or window) with a Striping magnitude higher than the limit.

Parameters

The format expected for the "Image Striping" inspector parameters is an XML fragment composed of an unordered series of elements:

- `amalfi:minStandardDeviation` the minimum standard deviation required for each band. Otherwise the Striping is not checked and the inspection is passed.

- `amalfi:minPeriod` the minimum acceptable period of the Striping noise. Otherwise the inspection is passed but some comments will appear in the inspection report.

- `amalfi:maxStriping` the maximum striping magnitude allowed per band. Use the value 1.8 to detect strong striping in 8-bits images.

• `amalfi:windowDensity` for optimization. The number of inspection window along lines. It is defaulted to 3.

• `amalfi:windowWidth` for expert only. An optional window width for the local Fourier analysis. It is defaulted to 256 and must be a power of 2 (e.g. 128, 256, 512).

• `amalfi:windowHeight` for expert only. An optional window height for the local Fourier analysis. It is defaulted to 256 and must be a power of 2 (e.g. 128, 256, 512).

Usage

This sample configuration is optimized to detect Striping noise in 8-bits images with at least 3 of standard deviation. The inspection fails if the striping magnitude exceeds 1.8. It performs the analysis inside 3 local windows per line and ignores striping noise with a period greater than 32 pixels. The window size is defaulted to 256 pixels.

```
<amalfi:inspection rdf:ID="imageStriping">
  <amalfi:name>Image Striping</amalfi:name>
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;imageStripingInspector"/>
      <amalfi:parameters rdf:parseType="Literal">
         <amalfi:minStandardDeviation>3
         </amalfi:minStandardDeviation>
         <amalfi:windowDensity>3</amalfi:windowDensity>
         <amalfi:minPeriod>32</amalfi:minPeriod>
        <amalfi:maxStriping>1.8</amalfi:maxStriping>
      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

User inspector

Description

The "User" inspector is an interactive process that asks a predefined question to the user and requests a passed/failed status.

Parameters

The "User" inspector accepts a single string parameter that describes the passed/failed criteria of the user inspection. Be careful to avoid any ambiguity.

Usage

A typical use of the "User Inspector" is the systematic visual inspection of optical and SAR products to check for well-known artefacts that cannot be detected with an automatic procedure.

```
<amalfi:inspection rdf:ID="missingBands">
  <amalfi:name>Missing Bands</amalfi:name>
  <amalfi:inspector>
    <rdf:Description>
      <amalfi:base rdf:resource="&amalfi;userInspector"/>
      <amalfi:parameters rdf:parseType="Literal">
        Checks that all bands are available and contain consistent
        data.
      </amalfi:parameters>
    </rdf:Description>
  </amalfi:inspector>
</amalfi:inspection>
```

Adding or modifying an Add-on

Adding an add-on consists in making available a META-INF/cortex-index.owl file from the Java™ classpath. Various methods are possible for setting up this configuration but the easiest means is probably to create a directory that includes a META-INF subdirectory which contains a cortex-index.owl file and adding the created top level directory to the CLASSPATH environment variable. Configuring Amalfi add-on resources in the Amalfi configuration file also uses this mechanism, and support jar as well as directories.

Having a directory form of an add-on may be a good tactic while for development phases or for occasional uses, but sealing the result in a JAR file, should in turn allow a more accurate configuration control and minimize the risk of hazardous editing.

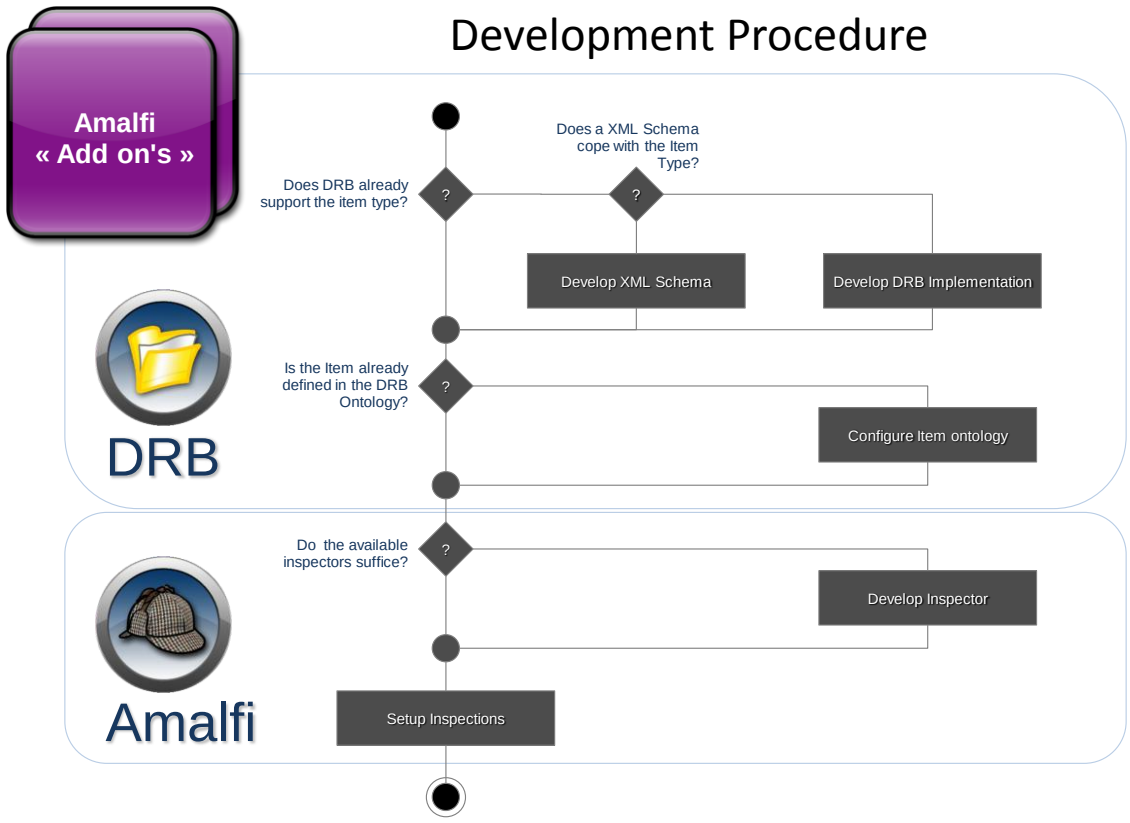
If you are beginning, it is highly recommended to start from a copy of one of the distributed add-on and, a priori, one that already implement similar goals to the one you main at. It is however, highly discouraged to update the distributed content and JAR files, since you will quickly reach difficulties to discriminate the files that are part of the original distribution from the others: this may lead to a loss of your production and could extensively increase any support activity while figuring out your software configuration status. Extra add-ons and topics should be placed outside the installation directory.

The same notice should be followed if you only need to update an existing add-on but with the fact that, in addition, you should disable the original one. Disabling one of the distributed Topics can be easily done by removing the corresponding JAR file (recommended only if you have kept the installation package to potentially restore the original files) or by renaming them, for example with an extension .original. Missing the disabling of original Topic will lead to duplicate definitions,

generating warning log entries and it may not be easy to distinguish which of the duplicates that are actually used by the software.

Use Case: Creating a new add-on for Amalfi.

The objective of this section is to describe by the example, how to create and configure a new Amalfi add-on. This example focuses on a Landsat CEOS binary file as the item to be controlled in Amalfi. Different steps to successfully reach this objective consist in following this procedure:



In the scope of this use case, we consider the following answers:

Question	Answer	Action to be performed
Does Drb already support the item type?	No	
Does a XML schema cope with the item type	Yes	Develop XML Schema
Is the Item already defined in a Drb Ontology?	No	Configure item ontology
Do the Available inspectors suffice?	No*	Develop inspector
		Setup Inspections

table 25 - Creating add-on procedure

(*)Note that we consider having to develop a specific inspector for the need of this demonstration, but, it shall be necessary first to check the existing built-in inspector (See "Built-in Inspectors" p73) before starting such an implementation.

Develop XML Schema

The focus example is a Landsat binary file. Drb (http://www.gael.fr/drb) offers the entire material to manage such a binary dataset.

To setup DRB-based XML schema, SDF breaks down any binary file into a tree of nodes. This XML Schema has few additional markups providing the physical description of the binary file. The additional main markups specified by SDF are:

SDF Markup	Markup description	
	Attribute	**Attribute description**
`sdf:block`	Block of data in the binary file.	
`sdf:offset`	Offset of the block from the previous sibling block.	
	`unit`	Toggles bit or byte unit. Default value is byte.
`sdf:length`	Physical length of the block.	
	`unit`	Toggles bit or byte unit. Default value is byte.
	`query`	An XQuery specifying how to compute the length value. The query can be as complex as necessary.
`sdf:occurrence`	Number of occurrences of the block.	
	`query`	An XQuery specifying how to compute the occurrence value. The query can be as complex as necessary.
`sdf:encoding`	Specify whether the block is encoded in ASCII, BINARY or EBCDIC. The default value is BINARY.	
`sdf:padding`	Padding (i.e. margin) to be considered before or after the block.	

SDF Markup	Markup description	
	Attribute	Attribute description
	type	Specifies the type of padding. The type can be "header" corresponding to a padding beginning the block. On the contrary the block type is "footer". This attribute has no default value.
	unit	Toggles bit or byte unit. Default value is byte.
sdf:byteOrder	Specify whether the block byte order is big-endian (MSB) or little-endian (LSB). Default is MSB.	

table 26 - Sdf markups

In order to reduce the code size of the XML schema presented here, we only focus on the ceos header. Example of sdf-schema describing CEOS header (i.e. *ceos-header.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:sdf="http://www.gael.fr/2004/12/drb/sdf"
   targetNamespace="http://www.gael.fr/schemas/ceos"
   xmlns:ceos="http://www.gael.fr/schemas/ceos"
   elementFormDefault="qualified">

<xs:element name="ceosFile">
<xs:complexType name="ceosHeader">
      <xs:annotation>
         <xs:documentation xml:lang="en">
            The CEOS Superstructure description for CCT-CCB-002
            Dump version : "0.01 - 02.12.2002"
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="recordSequenceNumber" type="xs:unsignedInt">
            <xs:annotation>
               <xs:documentation xml:lang="en">
                  Record number
               </xs:documentation>
               <xs:appinfo>
                  <sdf:block>
                     <sdf:encoding>BINARY</sdf:encoding>
```

```
            <sdf:length>4</sdf:length>
         </sdf:block>
      </xs:appinfo>
   </xs:annotation>
</xs:element>

<xs:element name="firstRecordSubTypeCode" type="xs:unsignedByte">
   <xs:annotation>
      <xs:documentation xml:lang="en">
         1st record subtype code
      </xs:documentation>
      <xs:appinfo>
         <sdf:block>
            <sdf:encoding>BINARY</sdf:encoding>
            <sdf:length>1</sdf:length>
         </sdf:block>
      </xs:appinfo>
   </xs:annotation>
</xs:element>

<xs:element name="recordTypeCode" type="xs:unsignedByte">
   <xs:annotation>
      <xs:documentation xml:lang="en">
         Record type code
      </xs:documentation>
      <xs:appinfo>
         <sdf:block>
            <sdf:encoding>BINARY</sdf:encoding>
            <sdf:length>1</sdf:length>
         </sdf:block>
      </xs:appinfo>
   </xs:annotation>
</xs:element>

<xs:element name="secondRecordSubTypeCode" type="xs:unsignedByte">
   <xs:annotation>
      <xs:documentation xml:lang="en">
         2nd record sub-type code
      </xs:documentation>
      <xs:appinfo>
         <sdf:block>
            <sdf:encoding>BINARY</sdf:encoding>
            <sdf:length>1</sdf:length>
         </sdf:block>
      </xs:appinfo>
   </xs:annotation>
```

```
         </xs:element>

         <xs:element name="thirdRecordSubTypeCode" type="xs:unsignedByte">
            <xs:annotation>
               <xs:documentation xml:lang="en">
                  3rd record sub-type code
               </xs:documentation>
               <xs:appinfo>
                  <sdf:block>
                     <sdf:encoding>BINARY</sdf:encoding>
                     <sdf:length>1</sdf:length>
                  </sdf:block>
               </xs:appinfo>
            </xs:annotation>
         </xs:element>

         <xs:element name="length" type="xs:unsignedInt">
            <xs:annotation>
               <xs:documentation xml:lang="en">
                  Length of this record
               </xs:documentation>
               <xs:appinfo>
                  <sdf:block>
                     <sdf:encoding>BINARY</sdf:encoding>
                     <sdf:length>4</sdf:length>
                  </sdf:block>
               </xs:appinfo>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Once the XML schema is defined, it is necessary to provide DRB with this schema in order to properly decode and retrieve the dataset structure. Simplest way to do it consists in manually using the drb embed XQuery/XPath scripting:

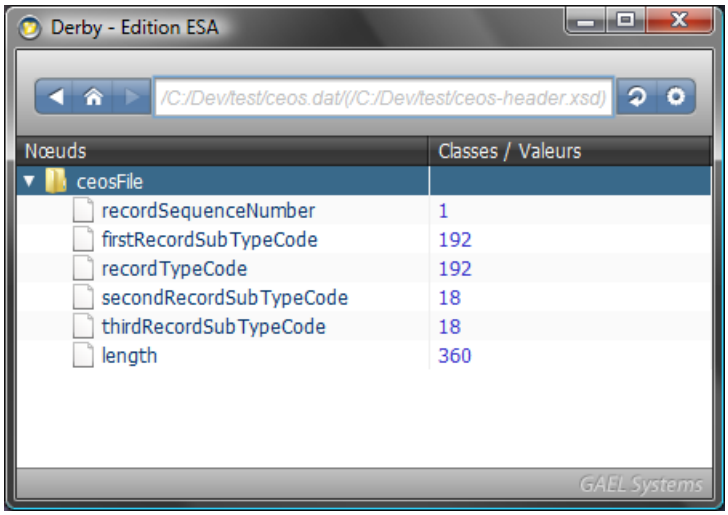
```
 > java -jar drb-2-3-release.jar \
      -s '/C/Dev/test/ceos.dat/(/C/Dev/test/ceos-header.xsd)*'

<ceosFile xmlns="http://www.gael.fr/schemas/ceos">
   <recordSequenceNumber>1</recordSequenceNumber>
   <firstRecordSubTypeCode>192</firstRecordSubTypeCode>
   <recordTypeCode>192</recordTypeCode>
   <secondRecordSubTypeCode>18</secondRecordSubTypeCode>
   <thirdRecordSubTypeCode>18</thirdRecordSubTypeCode>
```

```
   <length>360</length>
</ceosFile>
```

In this execution, we ensure that Drb properly applied the schema on binary data with showing its human readable XML representation.

Another way of testing this schema consists in using Derby Application (http://www.gael.fr/derby). Also based on DRB API®, it graphically uses sdf schema to decode data and display a browsable tree representation of this data. Following snapshot of Derby shows a ceos data decoded by the schema presented here before:



Note that command line might be preferred to the Derby UI application when testing schema, because the command line produces more verbose information than Derby.

Configure item Ontology

Once XML schema configured (see sections before), it is possible to create the ontology file that allow DRB API® automatically assign the schema to the data. According to our ceos example the ontology looks like:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY drb "http://www.gael.fr/drb#">
  <!ENTITY ceos "http://www.ceog.org/ccb/cct-002#">
]>
<rdf:RDF xmlns:owl="&owl;" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;"
   xmlns:drb="&drb;" xmlns="&ceos;">

   <owl:Ontology rdf:about="&ceos;" />
```

```
   <owl:Class rdf:about="&ceos;ceosHeaderOnly">
      <rdfs:label xml:lang="en">CEOS File</rdfs:label>
      <rdfs:subClassOf rdf:resource="&drb;item" />
      <drb:signature rdf:parseType="Resource">
         <drb:nameMatch>.*\.(?i)dat</drb:nameMatch>
      </drb:signature>
      <drb:implementationIdentifier>sdf</drb:implementationIdentifier>
      <drb:schemaLocation>ceos-header.xsd</drb:schemaLocation>
   </owl:Class>
```

This ontology description implements a new class called *http://www.ceog.org/ccb/cct-002#ceosHeaderOnly* subclass of *http://www.gael.fr/drb#item* that defines DRB properties.

It defines a signature that will be used by the drb resolver to link the data to its schema. Full syntax of this signature is defined in table "Main markups of a class of Item" p.71.

To be taken into account, this ontology file shall be placed in the following path:

```
META-INF/cortex-index.owl
```

and referenced into the drb classpath.

Following command sequence demonstrate that the ontology was used to resolve the data (query string does not use the schema as in the previous section):

```
> tree .
.
|-- drb-2-3-release.jar
|-- log4j-1.2.8.jar
|-- ceos.dat
|-- META-INF/
|   |-- cortex-index.owl
> setenv CLASSPATH .
> java \
   -Dfr.gael.drb.meta.DrbMetadataResolver=fr.gael.drbx.cortex.DrbCortexMetadataResolver \
   -jar drb-2-3-release.jar \
   -s 'ceos.dat'

<ceosFile xmlns="http://www.gael.fr/schemas/ceos">
   <recordSequenceNumber>1</recordSequenceNumber>
   <firstRecordSubTypeCode>192</firstRecordSubTypeCode>
   <recordTypeCode>192</recordTypeCode>
   <secondRecordSubTypeCode>18</secondRecordSubTypeCode>
   <thirdRecordSubTypeCode>18</thirdRecordSubTypeCode>
   <length>360</length>
</ceosFile>
```

Develop inspector

Developing an inspector consists in implementing the java interface "`fr.gael.amalfi.inspection.Inspector`", and declares this new inspector in Amalfi.

Implements the inspector interface

The following sample code show a basic example of an inspector. The inspector interface is defined in the amalfi-core-XX.jar file, and shall be added as dependency for the compilation.

```
package fr.gael.amalfi.core.inspection;

import fr.gael.amalfi.core.inspection.DefaultResult;
import fr.gael.amalfi.core.inspection.Result;
import fr.gael.amalfi.core.inspection.ResultStatus;
import fr.gael.amalfi.core.inspection.Inspector;
import fr.gael.drb.DrbItem;
import fr.gael.drb.DrbSequence;
import fr.gael.drb.query.Query;
/**
 * Sample of basic inspector controlling a ceos record
 * information.
 */
public class MyInspector implements Inspector
{
   /**
    * The identifier of the inspector. This property is supposed to be a read
    * only value and unique across the inspector implementations.
    *
    * @return the identifier of this inspector.
    */
   public String getIdentifier()
   {
   /* The inspector identifier is voluntary defined as a ontology
      class name because it will be declared as is in Amalfi.
   */
      return ("http://www.gael.fr/amalfi#myInspector");
   }
   /**
    * The name of the inspector.
    *
    * @return the name of this inspector.
    */
   public String getName()
   {
      return ("My Sample Inspector");   /**
    * Configure the inspector parameters.
```

```
 *
 * @param parameters the parameters as a string.
 * @throws NullPointerException if the provided parameter string is null.
 */
public void setParameters(String parameters)
{
   // No parameters
}
/**
 * Inspects the item according to the latest set parameters.
 *
 * @param item the item to be inspected.
 * @return the result of the inspection.
 */
public Result inspect(DrbItem item)
{
   // Prepare result
   DefaultResult result = new DefaultResult(ResultStatus.UNDEFINED, null);

   // Evaluate the XQuery with the given item as context item
   DrbSequence sequence = null;
   try
   {
      String s_query = new String("data(ceosFile/firstRecordSubTypeCode)=192");
      Query query = new Query(s_query);
      sequence = query.evaluate(item);
      if ((sequence == null) ||(sequence.getLength() == 0))
         throw new NullPointerException ("Query \"" +
            s_query + "\" returns empty result.");
   }
   catch (Exception exception)
   {
      result.setStatus(ResultStatus.ERROR);
      result.setMessage("Error while evaluating XQuery: '"
            + exception.getMessage() + "'");
      logger.debug("Error while evaluating XQuery.", exception);
      return result;
   }

   // Get effective boolean value of the resulting sequence
   boolean is_passed = Query.getEffectiveBooleanValue(sequence);

   // Determine result status from the EBV
   if (is_passed)
   {
      result.setStatus(ResultStatus.PASSED);
```

```
            // Build result message
            result.setMessage("Inspection successfully passed");

        }
        else
        {
            result.setStatus(ResultStatus.FAILED);
            result.setMessage("Inspection failed.");
        }
        // Return result
        return result;
    }
    /**
     * Retrieves working directory dedicated to this inspector
     */
    public String getWorkingDirectory () {return null;}

} // Inspector
```

Declare the inspector in Amalfi

Once implemented, the inspector can be declared into Amalfi applications with created a `META-INF/cortex-index.owl` file, placed in the Amalfi classpath, or declared as an add-on in the Amalfi configuration file. This owl file shall contain the following declarations:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY drb "http://www.gael.fr/drb#">
  <!ENTITY amalfi "http://www.gael.fr/amalfi#">
]>

<rdf:RDF xmlns:owl="&owl;" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;"
    xmlns:drb="&drb;" xmlns:amalfi="&amalfi;"
    xml:base="http://www.gael.fr/amalfi"
>
    <!-- Inspector Individuals -->
    <amalfi:inspector rdf:ID="myInspector">
        <amalfi:name>Custom Inspector</amalfi:name>
        <amalfi:class>
            fr.gael.amalfi.inspector.MyInspector
        </amalfi:class>
    </amalfi:inspector>
```

Note that we identified the inspector "`http://www.gael.fr/amalfi#myInspector`" in the inspector source code. This corresponds to the owl class name aggregating with the defined "`xml:base`".

Setup Inspection

Inspections declaration aggregates the existing owl classes defined for the drb item (See Configure item Ontology section just before). In the ceos example, the inspection can be defined as follow:

```
<rdf:Description rdf:about="&ceos;ceosHeaderOnly">
   <amalfi:inspection rdf:resource="&ceos;plan"/>
</rdf:Description>
```

In this example, the amalfi inspection called "*&ceos;plan*" is aggregated to the "*&ceos;ceosHeaderOnly*" drb item class previously defined in our example. Once connected to the item, the plan can be developed as follow:

```
<amalfi:inspection rdf:ID="plan">
      <amalfi:name>Ceos Header Plan</amalfi:name>
      <amalfi:plan>
         <rdf:Description>
            <amalfi:sequence>
               <rdf:Description>
                  <amalfi:inspection rdf:resource="&ceos;myInspectorTest"/>
                  <amalfi:inspection rdf:resource="&ceos;xsdValidator"/>
                  <amalfi:inspection rdf:resource="&ceos;xQueryTest"/>
               </rdf:Description>
            </amalfi:sequence>
         </rdf:Description>
      </amalfi:plan>
   </amalfi:inspection>
```

This plan declares a set of sub-inspection:

"&ceos;myInspectorTest"

```
   <amalfi:inspection rdf:ID="myInspectorTest">
      <amalfi:name>Test of MyInspector</amalfi:name>
      <amalfi:inspector>
         <rdf:Description>
            <amalfi:base rdf:resource="&amalfi;myInspector"/>
            <amalfi:parameters rdf:parseType="Literal"/>
         </rdf:Description>
      </amalfi:inspector>
   </amalfi:inspection>
```

"&ceos;xsdValidator"

```
<amalfi:inspection rdf:ID="xsdValidator">
      <amalfi:name>XML Schema Validation</amalfi:name>
      <amalfi:inspector>
         <rdf:Description>
            <amalfi:base rdf:resource="&amalfi;xmlSchemaInspector"/>
            <amalfi:parameters rdf:parseType="Literal">
               <amalfi:query>.</amalfi:query>
               <amalfi:failureLimit>100</amalfi:failureLimit>
            </amalfi:parameters>
         </rdf:Description>
      </amalfi:inspector>
   </amalfi:inspection>
```

"&ceos;xQueryTest"

```
   <amalfi:inspection rdf:ID="xQueryTest">
      <amalfi:name>inspection using XQuery inspector</amalfi:name>
      <amalfi:inspector>
         <rdf:Description>
            <amalfi:base rdf:resource="&amalfi;xqueryInspector"/>
            <amalfi:parameters rdf:parseType="Literal">
               <amalfi:query>
                  <![CDATA[
                  data(ceosFile/firstRecordSubTypeCode)=192
                  ]]>
               </amalfi:query>
               <amalfi:successMessage>
                  concat (name (ceosFile/firstRecordSubTypeCode),
                          " is ",
                          data (ceosFile/firstRecordSubTypeCode))
               </amalfi:successMessage>
               <amalfi:failureMessage>
                  concat (name (ceosFile/firstRecordSubTypeCode),
                          " is ",
                          data (ceosFile/firstRecordSubTypeCode),
                          " expected was 192")
               </amalfi:failureMessage>
            </amalfi:parameters>
         </rdf:Description>
      </amalfi:inspector>
   </amalfi:inspection>
```

What else?

Testing Amalfi Add-ons

Addons are load by amalfi via its configuration file (See "Add-on resolution with jar method" p.63) or by amalfi server with "—addon" switch (See "Server inspection loading" p. 39). Add-ons can be provided as directory or java archive (JAR file) of this directory.

Minimalist expected directory structure shall be:

```
ROOT/
   |- META-INF
      |- cortex-index.owl
```

Where `cortex-index.owl` file contains the Amalfi add-ons features described in this section.

Amalfi server will be able to loas this add-on as follow:

```
java -Xmx512m -jar amalfi-server-xx.jar --addon /path/to/ROOT
```

Or the Amalfi configuration file may looks like (usefull to run Compass os server without parameters):

```
<!-- Ontology currently editing/debuging -->
<resource xsi:type="OntologyResource" id="ceos" type="jar">
   <name>Amalfi ceos testing Ontology resource</name>
   <description>
      The Ontologies currently in test.
   </description>
   <url>http://www.gael.fr/drb#ceos</url>
   <configuration><![CDATA[
      <jar name="ceos addon" followInternalDependencies="false"
         xmlns="http://www.gael.fr/classpath/resolver/jar/configuration">
         <path>
            file:/path/to/ROOT/
         </path>
      </jar>
   ]]></configuration>
 </resource>
```

Amalfi add-ons deployment management

Before starting this chapter, it is required to download and install the following softwares:

- Maven 2 (http://maven.apache.org)
- Nexus (http://nexus.sonatype.org)

Note about configuration management: it is recommended to use configuration management software (i.e. svn, cvs, continuus…) to manage code evolution and versioning, and keep aligned nexus deployement with these versions. So Nexus can be configured to not allow redeploying a same version in its releases reporsitory.

Maven is used to create add-on as jar files and deploy them into nexus repository. The following maven Projet Object Model (POM file) gather necessary configuration to produce jar add-on and deploy it throught a remote repository:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<project
   Xmlns    = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://maven.apache.org/POM/4.0.0
       http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>int.esa.amalfi</groupId>
   <artifactId>custom-addon</artifactId>
   <packaging>jar</packaging>
   <version>1.0.0</version>
   <name>Amalfi sample addon</name>
   <distributionManagement>
      <repository>
         <uniqueVersion>true</uniqueVersion>
         <id>amalfi-addons </id>
         <name>Amalfi Add-ons</name>
         <url>http://server:8081/nexus/content/repositories/releases </url>
         <layout>default</layout>
      </repository>
   </distributionManagement>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <configuration>
               <archive>
                  <manifest>
                     <addClasspath>true</addClasspath>
                  </manifest>
               </archive>
            </configuration>
         </plugin>
      </plugins>
```

```
   </build>
</project>
```

To run maven :

```
maven clean package deploy
```

Once deployed, Amalfi Ontology resource shall be configure to remotely access this addon using "maven" mechanism as described in chapter "Add-on resolution with maven" p.65.

Deploying Amalfi over a Network

The architecture of Amalfi module simplifies the capability to deploy and inter-connects its modules:

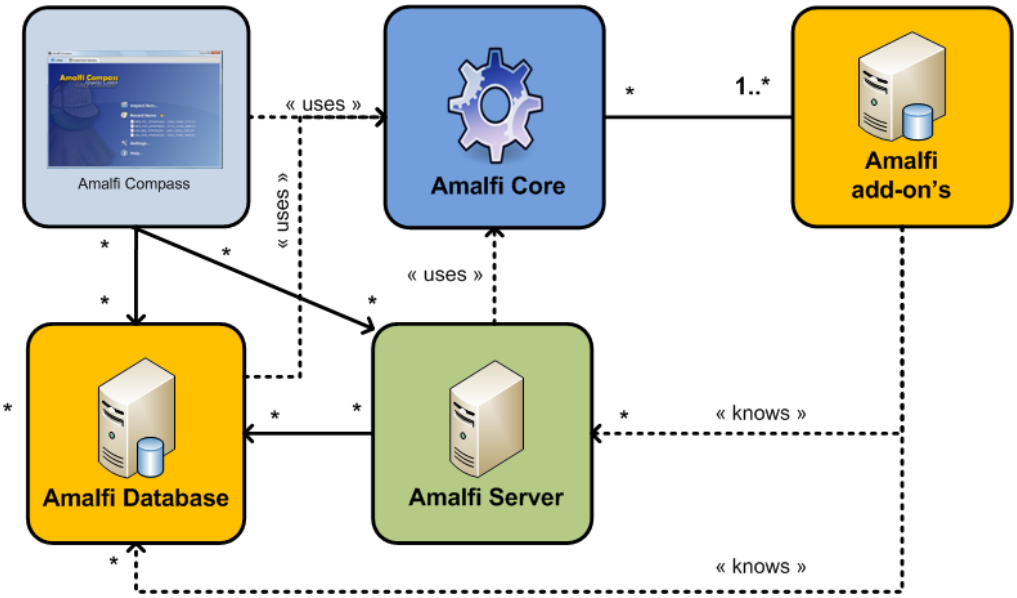


fig. 38 - Amalfi static deployment model

Each module can be separately deployed over a network as followed:

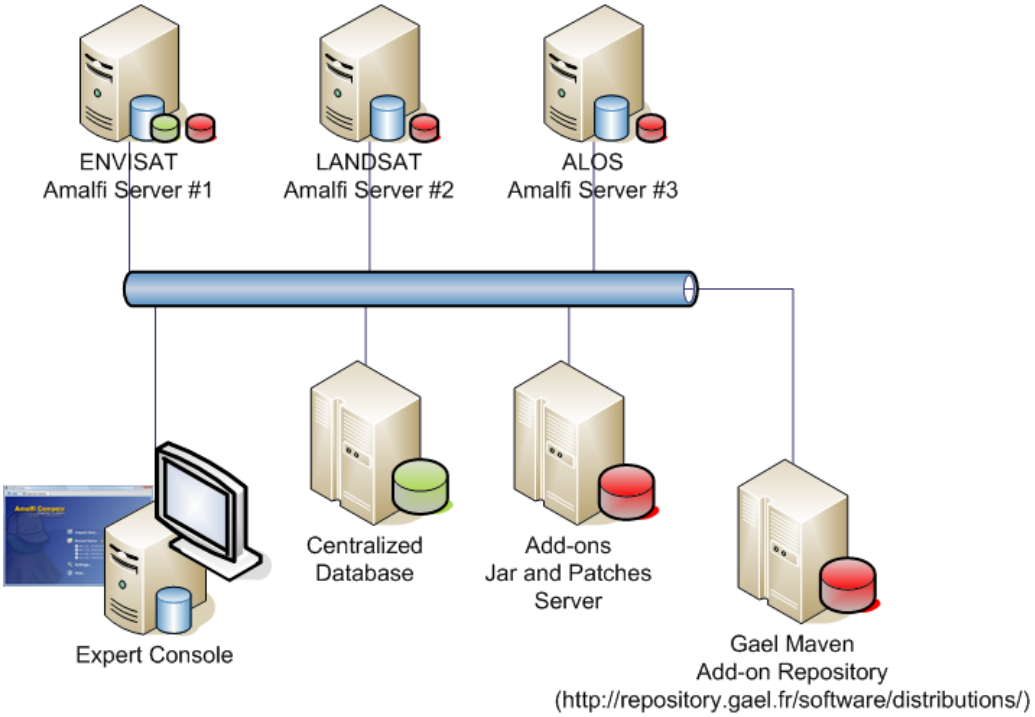


fig. 39 - Amalfi physical deployment sample

In this case, called operational, a set of servers are dedicated to the data processing, a database server centralizes inspections results via Database synchronisation, and other servers are used to manage add-ons. An expert console connected to servers monitors inspection queues.

This configuration requires allowing networking opened connections:

FROM\TO	Server #1	Server #2	Server #3	Centralized database	Jar/patches Add-on server	Expert console	Internet connection
Server #1	-	-	-	3306	-	1190	-
Server #2	-	-	-	3306	-	1190	-
Server #3	-	-	-	3306	-	1190	-
Centralized database	3306	3306	3306	-	-	3306	-
Jar/patches Add-on server	80	80	80	-	-	22/80	-
Expert console	-	-	-	-	-	-	-
Internet connection	443/80/22	443/80/22	443/80/22	-	-	443/80/22	

table 27 - Operational deployment networking needs

The ports reported here before are used for following services:

Port name	Port number
ssh/scp	22
http	80
https	443
RMI	1099
mysql	3306

table 28 - Port services

Appendix – Understanding Background Concepts

Amalfi Data Model

As previously introduced, Amalfi endeavors to be as independent as possible from the data type it can inspect. This behavior intends to make Amalfi relevant for the maximum number of environments and contexts of use, and avoid constraining it to a set of data types, which are in force at a specific period of time or into a specific domain.

To achieve this challenge, a clue element of the architectural design of Amalfi, is a federation of all processing components around a unified *Data Model*. Standing as an input/output interface between those processing components, the support of a new data type boils down to configure or implement a new binding of the *Data Model*. Indeed, the support of a new textural or binary format should not cripple the system, not require reworking the architectural design, and on the contrary, should make the existing features transparently benefiting from the novelty, or at least let them identify, without failure, that they have nothing to deal with it.

Another critical aspect of the architecture is that the *Data Model* has a very low level of semantic, to allow bindings of data of very different natures (i.e. metadata, images…).

The hereby purpose is to provide you with an overview of the main characteristics and terms that cross over the overall functionality of the software for a better understanding of graphical or command line interfaces of Amalfi.

The Amalfi *Data Model* considers that any data can be bound to a *Sequence* of *Items*, where item are of different nature. The relationships are depicted in following figure:

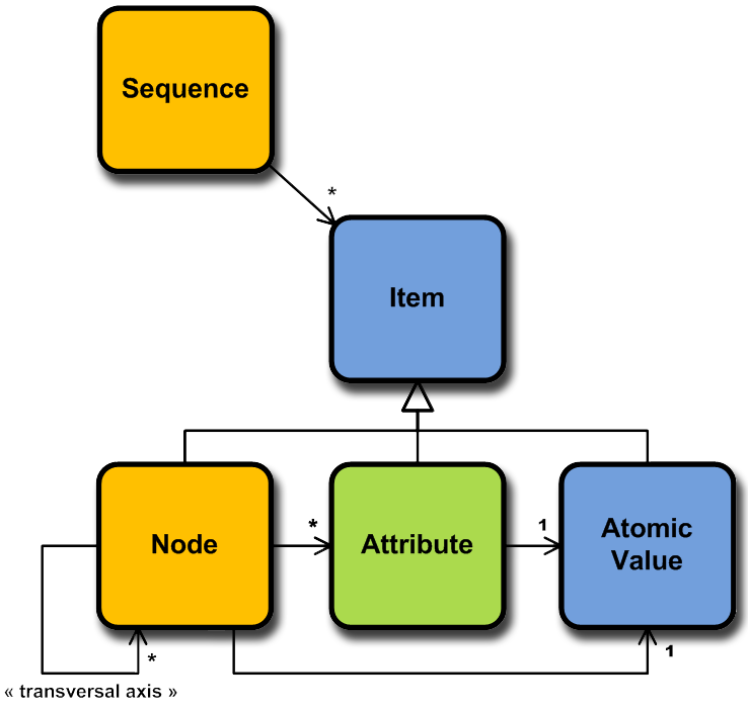


fig. 40 - Amalfi static data model

The *Sequence* can be the *Empty Sequence* if it contains nothing, can be a *Singleton* if it contains one *Item* only, or a collection of an unlimited number of *Items*. The term *Item* is an abstract concept that denotes a named element that can indifferently be specialized to a *Node*, an *Attribute* or an *Atomic Value*. These three specializations have different meanings and different types of relationships in the *Data Model*.

The *Nodes* intend to model the hierarchical structure (tree) of the bounded data. In modern windowing environments, the directory list is an excellent example of a tree of *Nodes*. The top of the tree components is the root directory or drive, and under that is a list of subdirectories. If the subdirectories contain further subdirectories, they are bounded to *Nodes* as well. The actual files found in any directory in this type of component are the leaves of the tree, also modeled as *Nodes*. Any data that contains parent-child relationships between chunks of information can be modeled as a tree of *Nodes*. Another common example is an organizational chart. In such a chart, every management position is a *Node*, with child *Nodes* representing the employees under the manager. The leaves of this organizational chart are the employees who are not in management position, and its root is the top level manager. Of course, real organizations don't always adhere to a strict tree structure; this example highlights one limitation of the current Amalfi *Data Model* that has voluntarily been selected to maintain an acceptable level of complexity with regard to the actual need of cyclic data modeling. Should the express need of cyclic modeling arise, the traversal axes of the *Data Model* should be augmented with another type of *Node* relationship, which should not impact the previously implemented bindings. The *Node* relationships currently available are illustrated in the figure below and summarized in the following table.

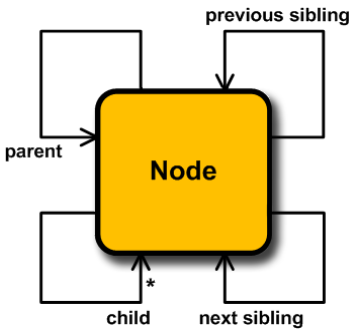


fig. 41 - Node traversal axes

Node Traversal Axis	Summary
Parent	The parent *Node* if the current *Node* is not the root *Node*
Child	The collection of child *Nodes*, if any.
Previous Sibling	The previous sibling if the current *Node* is not the first child of its parent *Node*
Next Sibling	The next sibling if the current *Node* is not the last child of its parent *Node*

table 29 - Summary of Node traversal axes

In the *Data Model*, *Nodes* can have attached *Attributes* that could be considered as *Metadata* of *Nodes* Typical *Attributes* could be the size of a file modeled as a *Node*, the *XML name* attribute of the `<person name="Smith"/>` *XML* fragment or the compression ratio of *ZIP* file entry.

Because *Nodes* and *Attributes* are not only symbolic notions about the modeled data, they can be assigned a given *Atomic Value* that actually binds a part of the data to a primitive type as integers, floating point numbers or strings of characters. The *Atomic Values* may also represent an array of primitive types.

Amalfi benefits from and reuses the Open Source *Data Request Broker - DRB API®* as its core component for data access. The version of *DRB API®* made available in the standard distribution of Amalfi provides the following *Data Model* bindings also called *DRB Implementations*.

DRB Implementation	Summary
File	The *File* implementation wraps around the local file system to bind directories and files as *Nodes* of the *Data Model*.
XML	The *XML* implementation is similar to the *Document Object Model (DOM)* interface specified by the *W3C*. Generally speaking, the *XML* markups correspond to *Nodes* of the *Data Model*, their attributes to *Attributes* of the *Data Model* and the textual content enclosed in the *XML* markups attributes quotes are bound to *Atomic Values* of the *Data Model*.

Structured Data File (SDF)	*SDF* is a critical implementation for accessing scientific or legacy data: it wraps around binary or *ASCII* files content according to an external description of the data structure. This external description is based on *XML Schema* documents, following a *W3C* recommendation. *SDF* makes use of set of extra markups to achieve the complete description of binary data representation, when this information could not be derived from the standard *XML Schema* markups: the *SDF* descriptors remain however fully compatible with the standard. These *XML Schema*'s are part of the *Representation Information* described in section 1.2, "Amalfi representation information" below. Amalfi standard distribution includes a large set of *XML Schema*'s that describe hundreds of data types.
FTP	The *FTP* implementation wraps around the remote file system exposed through *File Transfer Protocol (FTP)*. It binds directories and files as *Nodes* of the *Data Model*. Its behavior is similar to the *File* implementation described above but for remote data.
ZIP	The *ZIP* implementation wraps around *ZIP* file archives to bind their entries as *Nodes* of the *Data Model*. The complete directory structure is expanded instead of modeling a flat list of *ZIP* file entries.
TAR	Same as *ZIP* implementation for *Tape ARchive (TAR)* archives.
JAR	Same as *ZIP* implementation for *Java ARchive (JAR)* archives.

table 30 - DRB API® data model bindings made available with the Amalfi standard distribution

This version of *DRB API®* also supports the following *Atomic Value* types:

DRB Implementation	Array	Unsigned
`Binary`	N/A	N/A
`Boolean`	Yes	N/A
`Byte`	Yes	Yes
`DateTime`	No	N/A
`DayTimeDuration`	No	N/A
`Decimal`	No	N/A
`Double`	Yes	N/A
`Duration`	No	N/A

Float	Yes	N/A
Int	Yes	Yes
Integer	No	N/A
Long	Yes	Yes
Short	Yes	Yes
String	No	N/A
YearMonthDuration	No	N/A

table 31 - DRB API® atomic value types

The number of implementations and *Representation Information* available along with *DRB API®* constantly increases among its releases: at the time of writing *DRB API®* already includes new implementations as those supporting *HDF*, *NetCDF* or *VPF* files, and additional *Representation Information* packages are already available to support thousands of additional types; these are currently mainly focused on *Earth Observation* and *Geographical Information System (GIS)* data types, as those supporting *QuickBird*, *RADARSAT*, *NITF*, *DTED*, *DNG*, etc. data products, but *DRB API®* is not tied down to these domains: other data types from different domains will arise in the next future.

Please refer to http://www.gael.fr/drb Web site to learn more about *DRB API®* or to check if your data type is already available.

Amalfi Representation Information

Binding the data to a unified model is actually a critical concept, but Amalfi also includes a significant novelty with respect to its previous versions and predecessors: a *Representation Information* based on an *Ontology Model* connected to the *Data Model*.

Actually, and for understandable reasons, the data rarely includes the necessary information for its complete processing or *Quality Control*; some additional information about these data has to be retrieved and attached from elsewhere: the *Representation Information*, as clearly defined by the *OAIS Reference Model* described in [CCSDS-OAIS] document. This *Representation Information* brings meanings to the data, such as by providing structural information about the data content, or providing semantic about the data.

The important information required by Amalfi is the classification of the data, potentially hierarchically organized, and more significantly, the definition of the inspections that have to be performed for controlling their quality. Secondary information, mainly dealing with presentation, is also part of the Amalfi *Representation Information*, such as the metadata extraction means, the image access methods if relevant, or any other style-sheet that could enhance the software interfaces and reports.

The second clue element of the architectural design of Amalfi, is a federation of all resources around a unified *Ontology Model*. This latter is described in one or more *OWL* files, basically *XML* documents

following the *Ontology Web Language (OWL)* recommended by the *W3C*. Those *OWL* files can be scattered into several pieces that contribute to the model.

The interests of having an *Ontology Model*'s definition scattered in several *Topics* is at least twofold. In a first hand, it allows the development and maintenance of pieces of *Ontology* broken down thematically rather than sealed in a monolithic definition. They can be separated according to a category of data or a context of use, and their configuration can be individually controlled. For instance, the standard distribution of Amalfi includes a support of *ALOS* satellite data. It would have been correct to provide this support as a single *Topic*, but the distribution goes beyond and procures three *Topics* instead: one for *CEOS* data types (an abstract superstructure format followed by *ALOS* products and shared with many others), another for *ALOS* product types and last one including the inspections and other resources specific to the support of *ALOS* in the Amalfi context. This breakdown minimizes the duplicated resources, and fosters the reuse of components between the applications: thus, Amalfi simultaneously offers other applications to benefit from its contributions without imposing unnecessary resources (inspection definitions are segregated from the main *ALOS Topic*) and is offered the opportunity to receive back the improvements from those other applications that follow the same policy.

Amalfi can automatically discover the *OWL* files from multiple *JAR* archive files and compose a single *Ontology Model* for the application. Those *JAR* archive files may contain any other resources that generally contribute to the targeted *Representation Information*: each *JAR* archive is called a *Topic* in the *DRB Cortex Extension* vocabulary.

Describing further the concepts of *Representation Information* and *Ontology Modeling* is out of the scope of the present book, but further information about how they are implemented in Amalfi distribution; and how they could be configured is provided in section 6.3, “Ontology add-ons”.

Again, the architectural design and development of Amalfi has reused a *DRB API®* component to connect the *Data Model* to an *Ontology Model*: the so called *DRB Cortex Extension*.

Appendix – System Requirements

Amalfi software requires your computing environment to meet some minimum characteristics.

System requirements

- **Operating Systems**: all supporting Java 6 (see special note and table 32 -)
- **CPU**: equivalent to 500 MHz Pentium 3 or higher speed
- **System Memory (RAM)**: 512 Mb free minimum, 1024 Mb or greater free RAM recommended for extensive use
- **Screen**: 1024x768, "16 bit High Color" screen
- **Network**: required only for deployed installations, 128 Kbits/s recommended

Because all components of the Amalfi software are based on Java™, the supported Operating Systems are those required for installing a Java™ Virtual Machine. table 6, “Supported operating systems and desktops” - below provides the list of Operating Systems and Desktop Managers supported by Java according to SUN Microsystems at the time of writing. For up to date information, report to he following Web page:

http://java.sun.com/javase/6/webnotes/install/system-configurations.html

Rightmost column of table 32 -, denotes whether the Amalfi software has been tested on the Operating System.

Platform	Version	Desktop Managers	Tested
SolarisTM Operating System, 32-bit and 64-bit			
Solaris Sparc (32)	Solaris 10	JDS-2 (Gnome-Metacity) CDE-dtwm	×
	Solaris 9	Gnome-Metacity 2.4.34 or later CDE-dtwm	×
	Solaris 8	CDE-dtwm, Openwin-olwm	×
Solaris Sparc (64)	Solaris 10	JDS-Gnome-Metacity CDE-dtwm	×
	Solaris 9	Gnome-Metacity 2.4.34 or later, CDE	×
	Solaris 8	CDE-dtwm, Openwin-olwm	×

Solaris x86 (32)	Solaris 10	Gnome-Metacity, CDE	×
	Solaris 9	Gnome-Metacity, CDE	×
	Solaris 8	CDE, Openwin	×
Solaris x64 (64)	Solaris 10	JDS-Gnome-Metacity	×
	Windows 32-bit		
Windows	Windows XP Professional	Windows/Active for Windows	×
Intel IA32	Windows XP Home		✓
	Windows Server 2003		×
	Windows 2000 Professional		✓
	Windows 2000 Server		×
	Windows Vista		✓
	Windows 64-bit		
Windows x64	Windows XP	Windows/Active for Windows	×
32-bit mode	Windows Server 2003		×
	Windows Vista		×
Windows x64	Windows XP	Windows/Active for Windows	×
64-bit mode	Windows Server 2003		×
	Windows Vista		×
	Linux 32-bit		
Linux IA32	Red Hat 2.1, Red Hat Enterprise Linux 3.0, 4.0, 5.0	Gnome1.4-sawfish 1.0 or later Gnome 2.2-metacity 2.4 or later	×
	Suse Enterprise Linux Server 8, Suse Enterprise Linux Server 9, Suse Enterprise Linux Server 10, Suse Enterprise Linux Desktop	Gnome2.0.5-Metacity 2.6.2 or later (default: 2.4)	✓
	Turbo Linux 10 (ONLY Chinese and Japanese Locale. No english.)	Gnome-sawfish 1.0 or later	×

Linux 64-bit			
Linux x64 32-bit mode	Suse Enterprise Linux Server 8, Suse Enterprise Linux Server 9, Suse Enterprise Linux Server 10, Suse Enterprise Linux Desktop	Gnome2.0.5-Metacity 2.6.2 or later (default: 2.4)	×
	Red Hat Enterprise Linux 3.0, 4.0, 5.0	Gnome2.0.5-Metacity 2.6.2 or later (default: 2.4)	×
		Gnome-sawfish 1.0 or later	×
Linux x64 64-bit mode	Suse Enterprise Linux Server 8, Suse Enterprise Linux Server 9, Suse Enterprise Linux Server 10, Suse Enterprise Linux Desktop	Gnome2.0.5-Metacity 2.6.2 or later (default: 2.4)	×
	Red Hat Enterprise Linux 3.0, 4.0, 5.0	Gnome 2.2 - metacity 2.4 or later	×

table 32 - Supported operating systems and desktops

Software requirements

Amalfi software requires Java™ 6 Virtual Machine (JVM6) installed or higher version. The JVM6 may be run either under a Java™ Runtime Environment (JRE) (recommended for non-developing users) or Java™ Software Development Kit (JSDK). Visit http://java.sun.com/javase/downloads/index.jsp for downloading or further information.

Optional software

All optional software is automatically installed according to the user's selection made during installation procedure.

Appendix – Sample Configuration File

```
<configuration id="sample-configuration"
  xmlns="http://www.gael.fr/amalfi/core/configuration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <!-- Definition of the scope of this file -->
  <name>Example of Configuration file for Amalfi</name>
  <description>
    Defines configuration for compass application.
  </description>
  <url>example/amalfi-configuration.xml</url>

  <!-- any part of configuration can be imported as followed -->
  <import url="http://username:password@configuration_server/path/to/configuration/file"
cache="file:/local/path/to/file/used/for/cache"/>

  <!-- Definition of users -->
  <users>
    <!-- Default anonymous user - This user is required -->
    <user id="anonymous">
      <name>Anonymous User</name>
    </user>
    <user id="queue-submitter">
      <name>Inspection Queue Submitter</name>
      <mail>queue.submitter@amalfi.com</mail>
    </user>
    <user id="queue-administrator">
      <name>Inspection Queue Administrator</name>
      <mail>queue.administrator@amalfi.com</mail>
      <password algorithm="MD5">
        3cc8588bc6fb10cf0fd5907832f1cd8d
      </password>
    </user>
    <user id="database-submitter">
      <name>Database Submitter</name>
      <mail>database.submitter@amalfi.com</mail>
      <password algorithm="SHA">
        not_inserted_sha1_encrypted_password
      </password>
    </user>
    <user id="database-administrator">
      <name>Database Administrator</name>
      <mail>database.administrator@amalfi.com</mail>
      <password>not_encrypted_password</password>
    </user>
  </users>

  <!-- Definition of groups -->
  <groups>
    <group id="queue">
      <member>queue-submitter</member>
      <member>queue-administrator</member>
    </group>
    <group id="database">
      <member type="user">database-submitter</member>
      <member type="user">database-administrator</member>
    </group>
    <group id="administrators">
      <member type="user">queue-administrator</member>
      <member type="user">database-administrator</member>
    </group>
    <group id="users">
      <member type="group">queue</member>
      <member type="group">database</member>
      <member type="user">anonymous</member>
    </group>
  </groups>
  <!-- Definition of resources -->
  <resources>
    <!-- An example of inspection queue resource definition. -->
    <resource xsi:type="QueueResource" id="default-queue">
```

```
<name>Default Inspection Queue</name>
<description>
  This default inspection queue is mandatory in the configuration to manage not
  configured queues.
</description>
<url>rmi://localhost:1099/fr.gael.amalfi.server.Server</url>

<!-- Definition of privileges: who could access this queue? -->
<privileges>
  <!-- All members of the 'users' group can submit an inspection -->
  <privilege id="submit">
    <policy>
      <!-- By default nobody can submit an inspection -->
      <deny>
        <all />
      </deny>
      <!-- With the exception of members of the group 'users' -->
      <allow>
        <group>users</group>
        <user>anonymous</user>
      </allow>
    </policy>
  </privilege>

  <!-- Only members of the group 'administrators' could shutdown this queue, with only
       one exception -->
  <privilege id="shutdown">
    <policy>
      <deny>
        <all />
      </deny>
      <allow>
        <group>administrators</group>
        <!-- A special grant to Scott Wallace -->
        <user>anonymous</user>
      </allow>
    </policy>
  </privilege>
</privileges>

<!-- Definition of services attached to this queue -->
<services>
  <!-- A report broadcaster configured for sending mails -->
  <service id="reporting-broadcaster"
    xsi:type="ReportBroadcasterService">
    <!-- Optional filtering of input. It is possible to filter input item class,
         inspection status and inspection event. If no entry is defined, it means that
         the filter is always true for this section. For example, entering no item class
         means that the filter patched all the submitted items. Or setting no event will
         generates a mail at all the steps of the inspection. So it is recommended, at
         least, to select the inspection event and status.
         Possible events are (non case sensitive):
             - inspection_ran
             - inspection_done
             - inspection_canceled
             - queue_inspection_added
             - queue_inspection_removed
             - queue_inspection_shutdown
         Possible status are (non case sensitive):
             - Pending
             - Running
             - Done
             - Canceled
             - Passed
             - Failed
             - Error
             - Undefined -->
    <includes>
      <itemClass>http://www.esa.int/envisat#product</itemClass>
      <inspectionStatus>error</inspectionStatus>
      <inspectionStatus>failed</inspectionStatus>
    </includes>

    <!-- Selects an broadcasting via e-mail -->
    <broadcaster type="email">
      <!-- Optional mailer configuration -->
```

```
            <mailer>
              <server>
                <hostname>smtp.domain.org</hostname>
                <port>25</port>
                <username>smtp-user</username>
                <password encrypted=”false”>smtp-password</password>
                <!-- Is your mailer using TLS encryption ? -->
                <TLS>true</TLS>
              </server>
              <from name="Amalfi Inspection Report">
                amalfi@amalfi.com
              </from>
            </mailer>
            <!-- To all administrators -->
            <to type="group">administrators</to>
            <!-- With again a privilege to Scott Wallace in copy -->
            <cc type="user">wallace</cc>
            <!-- With a direct blinded copy to a spy -->
            <bcc type="email">bond@sis.gov.uk</bcc>
            <!-- Emailed report's name format -->
            <emailReportName>%i-report-%d</emailReportName>
          </broadcaster>
        </service>

        <!-- A report broadcaster configured for sending results in database -->
        <service id="reporting-broadcaster" xsi:type="ReportBroadcasterService">
          <!-- Filters are common to all the services, please refere to the documentation
               defined here before.. -->
          <includes>
            <itemClass>http://www.esa.int/envisat#product</itemClass>
            <event>inspection_done</event>
          </includes>

          <!-- Selects an broadcasting via database -->
          <broadcaster type="database">
            <!-- database configuration -->
            <database identifier="MainDb">
              <connectionUserId>smith</connectionUserId>
              <originator>KIRUNA</originator>
            </database>
          </broadcaster>
        </service>

       <!-- A scanner service that searches files from my repository and submit them to this
            queue -->
        <service id="N1-AVNIR2-fileScanner"
          xsi:type="FileScannerService">
          <!-- Base location for scanning -->
          <baseUrl>file:/anywhere/repository</baseUrl>
          <!-- Some filtering for not querying the universe -->
          <nameMatch>.*\.N1</nameMatch>
          <nameMatch>ALAV2.*</nameMatch>
          <!-- The scanning period expressed in ISO format -->
          <scanningPeriod>PT1S</scanningPeriod>
          <!-- To search beyond the base location -->
          <recursive>true</recursive>
        </service>

        <!-- File broadcaster aims to save the produced report into the specified directory.--
>
        <service id="report-saver" xsi:type="ReportBroadcasterService">
          <includes>
            <itemClass>http://www.gael.fr/drb#item</itemClass>
            <event>inspection_done</event>
          </includes>
          <broadcaster type="file">
            <file>
              <url>file:/directory/where/store/reports/</url>
            </file>
            <!-- Report's name format -->
            <fileReportName>%i-report-%d</fileReportName>
          </broadcaster>
        </service>
      </services>
    </resource>
```



```
    <!--  Ontology Resources : 2 possible types of resources:
           - "maven" type section:
             Uses Maven POM (Project object Model) style definition. This declaration allows a
             maven-like mechanism able to Cache ontology an manage availability of the network
             thanks to an artifactId, a groupId and the ontology version.

           - "jar" type section:
             This jar type section gather ontologies to be used by amalfi core system. System
             will load jar or directory defined here using the given URL. No cache is
             performed. This section includes a set of ontologies, each ones can be configured
             with a path (URL styled) and a list of dependencies to be loaded in a same time.

             The path section allows pointing to a local or remote url:
                http://www.gael.fr/distribution/mylibrary.jar

                http://www.gael.fr/distribution/my_library_directory/
                file:/d:/libraries/mylibrary.jar
                file:/d:/libraries/mylibrary_directory.
             Note that if the path defines a local/remote directory a trailer "/" must be
             added.

             Dependencies tag gather a list of dependency refers this path.
             The syntax used for this paths are the same as the Ontologies paths.
             Each ontology can defines its name (optional attribute) Which is a text free
             attribute.

             The followInternalDependencies attribute allow the system to automatically
resolve
             the path dependencies in addition to the ones explicitly proposed in the
             dependency entries.

             The automatic resolution uses "Class-Path" entry of the jar manifest. If the
given
             URL points to a directory or a jar where is missing this manifest entry, no
             automatic dependencies will be resolved. The base URL used to resolve these
             dependencies is the path where is stored the parent jar. -->

    <!-- Ontology currently editing/debuging -->
    <resource xsi:type="OntologyResource" id="landsat" type="jar">
      <name>Amalfi LANDSAT Ontology resource</name>
      <description>
        The Ontologies used to decode and retrieve Amalfi tests for
        the overall LANDSAT products.
      </description>
      <url>http://www.gael.fr/drb#landsat</url>
      <configuration><![CDATA[

        <jar name="LANDSAT Inspections" followInternalDependencies="false"
          xmlns="http://www.gael.fr/classpath/resolver/jar/configuration">
          <path>
file:/home/pidancier/svn/trunk/amalfi/configuration/landsat/ceos/src/main/resources/
          </path>
          <dependencies>
            <dependency>
file:/home/pidancier/svn/trunk/drb/extensions/cortex/topics/landsat/src/main/resources/
            </dependency>
            <dependency>
file:/home/pidancier/svn/trunk/drb/extensions/cortex/topics/ceos/src/main/resources/
            </dependency>
          </dependencies>
        </jar>

      ]]></configuration>
    </resource>

    <!-- Addon's repositories definition -->
    <resource xsi:type="OntologyRepositoryResource" id="gael-main">
      <name>Main Gael Ontology server</name>
      <description>Gael's ontologies server.</description>
      <url>scp://repository.gael.fr/repository/software/distributions</url>
      <username>username</username>
      <password>password</password>
    </resource>

    <!-- ENVISAT addon -->
    <resource xsi:type="OntologyResource" id="envisat" type="maven">
```

```
      <name>Amalfi ENVISAT Ontology resource</name>
      <description>
        The Ontologies used to decode and retrieve Amalfi inspections for the overall ENVISAT
products.
      </description>
      <url>http://www.gael.fr/drb#envisat</url>
      <configuration><![CDATA[

        <project>
          <modelVersion>4.0.0</modelVersion>
          <groupId>fr.gael.amalfi</groupId>
          <artifactId>amalfi-config-envisat</artifactId>
          <packaging>jar</packaging>
          <version>1-0-rc-3</version>
        </project>

      ]]></configuration>
    </resource>

    <!-- Example of a referenced Amalfi Database -->
    <resource xsi:type="DatabaseResource" id="MainDb">
      <name>The Main Amalfi Database</name>
      <description>Main amalfi database used for this instance of installation.</description>
      <url>jdbc:mysql://localhost:3306/amalfi</url>
      <login>amalfi</login>
      <password encrypted="true">OXLyeUFV4eU=</password>
      <driver protocol="mysql">com.mysql.jdbc.Driver</driver>
      <privileges>
        <privilege id="submit">
          <policy>
            <deny>
              <all />
            </deny>
            <allow>
              <group>users</group>
            </allow>
          </policy>
        </privilege>
        <privilege id="query">
          <policy>
            <deny>
              <all />
            </deny>
            <allow>
              <user>smith</user>
            </allow>
          </policy>
          <parameters>
            <parameter id="itemClassFilter">http://www.gael.fr/drb#item</parameter>
          </parameters>
        </privilege>
      </privileges>
    </resource>

  <!-- Example of a Cron scheduler -->
    <resource xsi:type="CronResource" id="CronSceduler">
      <name>Amalfi Cron scheduler</name>
      <description>
        Scheduler used in amalfi to perform various periodic Treatments
      </description>
      <url>http://www.gael.fr/amalfi#schduler</url>
      <!-- All the jobs defined here requires amalfi-database module be installed and
           available. -->
      <cron compensation="0">
      <!-- Performs Purge (Report content only "full"=false) last one year (-P1Y) of inputs in
           database at a period of all the years in January, 5th at 00h00 ("00 00 05 01 *") --
>
        <job id="purge" name="Perfoming database Purge" schedule="00 00 05 01 *"
          disabled="false" startup="false">
          <class name="fr.gael.amalfi.database.service.DatabasePurge">
            <param value="jdbc:mysql://localhost:3306/amalfi" name="url"/>
            <param name="db_login" value="amalfi"/>
            <param name="db_password" value="OXLyeUFV4eU="/>
            <param name="db_encrypted" value="true"/>
            <param name="user" value="anonymous" />
            <param name="since" value="-P1Y" />
```

```
            <param name="to" value="PT0S" />
            <param name="full" value="false" />
          </class>
        </job>

        <job id="clean-tmp" name="Clean temporary files" schedule="00 18 * * *"
          disabled="false" startup="false">
          <class name="fr.gael.amalfi.database.service.SystemCommand">
            <param name="param0" value="rm" />
            <param name="param1" value="*" />
            <param name="dir" value="/tmp" />
            <param name="fork" value="true" />
          </class>
        </job>

        <job id="synchronize-laptop" name="Synchronize database from laptop"
          schedule="0 0 * * 0" disabled="false" startup="false">
          <class name="fr.gael.amalfi.database.service.DatabaseSynchronize">
            <param name="from_url" value="jdbc:mysql://192.168.200.7:3306/amalfi" />
            <param name="from_login" value="amalfi"/>
            <param name="from_password" value="OXLyeUFV4eU="/>
            <param name="from_encrypted" value="true"/>
            <param name="to_url" value="jdbc:mysql://localhost:3306/amalfi" />
            <param name="to_login" value="amalfi"/>
            <param name="to_password" value="OXLyeUFV4eU="/>
            <param name="to_encrypted" value="true"/>
          </class>
        </job>
      </cron>
    </resource>
  </resources>
</configuration>
```

Appendix – Configuration File XML Schema (normative)

```
<?xml version="1.0" encoding="utf-8" ?>

<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns="http://www.gael.fr/amalfi/core/configuration"
           xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
           jaxb:version="1.0" elementFormDefault="qualified"
           targetNamespace="http://www.gael.fr/amalfi/core/configuration"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="Import">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded"
        name="import-option">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="url" type="xs:string" />
    <xs:attribute name="cache" type="xs:string" />
  </xs:complexType>

  <xs:complexType name="User">
    <xs:annotation>
      <xs:documentation>
        The user is used to manage the authentication and privileges inside
        Amalfi core system. It is defined by an unique identifier, a name,
        an e-mail address and the password used for the amlfi core resources authentification.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element minOccurs="0" name="mail" type="xs:string" />
      <xs:element minOccurs="0" name="password">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="algorithm" use="optional">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="MD2" />
                    <xs:enumeration value="MD5" />
                    <xs:enumeration value="SHA" />
                    <xs:enumeration value="SHA-256" />
                    <xs:enumeration value="SHA-384" />
                    <xs:enumeration value="SHA-512" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>

  <xs:complexType name="Group">
    <xs:annotation>
      <xs:documentation>
        The group gather a set of member that can be users or other groups.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="member">
        <xs:complexType>
```

```
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="type" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="user" />
                  <xs:enumeration value="group" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="Service" abstract="true">
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType xsi:type="Service" name="FileScannerService">
  <xs:complexContent mixed="false">
    <xs:extension base="Service">
      <xs:annotation>
        <xs:documentation>
          The file scanner service periodically scans the provided baseUrl and
          reports added, changed and removed files of this path. The sequence
          of patterns called "nameMatch" allows fine tunning of the files to
          be taken into account.
        </xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="baseUrl" type="xs:string" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="nameMatch" type="xs:string" />
        <xs:element name="scanningPeriod" type="xs:duration" />
        <xs:element name="recursive" type="xs:boolean" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType xsi:type="Service" name="ReportBroadcasterService">
  <xs:complexContent mixed="false">
    <xs:extension base="Service">
      <xs:annotation>
        <xs:documentation>
          Configuration definition used to defines the report broadcaster service. In the
          case of the broadcaster type is "email", it is necessary to define the mailer
          service resources. Includes and excludes patterns allows fine tuning of the
          broadcasts activation events.
        </xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element minOccurs="0" name="includes" type="ReportFilter" maxOccurs="1" />
        <xs:element minOccurs="0" name="excludes" type="ReportFilter" maxOccurs="1" />
        <xs:element name="broadcaster" maxOccurs="1" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:sequence>
                <xs:element name="mailer" maxOccurs="1" minOccurs="0">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="server">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="hostname" type="xs:string"/>
                            <xs:element minOccurs="0" maxOccurs="1" name="port"
                              type="xs:unsignedShort"/>
                            <xs:element minOccurs="0" maxOccurs="1" name="username"
                              type="xs:string"/>
                            <xs:element minOccurs="0" maxOccurs="1" name="password">
                              <xs:complexType>
                                <xs:simpleContent>
                                  <xs:extension base="xs:string">
```

```
                        <xs:attribute name="encrypted" type="xs:boolean" />
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
                <xs:element name="TLS" type="xs:boolean" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="from">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="name" type="xs:string"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
          <xs:element minOccurs="0" name="replyTo" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="to">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute default="email" name="type">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="email" />
                  <xs:enumeration value="user" />
                  <xs:enumeration value="group" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="cc">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute default="email" name="type">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="email" />
                  <xs:enumeration value="user" />
                  <xs:enumeration value="group" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="bcc">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute default="email" name="type">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="email" />
                  <xs:enumeration value="user" />
                  <xs:enumeration value="group" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="emailReportName" type="xs:string" />
  </xs:sequence>
```

```
              <xs:sequence>
                <xs:element name="database" maxOccurs="1" minOccurs="0">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="connectionUserId" type="xs:string" />
                      <xs:element name="originator" type="xs:string" />
                    </xs:sequence>
                    <xs:attribute name="identifier" type="xs:string" />
                  </xs:complexType>
                </xs:element>
              </xs:sequence>

              <xs:sequence>
                <xs:element name="file" maxOccurs="1" minOccurs="0">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="url" type="xs:string" />
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="fileReportName" type="xs:string" />
              </xs:sequence>
            </xs:sequence>
            <xs:attribute default="email" name="type">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="email" />
                  <xs:enumeration value="database" />
                  <xs:enumeration value="file" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Privilege">
  <xs:sequence>
    <xs:element name="policy">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" maxOccurs="1" name="deny" type="RightsPolicy" />
          <xs:element minOccurs="0" maxOccurs="1" name="allow" type="RightsPolicy" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element minOccurs="0" name="parameters">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" maxOccurs="unbounded" name="parameter">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="id" type="xs:string" />
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="query" />
        <xs:enumeration value="submit" />
        <xs:enumeration value="shutdown" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="scope" type="xs:string" />
</xs:complexType>
```

```
<xs:complexType name="RightsPolicy">
  <xs:annotation>
    <xs:documentation>
      Defines the users and groups rights to be attached to the resources. These rights
      policy might be used to allow or deny access to resources. Scope of the policy covers
      "users" (as defined in the "users" section), "group" (as defined in the "groups"
      section) or all. Choosing "all" means the all other policy in this context will not be
      taken into account.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence maxOccurs="unbounded" minOccurs="0">
    <xs:element minOccurs="0" name="all" />
    <xs:element minOccurs="0" maxOccurs="unbounded" name="user" type="xs:string" />
     <xs:element minOccurs="0" maxOccurs="unbounded" name="group" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ReportFilter">
  <xs:annotation>
    <xs:documentation>
      The report filter is used to manage fine tuning of broadcasting Event when the context
      matches (includes) or not matches (excludes) these filters.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="itemClass" type="xs:string" />
    <xs:element minOccurs="0" maxOccurs="unbounded" name="inspectionStatus">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="passed" />
          <xs:enumeration value="failed" />
          <xs:enumeration value="error" />
          <xs:enumeration value="pending" />
          <xs:enumeration value="done" />
          <xs:enumeration value="canceled" />
          <xs:enumeration value="undefined" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="event">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="inspection_ran" />
          <xs:enumeration value="inspection_cancel" />
          <xs:enumeration value="inspection_done" />
          <xs:enumeration value="queue_inspection_added" />
          <xs:enumeration value="queue_inspection_removed" />
          <xs:enumeration value="queue_shutdown" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Resource" abstract="true">
  <xs:annotation>
    <xs:documentation>*
      Resources are all the resources configured in Amalfi core application such as
      inspection queues. Each of these resources are recognized tanks to its identifier.
      Services can be configured other these resources with "services" entries, and
      users/groups privileges are managed with "privileges" entries.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="description" type="xs:string" />
    <xs:element name="url" type="xs:string" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType xsi:type="Resource" name="QueueResource">
  <xs:complexContent mixed="false">
    <xs:extension base="Resource">
      <xs:sequence>
```

```
        <xs:element minOccurs="0" name="privileges">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="privilege">
                <xs:complexType>
                  <xs:complexContent mixed="false">
                    <xs:extension base="Privilege" />
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element minOccurs="0" name="services">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="service" type="Service" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType xsi:type="Resource" name="DatabaseResource">
  <xs:complexContent mixed="false">
    <xs:extension base="Resource">
      <xs:sequence>
        <xs:element name="login" type="xs:string" />
        <xs:element name="password">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="encrypted" type="xs:boolean" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="driver">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="protocol" use="optional">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="mysql" />
                    </xs:restriction>
                  </xs:simpleType>
                </xs:attribute>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element minOccurs="0" name="privileges">
          <xs:annotation>
            <xs:documentation>
              defines the amalfi users database access privileges.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="privilege"
                type="Privilege" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<xs:complexType xsi:type="Resource" name="OntologyResource">
  <xs:complexContent mixed="false">
    <xs:extension base="Resource">
```

```
      <xs:sequence>
        <xs:element name="configuration" type="xs:string" maxOccurs="1" minOccurs="1" />
      </xs:sequence>
      <xs:attribute name="type">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="jar" />
            <xs:enumeration value="maven" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType xsi:type="Resource" name="OntologyRepositoryResource">
  <xs:complexContent mixed="false">
    <xs:extension base="Resource">
      <xs:choice>
        <xs:sequence>
          <xs:element name="username" type="xs:string" maxOccurs="1" minOccurs="0" />
          <xs:element name="password" type="xs:string" maxOccurs="1" minOccurs="0" />
        </xs:sequence>
        <xs:sequence>
          <xs:element name="privateKey" type="xs:string" maxOccurs="1" minOccurs="0" />
          <xs:element name="passphrase" type="xs:string" maxOccurs="1" minOccurs="0" />
        </xs:sequence>
      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType xsi:type="Resource" name="CronResource">
  <xs:complexContent mixed="false">
    <xs:extension base="Resource">
      <xs:sequence>
        <xs:element name="cron">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="job">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element maxOccurs="unbounded" name="class">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element minOccurs="0" maxOccurs="unbounded" name="param">
                            <xs:complexType>
                              <xs:attribute name="name" type="xs:string" use="required" />
                              <xs:attribute name="value" type="xs:string" use="optional" />
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:string" use="required" />
                        <xs:attribute default="false" name="continuable"
                           type="xs:boolean" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:string" use="required"/>
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="schedule" type="xs:string" use="required" />
                  <xs:attribute default="false" name="disabled" type="xs:boolean" />
                  <xs:attribute default="false" name="startup" type="xs:boolean" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="compensation" type="xs:long" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Configuration">
  <xs:annotation>
```

```
  <xs:documentation>
    Configuration contains all informations used to run amalfi system. The configuration
    gather users definition, groups definition, resources list and their services and
    privileges.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="name" type="xs:string">
    <xs:annotation>
      <xs:documentation>The name given to this configuration.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="description" type="xs:string">
    <xs:annotation>
      <xs:documentation>A description of this configuration.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="url" type="xs:string" />
  <xs:element minOccurs="0" maxOccurs="unbounded" name="import">
    <xs:annotation>
      <xs:appinfo>Not implemented.</xs:appinfo>
      <xs:documentation>
        Allows to import other configuration inside this configuration. Imported
        groups and users are allowed to access local resources, but local users are
        not allowed to access imported resources. The import url can be
        an XML configuration file. Importing an XML configuration can be done as followed:
                <import url="file://path/to/configuration.xml" />
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:extension base="Import" />
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="users" maxOccurs="1" minOccurs="1">
    <xs:annotation>
      <xs:documentation>
        List of users that may have special privileges  within the different resources
        defined in this configuration. A user is always defined by its unique identifier,
        a common name and an e-mail. To manage user authentication other privileges, it is
        possible to defines a user password. The system supports a large set of encryption
        algorithms to preserve passwords privacy.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="unbounded" name="user">
          <xs:complexType>
            <xs:complexContent mixed="false">
              <xs:extension base="User" />
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
   <xs:element name="groups" maxOccurs="1" minOccurs="0">
    <xs:annotation>
      <xs:documentation>
        Groups gather all the groups configured and managed by this system. A group is a
        set of members (users or groups). Each group is defined by its unique identifier.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="group">
          <xs:complexType>
            <xs:complexContent mixed="false">
              <xs:extension base="Group" />
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
      <xs:element name="resources" maxOccurs="1" minOccurs="1">
        <xs:annotation>
          <xs:documentation />
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" name="resource" type="Resource" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" />
  </xs:complexType>
  <xs:element name="configuration" type="Configuration"></xs:element>

</xs:schema>
```

Appendix – Amalfi Report XML schema (normative)

```
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   xmlns="http://www.gael.fr/amalfi"
   xmlns:amalfi="http://www.gael.fr/amalfi"
   targetNamespace="http://www.gael.fr/amalfi">

   <xs:element name="report" type="amalfi:ReportType"/>

   <xs:complexType name="ReportType">
      <xs:sequence>
         <xs:element name="item" type="ItemType"/>
         <xs:element minOccurs="0" maxOccurs="unbounded"
            name="inspection" type="InspectionType"/>
      </xs:sequence>
      <xs:attribute name="date" type="xs:dateTime" />
   </xs:complexType>

   <xs:complexType name="ItemType">
      <xs:sequence>
         <xs:element name="description" type="xs:string"/>
         <xs:element name="metadataSet">
            <xs:complexType>
            <xs:sequence>
               <xs:element name="metadata">
                  <xs:complexType>
                  <xs:sequence>
                     <xs:any namespace="##other" processContents="strict"/>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string"/>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>

            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="class" type="xs:string" />
      <xs:attribute name="className" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="url" type="xs:string" />
   </xs:complexType>


   <xs:complexType name="InspectionType">
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded"
            name="message" type="xs:string"/>
         <xs:element minOccurs="0" maxOccurs="unbounded"
            name="inspection" type="InspectionType"/>
      </xs:sequence>
      <xs:attribute name="creation" type="xs:dateTime" />
      <xs:attribute name="duration" type="xs:duration" />
      <xs:attribute name="execution" type="xs:dateTime" />
      <xs:attribute name="item" type="xs:string" />
      <xs:attribute name="itemUrl" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="priority" type="xs:unsignedInt" />
      <xs:attribute name="processingStatus">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="Done" />
               <xs:enumeration value="Pending" />
               <xs:enumeration value="Running" />
               <xs:enumeration value="Canceled" />
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="status">
         <xs:simpleType>
```

```
            <xs:restriction base="xs:string">
               <xs:enumeration value="Passed" />
               <xs:enumeration value="Failed" />
               <xs:enumeration value="Error" />
               <xs:enumeration value="Undefined" />
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>

   <xs:complexType name="MessageType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="contentType" type="xs:string" />
         </xs:extension>
       </xs:simpleContent>
   </xs:complexType>

      <xs:complexType name="StatusType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="contentType" type="xs:string" />
         </xs:extension>
       </xs:simpleContent>
   </xs:complexType>

</xs:schema>
```

Appendix – License Terms

These license terms are an agreement between GAEL Consultant (or its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any.

The terms also apply to any GAEL Consultant

- updates,
- supplements,
- Internet-based services, and
- support services

for this software, unless other terms accompany those items. If so, those terms apply.

BY USING THE SOFTWARE, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT USE THE SOFTWARE.

If you comply with these license terms, you have the rights below.

Installation rules and rights

General

You may install and use any number of copies of the software on your devices.

Distribution

You may copy and distribute the software, provided that:

- each copy is complete and unmodified, including presentation of this agreement for each user's acceptance; and
- you indemnify, defend, and hold harmless GAEL Consultant and its affiliates and suppliers from any claims, including attorneys' fees, related to your distribution of the software.

You may not:

- distribute the software with any non-GAEL Consultant software that may use the software to enhance its functionality,
- alter any copyright, trademark or patent notices in the software,
- use GAEL Consultant's or affiliates or suppliers' name, logo or trademarks to market your products or services,
- distribute the software with malicious, deceptive or unlawful programs, or
- modify or distribute the software so that any part of it becomes subject to an Excluded License. An Excluded License is one that requires, as a condition of use, modification or distribution, that
- the code be disclosed or distributed in source code form; or

- others have the right to modify it.

Scope of the license

The software is licensed, not sold. This agreement only gives you some rights to use the software. GAEL Consultant reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the software only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the software that only allow you to use it in certain ways.

You may not

- work around any technical limitations in the software;
- reverse engineer, decompile or disassemble the software, except and only to the extent that applicable law expressly permits, despite this limitation;
- make more copies of the software than specified in this agreement or allowed by applicable law, despite this limitation;
- publish the software for others to copy;
- rent, lease or lend the software; or
- use the software for commercial software hosting services.

Backup and copy

You may make one backup copy of the software. You may use it only to reinstall the software.

Documentation

Any person that has valid access to your computer or internal network may copy and use the documentation for your internal, reference purposes.

Transfer to another device

You may uninstall the software and install it on another device for your use. You may not do so to share this license between devices.

Export restrictions

The software is subject to French and European export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the software. These laws include restrictions on destinations, end users and end use.

Support services

Because this software is "as is," we may not provide support services for it.

Entire agreement

This agreement, and the terms for supplements, updates, Internet-based services and support services that you use, are the entire agreement for the software and support services.

Applicable law

The French laws apply.

Disclaimer of warranty

The software is licensed "as-is." You bear the risk of using it. GAEL Consultant gives no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this agreement cannot change. To the extent permitted under your local laws, GAEL Consultant excludes the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

Limitation on and exclusion of remedies and damages

To the extent not prohibited by applicable law, in no event shall GAEL Consultant liable for personal injury, or any incidental, special, indirect or consequential damages whatsoever, including, without limitation, damages for loss of profits, loss of data, business interruption and any other commercial damages or losses, arising out of or related to your use or inability to use the GAEL Consultant Software, however caused, regardless of the theory of liability (contract, tort or otherwise) and even if GAEL Consultant has been advised of the possibility of such damages. Some jurisdictions do not allow the limitation of liability for personal injury, or of incidental or consequential damages, so this limitation may not apply to you. In no event shall GAEL Consultant's total liability to you for all damages (other than as may be required by applicable law in cases involving personal injury) exceed the amount of five Euros (5.00 €). The foregoing limitations will apply even if the above stated remedy fails of its essential purpose.

Acronyms and Abbreviations

ASCII	American Standard Code for Information Interchange
CCSDS	Consultative Committee for Space Data Systems
DOM	Document Object Model
DRB	Data Request Broker
EBV	Effective Boolean Value
FTP	File Transfer Protocol
ISO	International Organization for Standardization
JAR	Java ARchive
OWL	Web Ontology Language
POM	Project Object Model
RDF	Resource Description Framework
RPM	RedHat Package Manager
SDF	Structured Data File
TAR	Tape ARchive
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XSD	XML Schema Description
ZIP	Compression format

Glossary of Terms

add-on	Amalfi package containing the entire configuration to perform dataset inspection.
Data	A reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing. Examples of data include a sequence of bits, a table of numbers, the characters on a page, the recording of sounds made by a person speaking, or a moon rock specimen - [CCSDS-OAIS]
DRB Cortex Extension	Drb Cortex is an extension to Drb able to provide semantic information inside Drb items.
Format	Format is a way of encoding data in a file.
Inspection	Conformity evaluation by observation and judgment accompanied as appropriate by measurement, testing or gauging - [ISO 9000:2000]
Item (Drb scope)	Item in Drb is the most abstract element managed by Drb. Existing implementations of items is Node, Value or Attribute.
Ontology	Ontology is usually shown as a graph defining relationship between simple element, and given semantic to.
Ontology Model	The model above an ontology. Drb Cortex API uses RDF/OWL model to provides semantics to drb items.
Quality Control	Part of quality management focused on fulfilling quality requirements - [ISO 9000:2000]
Physical Object	An object (such as a moon rock, bio-specimen, microscope slide) with physically observable properties that represent information that is considered suitable for being adequately documented for preservation, distribution, and independent usage - [CCSDS-OAIS]
Reporting	Returning a formatted set of results.
Representation Information	The information that maps a Data Object into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e., a Data Object) is mapped into a symbol - [CCSDS-OAIS]
Result Persistency	A way of keeping result. (database, mail ...)
Structure Information	The information that imparts meaning about how other information is organized. For example, it maps bit streams to common computer

	types such as characters, numbers, and pixels and aggregations of those types such as character strings and arrays - [CCSDS-OAIS]
Topic	Drb package containing configuration able to decode dataset.

Bibliography

[CCSDS-OAIS] Reference Model for an Open Archival Information System (OAIS). Recommendation for Space Data Systems Standards. January, 2002. Blue Book, Issue 1. Copyright © 2002 Consultative Committee for Space Data Systems (CCSDS).

[ISO 9000:2000] Quality management systems. Fundamentals and vocabulary. ISO Standard. September 20, 2005. Revision 2. Copyright © 2005 International Organization for Standardization (ISO).

[OWL] OWL Web Ontology Language: Reference. W3C Recommendation. February 10, 2004. Version 1.0. Copyright © 2004 World Wide Web Consortium (W3C).

[RDF] RDF/XML Syntax Specification (Revised). W3C Recommendation. February 10, 2004. Version 1.0. Copyright © 2004 World Wide Web Consortium (W3C).

[XML] eXtensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. October 6, 2000. Version 1.0. Copyright © 2000 World Wide Web Consortium (W3C).

[XML-SCHEMA] XML Schema: Primer. W3C Recommendation. May 2, 2001. Version 1.0. Copyright © 2001 World Wide Web Consortium (W3C).

[XML-SCHEMA-STRUCT] XML Schema: Structures. W3C Recommendation. May 2, 2001. Version 1.0. Copyright © 2001 World Wide Web Consortium (W3C).

[XML-SCHEMA-TYPES] XML Schema: Data Types. W3C Recommendation. May 2, 2001. Version 1.0. Copyright © 2001 World Wide Web Consortium (W3C).

[XQUERY] XQuery 1.0: An XML Query Language. W3C Recommendation. January 23, 2007. Version 1.0. Copyright © 2007 World Wide Web Consortium (W3C).